TraceX

Equipe de projet L2S1

Projet TraceX

Conception détaillée

Version du document :	B.05
Date de document :	20/05/2025
Soumis le :	20/05/2025
Auteurs :	ALLAHOUM Abdelmalek Said, HUANG Maxime, KIM Léa, ZHENG Jacques
Type de diffusion :	Document électronique (.pdf)
Confidentialité :	Réservé aux étudiants UFR Maths-Info de l'Université Paris Cité

Table des matières

1. Introduction

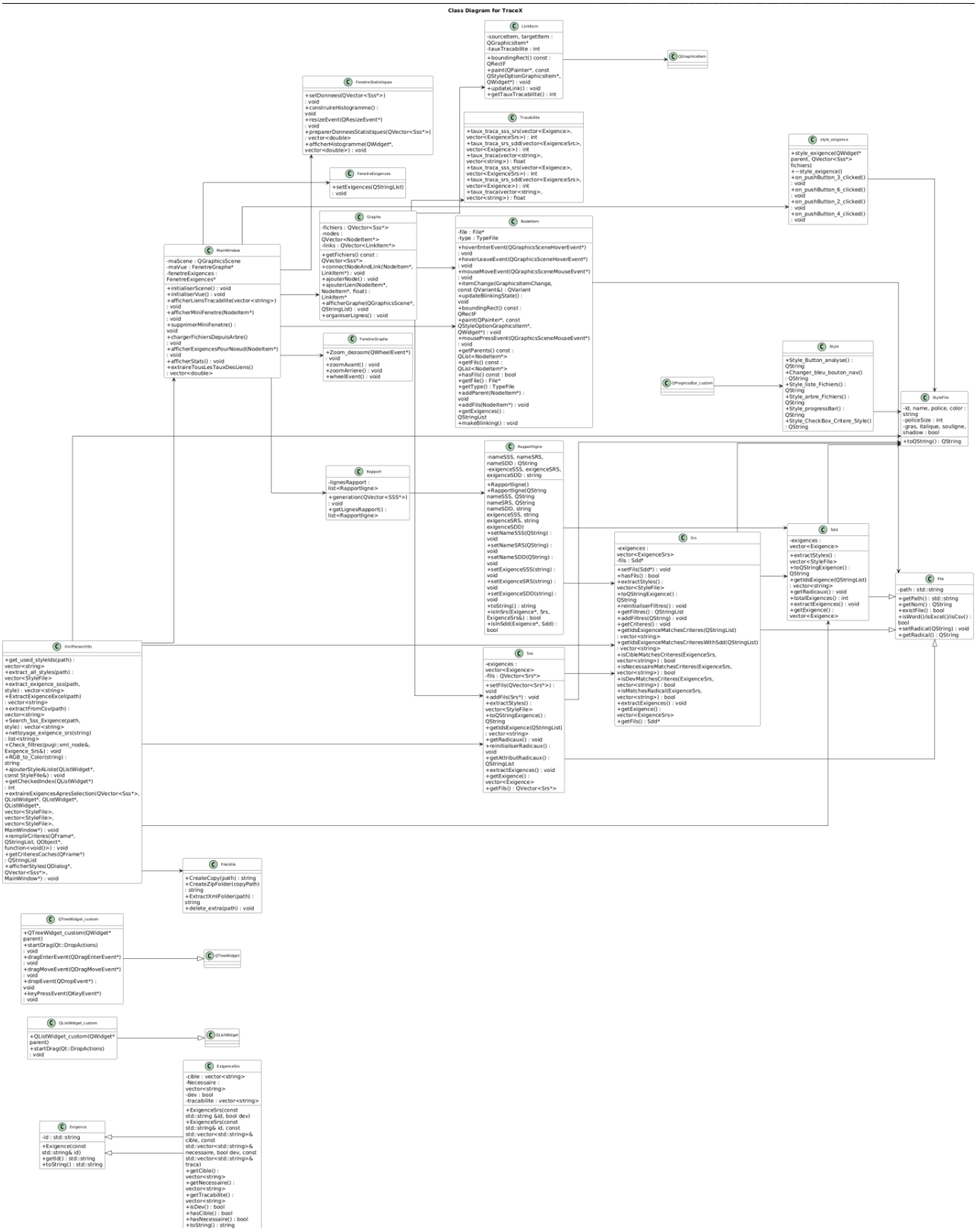
Ce document présente la conception détaillée qui affine la conception générale du projet d'analyse et de traçabilité des exigences. Il vise à définir précisément les modules, classes, méthodes et structures de données qui seront implémentés afin de garantir une réalisation efficace du projet et validé dans le cahier de recette.

2. Architecture Générale

L'application repose sur une architecture modulaire composée des modules suivants :

- **Module d'importation et d'analyse** : Chargé de lire et analyser les documents fournis (Word, Excel, CSV).
- **Module de traitement des exigences** : Responsable de l'extraction des exigences et de la vérification de leur présence dans les documents.
- **Module de visualisation** : Permet l'affichage des graphes de traçabilité.
- **Module de filtrage et statistiques** : Fournit des options de filtrage et calcule le taux de traçabilité.
- **Interface graphique** : Interagit avec l'utilisateur pour l'importation et la manipulation des données.

Diagramme de classe



3. Détails des Modules

La Classe MainWindow constitue le cœur de l'application : il centralise les principales fonctionnalités telles que l'importation des fichiers et l'affichage du graphe de traçabilité ainsi que les statistiques des liens de traçabilité.

Classe MainWindow

Nom	MainWindow	
Héritage	QMainWindow	
Attributs	ui: **Ui::MainWindow***	Une fenêtre de dialogue générée par Qt Designer.
	Analyse: **QPushButton***	Bouton d'analyse dans la barre de navigation.
	PAGE_HOME: **const int** PAGE_IMPORT: **const int** PAGE_HELP: **const int** PAGE_ANALYSE:	Constantes qui contiennent l'index de chaque pages de la navigation.
	const int	
	Fichier_upload: **QStringList**	Liste des fichiers importés avec le bouton "upload manually".
	nb_fichier_upload: **int**	Nombre de fichiers uploadés
	Fichier_extrait: **QVector<Sss*>**	Contient les fichiers extraits après analyse dans u(sous forme d'une arborescence).
	liste_fichiers: **QListWidget_custom***	Liste des fichiers affichés sur le tableau.
	arbre_fichiers: **QTreeWidget_custom***	Arbre de données représentant les relations des fichiers
	maVue: **FenetreGraphe***	Représente la fenêtre du graphe

	maScene: **QGraphicsScene**	Scène graphique pour le graphe
	Statistics **QPushButton***	Bouton de navigation vers la page "Statistiques"
	fenetreExigences **FenetreExigences***	Mini-fenêtre flottante affichant les exigences liées aux nœuds
	menuOpen **bool**	Indique si la barre latérale est ouverte ou fermée
	texteActuel **QString**	Texte affiché progressivement dans l'animation
	I **int**	Index courant dans l'animation lettre par lettre
	Timer QTimer*	Timer utilisé pour l'affichage dynamique du texte
Méthodes	MainWindow() (Constructeur)	Entrée: QWidget* parent : widget parent, initialisé à nullptr par défaut Sortie: void Explication: Initialise l'interface et ses widgets.
	~MainWindow() (Destructeur)	Entrée: / Sortie: / Explication: Libère la mémoire utilisée dynamiquement par "MainWindow".
	getFichierUpload()	Entrée:/ Sortie: QStringList& Explication: Retourne la liste des fichiers importés.
	resetFichierUpload()	Entrée:/ Sortie: void Explication: Vide la liste des fichiers uploadés

	afficherStats()	Entrée:/ Sortie: void Explication: Affiche l'histogramme des taux de traçabilité
	Bouton_graphe()	Entrée:/ Sortie: void Explication: Gère l'action du bouton "Graphe"
	afficherStats()	Entrée:/ Sortie: void Explication: Affiche l'histogramme des taux de traçabilité
	on_DeleteAll_clicked()	Entrée:/ Sortie: void Explication: Supprime tous les fichiers de la liste
	updateGraphWithCriteres()	Entrée:/ Sortie: void Explication: Met à jour le graphe en fonction des filtres cochés
	Ajout_Analyse_bouton()	Entrée: / Sortie: / Explication:Ajoute dynamiquement un bouton "Analyse" dans la barre de navigation.
	AnimationFonctions()	Entrée: / Sortie: / Explication: Lance les animations de démarrage de l'application
	Animation_afficherProchaineLettre(QString, QLabel)*	Entrée: / Sortie: / Explication: Affiche une phrase lettre par lettre dans un label
	on_BarreLateraleMenu_clicked()	Entrée: / Sortie: / Explication:Gère l'ouverture/fermeture de la barre latérale avec animation

	afficherExigencesPourNoeud(NodeItem)*	Entrée: / Sortie: / Explication: Affiche les exigences liées à un nœud du graphe cliqué
	resizeEvent(QResizeEvent*)	Entrée: / Sortie: / Explication : Repositionne dynamiquement la fenêtre des exigences
	getConteneurExigenceLayout()	Entrée: / Sortie: QVBoxLayout* Explication: Repositionne dynamiquement la fenêtre des exigences
	on_BoutonTelechargementCSV_clicked()	Entrée: / Sortie: / Explication : Gère l'export CSV du rapport de traçabilité
	fadeIn_Widget(QWidget, int, int)*	Entrée: / Sortie: / Explication: Anime l'apparition d'un
	Debug_extraction_fichier()	Entrée: / Sortie: / Explication: Affiche sur la console les fichiers extraits
	Afficher_fichier_upload()	Entrée: / Sortie: / Explication: Affiche sur le tableau les fichiers importés
	onAnalyseClicked()	Entrée: / Sortie: / Explication: Dirige l'utilisateur vers la page "Analyse".

	Reinitialise_bouton()	Entrée: / Sortie: / Explication: Change la couleur du bouton rentré en paramètre en bleu, change les autres boutons en blanc et redirige l'utilisateur vers la page concernée.
	newGraphWindow()	Entrée: / Sortie: / Explication: Retourne le bouton "Analyse".
	getAnalyse()	Entrée: / Sortie: QPushButton* Explication: Retourne le bouton "Analyse".
	getNav()	Entrée: / Sortie: QHBoxLayout* Explication: Retourne la barre de navigation sous forme de QHBoxLayout.
	ActualiserQSS(QWidget* widget)	Entrée: QWidget *widget Sortie: / Explication: Réapplique la feuille de style au widget spécifié
	Affiche_Widget_custom()	Entrée: / Sortie: / Explication: Affiche les widgets personnalisés sur l'interface.
	on_Home_clicked() (slot)	Entrée: / Sortie: / Explication: Redigire l'utilisateur sur la page d'accueil après avoir cliqué sur le bouton "Home".
	on_Help_clicked() (slot)	Entrée: / Sortie: / Explication: Redigire l'utilisateur sur la page d'aide après avoir cliqué sur le bouton "Help".

	on_upload_fichier_clicked() (slot)	Entrée: / Sortie: / Explication: Ouvre une boîte de dialogue lorsque l'utilisateur appuie sur le bouton "upload manually" pour sélectionner des fichiers à importer.
	on_Import_clicked() (slot)	Entrée: / Sortie: / Explication: Redirige l'utilisateur sur la page d'importation après avoir cliqué sur le bouton "Import".
	on_Filtrage_fichiers_clicked() (slot)	Entrée: / Sortie: / Explication: Extrait les fichiers importés, les affichent sur la console et ouvre la fenêtre de style.
	Lancer_filtre_style()	Entrée: / Sortie: / Explication: Lance la fenêtre de dialogue "style_exigence.ui".
	Convertisseur_bit()	Entrée: qint64 bit Sortie: / Explication: Convertir un nombre de bits en une chaîne de caractères représentant sa taille en octets (Ko,Mo ou Go).
	DeleteAll()	Entrée: / Sortie:/ Explications :Permet de supprimer tous les fichiers du tableau et renvoie un message, si l'utilisateur souhaite vraiment supprimer et lorsque tous les fichiers sont supprimés les compteur du fichier dans le tableau est remis à 0 , dans le cas où le tableau ne

		possède aucun fichier alors un message est envoyé à l'utilisateur.
	Bouton_nav()	Entrée: QPushButton *bouton, int page Sortie: / Explication: Change la couleur du bouton rentré en paramètre en bleu, change les autres boutons en blanc et * redirige l'utilisateur vers la page concernée.
Dépendance	Qtreewidget_custom/ QListwidget_custom/ FenetreGraphe / Sss/ Style/ Style_exigence/ Graphe	

3.1. Module d'Importation et structure de fichiers

Ce module est chargé de gérer l'importation des fichiers et leur organisation selon une hiérarchie spécifique (SSS -> SRS -> SDD).

Il s'appuie sur deux classes principales :

- QListWidget_custom : permet de visualiser les fichiers importés sous forme de liste avec la fonctionnalité de glisser et déposer.
- QTreeWidget_custom : organise les fichiers selon leur type (SSS, SRS, SDD) dans une arborescence, également basée sur le glisser-déposer. Ces deux widgets personnalisés facilitent l'interaction de l'utilisateur avec les fichiers, en proposant une interface intuitive et modifiable dynamiquement. L'objectif est de permettre une structuration claire des fichiers en fonction de leur rôle dans le processus de traçabilité.

Classe QListWidget_custom

Nom	QListWidget_custom	
Héritage	/	
Méthodes	QListWidget_custom() (Constructeur)	Entrée: QWidget* parent (optionnel, nullptr par défaut) Sortie: / Explication: Initialise un QListWidget personnalisé acceptant le glisser-déposer.
	startDrag()	Entrée: / Sortie: QString (style CSS) Explication: Permet de démarrer glisser-déposer pour l'élément sélectionné.
Dépendance	/	

Classe QTreeWidget_custom

Nom	QTreeWidget_custom	
Héritage	QTreeWidget	
Méthodes	QTreeWidget_custom() (constructeur)	Entrée: QWidget *parent (optionnel, nullptr par défaut) Sortie: / Explication: Initialise un QTreeWidget personnalisé acceptant le dépôt.
	startDrag()	Entrée: Qt::DropActions actions Sortie: / Explication: Démarre un dépôt de l'élément sélectionné.
	dragMoveEvent()	Entrée: QDragMoveEvent *evenement Sortie: / Explication: Accepte l'action de déplacement si les données sont du texte.

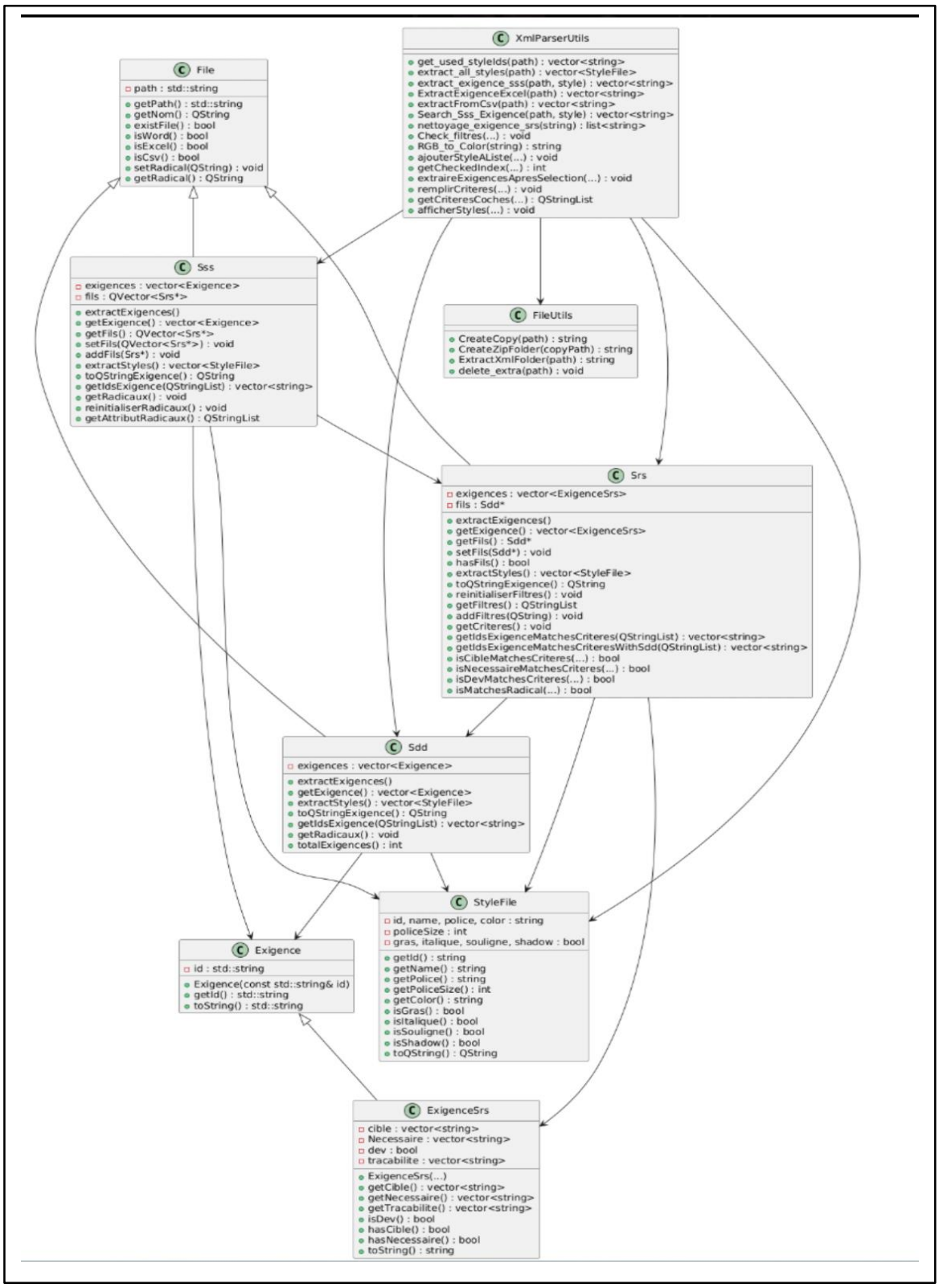
	keyPressEvent()	Entrée: QDragMoveEvent *evenement Sortie: / Explication: Supprime l'élément sélectionné si la touche “Suppr” est pressée
	dropEvent()	Entrée: QDropEvent *evenement Sortie: / Explication: Gère le dépôt d'un fichier dans l’arbre de données et l'organise en SSS, SRS ou SDD selon sa position.
	dragEnterEvent()	Entrée: QDragEnterEvent *evenement Sortie: / Explication: Vérifie si les données sont du texte et autorise l'entrée dans le widget.
Dépendance	/	

Classe QProgressBar_custom

Nom	QProgressBar_custom	
Attributs	· Progress : QProgressDialog* ○ **Signification :** Dialogue de progression pour montrer l’attente · ProgressBar : QProgressBar* ○ **Signification :** Barre de progression animée · nomTache : QLabel* ○ **Signification :** Label affichant le nom de la tâche	
Méthodes	QProgressBar_custom(QWidget*) : void	Constructeur. Configure une barre de progression indéterminée avec style personnalisé
	updateProgress(int, const QString&) : void	Met à jour la valeur de la barre et le nom de la tâche affichée
Dépendance	Aucune	

3.2. Module d'Analyse et d'extraction d'exigences :

Diagramme de classe



Objectif

— Extraire les données pertinentes (styles, structures XML, tableaux, etc.).

Composants Utilisés

— **pugixml** pour l'analyse des fichiers XML internes des DOCX et XLSX.

La conception détaillée

Détailler les classes :

— **Exigence** : La classe **Exigence** servira à définir des exigences de fichier Sss et Sdd sous forme d'objets de type **Exigence**, contenant un **id** représentant le nom de l'exigence. Elle implémentera notamment la fonction **getId()**, qui permettra de récupérer ce nom. Cette fonctionnalité sera essentielle pour comparer deux exigences et vérifier leur égalité, ainsi que pour le calcul du **taux de traçabilité**. De plus, elle servira de **classe de base** pour la classe **ExigenceSrs**, grâce à un mécanisme d'**héritage**.

Nom	Exigence	
Attributs	· id : string ○ **Signification :** le nom de l'exigence	
Méthodes	getId() : string	Retourne la valeur de l'**id** de l'exigence
	toString() : string	Retourne une **chaine de caractères** pour afficher les attributs de l'exigence
Dépendance	Aucune	

— **ExigenceSrs :** La classe **ExigenceSrs** permettra de définir des exigences spécifiques aux fichiers **Srs** sous forme d'objets de type **ExigenceSrs**. Elle héritera de la classe **Exigence**, récupérant ainsi l'**id**, qui représente le nom de l'exigence. En plus de cet identifiant, elle contiendra un **vecteur cible**, listant le matériel (**Hardware**) concerné par l'exigence, ainsi qu'un **vecteur nécessaire**, indiquant les clients pour lesquels l'exigence est nécessaire. Un attribut supplémentaire précisera le **statut de développement** de l'exigence (développé ou non développé).

Elle implémentera plusieurs méthodes essentielles :

— getCible(), qui permettra de récupérer la liste des matériels concernés.
— getNecessaire(), qui renverra les clients pour lesquels l'exigence est requise.
— getDev(), qui indiquera le **statut de développement** de l'exigence.

Ces méthodes joueront un rôle clé dans le **filtrage des exigences**, en renvoyant des critères exploitables pour le tri. De plus, la classe inclura les fonctions hasCible(), qui vérifiera si des

cibles sont définies dans le document, et hasNecessaire(), qui déterminera si un matériel spécifique a été mentionné.

<table>
<tr><td>Nom</td><td colspan="2">ExigenceSrs</td></tr>
<tr><td>Attributs</td><td colspan="2">· cible : vector<string>
○ Signification : La cible des exigences.

· necessaire : vector<string>
○ Signification : Les clients pour lesquelles l’exigence est nécessaire.

· dev : bool
○ Signification : Le statut de l’exigence (développé ou non développé)

· tracabilite : vector<string>
○ Signification : Stocke les exigences tracées par une exigence du fichier Srs.</td></tr>
<tr><td rowspan="5">Méthodes</td><td>getCible() : vector<string></td><td>Retourne la valeur de l’attribut « cible ».</td></tr>
<tr><td>getNecessaire() : vector<string></td><td>Retourne la valeur de l’attribut « necessaire ».</td></tr>
<tr><td>getDev() : bool</td><td>Retourne la valeur de l’attribut « dev ».</td></tr>
<tr><td>hasCible() : bool</td><td>Retourne vrai si le vecteur « cible » n’est pas vide et faux sinon.</td></tr>
<tr><td>hasNecessaire() : bool</td><td>Retourne vrai si le vecteur « necessaire » n’est pas vide et faux sinon.</td></tr>
<tr><td></td><td>toString() : string</td><td>Retourne une chaine de caractères qui permet d’afficher les attributs d’un objet de type ExigenceSrs.</td></tr>
</table>

	getTracabilite() : vector<string>	Retourne le vecteur « tracabilite »
Dépendance	**Héritage :** Exigence	

- **StyleFile :** La classe **StyleFile** permettra de représenter un **style Docx** sous forme d'objets de type **StyleFile**. Chaque instance contiendra plusieurs attributs définissant les caractéristiques du style, notamment :

  - un **id**, correspondant à l'**identificateur** du style dans le fichier styles.xml,
  - une **police**, indiquant la **police** utilisée dans le document Word,
  - une **policeSize**, représentant la **taille** de la police,
  - un attribut **color**, spécifiant la **couleur** du style,
  - un booléen **gras**, indiquant si le texte est en **gras** ou non,
  - un booléen **italique**, précisant si le texte est en **italique** ou non,
  - un booléen **souligne**, déterminant si le texte est **souligné** ou non,
  - un booléen **shadow**, signalant la présence ou l'absence d'une ombre sur le texte.

Elle implémentera plusieurs méthodes :

- **getId(), getPolice(), getPoliceSize()** et **getColor()**, qui permettront de récupérer respectivement l'identificateur, la police, la taille de la police et la couleur du style. De plus, la méthode **getId()** sera nécessaire pour comparer l'**égalité** de deux styles qui sera déterminé si leurs identificateurs respectifs sont égaux.
- **isGras(), isItalique(), isSouligne()** et **isShadow()**, qui indiqueront si le style contient respectivement du **gras**, de l'**italique**, du **souligné** ou une **ombre**.
- **toQString()**, qui générera une **QString** (chaîne de caractères adaptée à un affichage graphique) à partir des différents attributs du style.

Nom	StyleFile
Attributs	· id : string ○ **Signification :** l'identificateur du style utilisé pour les exigences. · police : string ○ **Signification :** la police du style. · policeSize : int ○ **Signification :** la taille du style. · color : string ○ **Signification :** La couleur utilisée pour le style. · gras : bool ○ **Signification : Vrai** si le style inclut le gras, **faux** sinon. · italique : bool ○ **Signification : Vrai** si le style inclut de l'italique, **faux**

<table>
<tr><td></td><td colspan="2">sinon.

· souligne : bool

○ Signification : Vrai si le style inclut du souligné, faux
sinon.

· shadow : bool

○ Signification : Vrai si inclut une ombre, faux sinon</td></tr>
<tr><td rowspan="9">Méthodes</td><td>getId() : string</td><td>Retourne l'identificateur du style</td></tr>
<tr><td>getPolice() : string</td><td>Retourne la police du style</td></tr>
<tr><td>getPoliceSize() : int</td><td>Retourne la taille du style</td></tr>
<tr><td>getColor() : string</td><td>Retourne la couleur du style</td></tr>
<tr><td>isGras() : bool</td><td>Retourne vrai si le style inclut le gras, faux sinon.</td></tr>
<tr><td>isItalique() : bool</td><td>Retourne vrai si le style inclut de l'italique, faux
sinon.</td></tr>
<tr><td>isSouligne() : bool</td><td>Retourne vrai si le style inclut du souligné, faux
sinon.</td></tr>
<tr><td>isShadow() : bool</td><td>Retourne vrai si le style inclut une ombre, faux
sinon.</td></tr>
<tr><td>toQstring() : Qstring</td><td>Retourne un QString (chaine de caractère utilisé
pour l'affichage graphique)</td></tr>
<tr><td>Dépendance</td><td colspan="2">Aucune</td></tr>
</table>

— **File :** La classe **File** servira à définir des fichiers sous formes d'objets de type **File**, contenant un **path** représentant le **chemin d'accès** du fichier. Elle implémentera notamment la fonction **getPath**(), qui permettra de récupérer le chemin d'accès du fichier ainsi que la méthode **supprime**() qui permet de **supprimer** les fichiers associés à l'instance de **File**. Elle servira de classe de base pour les classes dérivées **Sdd**, **Srs** et

Sss, qui hériteront de ses fonctionnalités et de ses attributs.

Nom	File	
Attributs	· path : string ○ **Signification :** le **chemin d'accès** du fichier.	
Méthodes	getPath() : string	Retourne la valeur du **chemin d'accès** du fichier.
	toString() : string	Retourne une **chaîne de caractères** pour afficher les attributs d'un fichier.
	supprimer() : void	Supprimer un fichier **inutiles** créés en plus du pc de l'utilisateur.
	getNom() : Qtring	Retourner juste le nom d'un fichier.
	existFile() : bool	Retourne **vrai** si le fichier existe, **faux** sinon.
	isWord() : bool	Retourne **vrai** si le fichier est un fichier Word, **faux** sinon.
	isExcel() : bool	Retourne **vrai** si le fichier est un fichier Excel, **faux** sinon.
	IsCsv() : bool	Retourne **vrai** si le fichier est un fichier CSV, **faux** sinon.
	GetRadical : QString	Retourne la valeur de l'attribut « radical ».
	SetRadical(QString radical) : void	Affecte la valeur de la variable passée en argument (radical) à l'attribut « radical »
Dépendances	Aucune	

— **Sss :** La classe **Sss** représente un fichier sous forme d'objet de type **Sss**. Elle hérite de la classe **File**, récupérant ainsi l'attribut **path**, qui correspond au **chemin d'accès** du fichier.

En plus de cet attribut, la classe contient :

- Un vecteur **exigences**, qui stocke l'ensemble des **exigences** définies dans un document **SSS**.
- Un attribut **statique style**, représentant le **style unique** appliqué aux **exigences** dans un **SSS**.

Elle implémente les méthodes suivantes :

- **setStyle(sf)** : Attribue à l'attribut style le **style sf**.
- **addExigence(e)** : Ajoute une **exigence** au vecteur **exigences**.
- **extractExigences()** : Extrait les **exigences** d'un document **SSS** et les ajoute au vecteur **exigences** en utilisant addExigence(e).
- **extractStyles()** : Identifie et extrait tous les **styles** utilisés dans un document **SSS**.
- **totalExigences()** : Retourne le nombre total d'exigences présentes dans le fichier SSS.

<table>
<tr><td>Nom</td><td colspan="2">Sss</td></tr>
<tr><td>Attributs</td><td colspan="2">• style : static StyleFile
○ Signification : Le style appliquée aux exigences d'un Sss.
○ Static : l'attribut est « static » car le style appliqué aux exigences est le même pour tous les fichiers Sss au format Docx.
• exigences : vector<Exigence> :
○ Signification : les exigences définies dans un Sss.</td></tr>
<tr><td rowspan="2">Méthodes</td><td>SetStyle(sf :
StyleFile) : void</td><td>• Entrée (sf : StyleFile) : Le style appliquée
aux exigences d'un Sss.
La méthode permet d'attribuer un style spécifique, en l'occurrence « sf » (le style appliqué aux exigences d'un Sss), à l'attribut style.</td></tr>
<tr><td>AddExigence(e : Exigence) : void</td><td>• Entrée (e : Exigence) : une exigence définit
dans un fichier Sss.
La méthode permet d'ajouter une exigence « e » au
vecteur exigences.</td></tr>
</table>

	extractExigences( ) : void	La méthode permet d'extraire les exigences définies dans un fichier Sss et de les ajouter au vecteur **exigences**.
	extractStyles() : void	La méthode permet d'extraire les styles appliqués aux exigences d'un fichier Sss au format **Docx**.
	totalExigences() : int	Retourne le nombre d'exigence dans un fichier Sss (le taille du vecteur **exigences**)
	getExigences() : vector<Exigence>	Retourne l'attribut « exigences »
	GetFils() : QVector<Srs*>	Récupère la liste des fichiers SRS enfants associés à ce SSS.
	SetFils() : QVector<Srs*>	Définit les fichiers SRS enfants associés à ce SSS.
	AddFils() : void	Ajoute un SRS comme enfant du SSS.
	ExtractStyles() : vector<StyleFile>	Extrait tous les styles disponibles depuis le fichier SSS.
	ToQStringExigence() : QString	Retourne tous les identifiants d'exigences sous forme de texte.
	GetIdsExigence(QStringList) : vector<string>	Récupère les identifiants des exigences filtrés par radical.
	GetRadicaux() : void	Extrait les radicaux à partir des identifiants d'exigence, et les enregistre.
	ReinitialiserRadicaux() : void	Vide la liste statique des radicaux enregistrés.
	GetAttributsRadicaux() : QStringList	Récupère la liste des radicaux sélectionnables dans une vue utilisateur.

Dépendances	**Héritage** : File **Association** : Exigence (1 ou plusieurs) **Association** « **static** » : StyleFile

— **Srs :** La classe **Srs** représente un fichier sous forme d'objet de type **Srs**. Elle hérite de la classe **File**, récupérant ainsi l'attribut **path**, qui correspond au **chemin d'accès** du fichier.

En plus de cet attribut, la classe contient :

— Un vecteur **exigences**, qui stocke l'ensemble des **exigences** définies dans un document **SRS**.

— Un attribut **statique style**, représentant le **style unique** appliqué aux **exigences** dans un **SRS**.

Elle implémente les méthodes suivantes :

— **setStyle(sf)** : Attribue à l'attribut style le **style sf**.

— **addExigence(e)** : Ajoute une **exigence** au vecteur **exigences**.

— **extractExigences()** : Extrait les **exigences** d'un document **SRS** et les ajoute au vecteur **exigences** en utilisant addExigence(e).

— **extractStyles()** : Identifie et extrait tous les **styles** utilisés dans un document **SRS**.

— **totalExigences()** : Retourne le nombre total d'exigences présentes dans le fichier SRS.

Nom	SRS	

Attributs	• style : static StyleFile
	  ○ **Signification** : Le style appliquée aux exigences d'un Srs.
	  ○ **Static** : l'attribut est « static » car le style appliqué aux exigences est le même pour tous les fichiers Srs au format Docx.
	• exigences : vector<ExigenceSrs> :
	  ○ **Signification** : les exigences définies dans un Srs.
	• fils : Sdd*
	  ○ **Signification :** le fichiers Sdd fils du Srs**.**	

Méthodes	SetStyle(sf : StyleFile) : void	• **Entrée** (sf : StyleFile) : Le style appliquée aux exigences d'un Srs. La méthode permet d'attribuer un style spécifique, en l'occurrence « sf » (le style

		appliqué aux exigences d'un Srs), à l'attribut **style**.
	AddExigence(e : ExigencSrs) : void	• **Entrée** (e : Exigence) : une exigence définit dans un fichier Srs. La méthode permet d'ajouter une exigence « e » au vecteur **exigences**.
	extractExigences( ) : void	La méthode permet d'extraire les exigences définies dans un fichier Srs et de les ajouter au vecteur **exigences**.
	extractStyles() : void	La méthode permet d'extraire les styles appliqués aux exigences d'un fichier Srs au format **Docx**.
	totalExigences() : int	Retourne le nombre d'exigence dans un fichier Srs (le taille du vecteur **exigences**)
	GetExigence() : vector<ExigenceSrs>	Récupère la liste des exigences associées à ce fichier Srs.
	GetFils() : Sdd*	Récupère le fichier fils associé à cet objet Srs.
	SetFils(Sdd*) : void	Définit le fichier fils (Sdd) associé à cet objet Srs.
	HasFils() : bool	Vérifie si un fichier fils est associé à cet objet Srs.
	ToQStringExigence() : QString	Convertit les identifiants des exigences en une chaîne QString.

	ReinitialiserFiltres() : void	Réinitialise la liste des filtres de la classe Srs.
	GetFiltres() : QStringList	Récupère la liste des filtres associés à la classe Srs.
	AddFiltres() : void	Ajoute un filtre à la liste des filtres.
	GetCritères() : void	Extrait et ajoute les critères de filtres à la liste des filtres.
	GetIdsExigenceMatchesCriteres(QStringList) : vector<stirng>	Récupère les identifiants des exigences qui correspondent aux critères sélectionnés.
	GetIdsMatchesCriteresWithSdd(QStringList)	Récupère les identifiants des exigences SRS qui correspondent à au moins un des critères sélectionnés.
	IsCibleMatchesCriteres() : bool	Vérifie si la cible de l'exigence correspond à l'un des critères sélectionnés.
	IsNecessaireMatcheesCriteres() : bool	Vérifie si la nécessité de l'exigence correspond à l'un des critères sélectionnés.
	IsDevMatchesCriteres() : bool	Vérifie si l'état de développement de l'exigence correspond à l'un des critères sélectionnés.
	IsMatchesRadical() : bool	Vérifie si l'ID de l'exigence correspond à l'un des radicaux sélectionnés.
Dépendances	**Héritage** : File **Association** : Exigence (1 ou plusieurs) **Association** « **static** » : StyleFile	

— **Sdd :** La classe **Sdd** représente un fichier sous forme d'objet de type **Sdd**. Elle hérite de la classe **File**, récupérant ainsi l'attribut **path**, qui correspond au **chemin d'accès** du fichier.

En plus de cet attribut, la classe contient :

— Un vecteur **exigences**, qui stocke l'ensemble des **exigences** définies dans un document **SDD**.

— Un attribut **statique style**, représentant le **style unique** appliqué aux **exigences** dans un **SDD**.

Elle implémente les méthodes suivantes :

— **setStyle(sf)** : Attribue à l'attribut style le **style sf**.

— **addExigence(e)** : Ajoute une **exigence** au vecteur **exigences**.

— **extractExigences()** : Extrait les **exigences** d'un document **SDD** et les ajoute au vecteur **exigences** en utilisant addExigence(e).

— **extractStyles()** : Identifie et extrait tous les **styles** utilisés dans un document **SDD**.

— **totalExigences()** : Retourne le nombre total d'exigences présentes dans le fichier SDD.

<table>
<tr><td>Nom</td><td colspan="2">Sdd</td></tr>
<tr><td>Attributs</td><td colspan="2">• style : static StyleFile
○ Signification : Le style appliquée aux exigences d'un Sdd.
○ Static : l'attribut est « static » car le style appliqué aux exigences est le même pour tous les fichiers Sdd au format Docx.
• exigences : vector<Exigence> :
○ Signification : les exigences définies dans un Sdd.</td></tr>
<tr><td rowspan="3">Méthodes</td><td>SetStyle(sf : StyleFile) : void</td><td>• Entrée (sf : StyleFile) : Le style appliquée aux exigences d'un Sdd.
La méthode permet d'attribuer un style spécifique, en l'occurrence « sf » (le style appliqué aux exigences d'un Sdd), à l'attribut style.</td></tr>
<tr><td>AddExigence(e : Exigence) : void</td><td>• Entrée (e : Exigence) : une exigence définit dans un fichier Sdd.
La méthode permet d'ajouter une exigence « e » au vecteur exigences.</td></tr>
<tr><td>extractExigences() : void</td><td>La méthode permet d'extraire les exigences définies dans un fichier Sdd et de les ajouter au vecteur exigences.</td></tr>
</table>

	extractStyles() : void	La méthode permet d'extraire les styles appliqués aux exigences d'un fichier Sdd au format **Docx**.
	totalExigences() : int	Retourne le nombre d'exigence dans un fichier Sdd (le taille du vecteur **exigences**)
	GetRadicaux() : void	Extrait les radicaux des exigences dans le fichier SDD
	ToQStringExigence() : QString	Retourne tous les identifiants d'exigences du fichier SDD sous forme de texte lisible
	GetIdsExigence(QStringList) : vector<string>	Récupère les identifiants de toutes les exigences.
Dépendances	**Héritage** : File **Association** : Exigence (1 ou plusieurs) **Association** « **static** » : StyleFile	

Bibliothèques utilisés :

- **Pugixml : pugixml** est une bibliothèque C++ légère et rapide dédiée à la manipulation de fichiers XML. Elle permet de **parser**, **modifier**, **générer** et **interroger** des fichiers XML avec une API simple et efficace. Elle a été utilisé dans les méthodes **« extractExigences »** et **« extractStyles() »**.

3.3. Module de visualisation et de sélection des styles

Ce module joue un rôle clé dans la chaîne de traçabilité. Il permet de visualiser et de sélectionner les styles présents dans les documents importés, en vue de l'extraction ultérieure des exigences. Comme pour les autres modules, cette section décrit les classes impliquées, leur rôle et les traitements effectués.

L'identification des exigences reposant principalement sur les caractéristiques de style (police, taille, couleur, gras, etc.), ce module offre une interface permettant d'examiner ces styles de manière détaillée. L'utilisateur peut ainsi choisir les styles à considérer comme indicateurs d'exigences.

Cette étape garantit une détection précise et adaptée au contenu des documents analysés. Plusieurs classes sont mobilisées pour assurer ces fonctionnalités.

- **Style_exigence** : interface graphique permettant à l'utilisateur de sélectionner les styles à prendre en compte comme exigences.
- **StyleFile** : classe représentant un style extrait d'un fichier (police, couleur, etc.).

Classe Style_exigence

Nom	**Style_exigence**	
Héritage	QDialog	
Attributs	ui: **Ui::style_exigence**	Une interface utilisateur générée par Qt Designer.
	fichiers_extraits: **QVector<Sss*>**	Une liste des fichiers extraits de l'arbre de données stockés dans un vecteur de classe Sss.
Méthodes	style_exigence() (constructeur)	Entrée: MainWindow* mainWindow, QWidget* parent, QVector<Sss*> fichiers Sortie: /
	~style_exigence() (destructeur)	Entrée: / Sortie: /
	on_pushButton_2_clicked()	Entrée: / Sortie: void Explication: Affiche la page de style des exigences SRS (bouton 2)
	on_pushButton_3_clicked()	Entrée: / Sortie: void Explication: Affiche la page de style des exigences SRS (bouton 3)
	on_pushButton_4_clicked()	Entrée: / Sortie: void Explication: Affiche la page de style des exigences SSS

	on_pushButton_6_clicked()	Entrée: / Sortie: void Explication: Affiche la page de style des exigences SDD
Dépendance	/	

Classe fenetreExigence

Nom	FentreExigence	
Attributs	mainwindow **MainWindow***	Pointeur vers la fenêtre principale
	listeExigences **QListWidget***	Liste affichant les exigences liées à un nœud sélectionné
Méthodes	FenetreExigences(MainWindow *, QWidget*) (constructeur)	Entrée: MainWindow* mainWindow, QWidget* parent, QVector<Sss*> fichiers Explication : Initialise la mini-fenêtre des exigences dans MainWindow
	setExigences(const QStringList&) (destructeur)	Entrée: / Sortie: / Explication : Met à jour la liste affichée des exigences
Dépendance	**Héritage :** QDialog	

Objectif

- Identifier les styles utilisés pour extraire les exigences

3.4. Module de Visualisation

Ce module est dédié à l'affichage graphique des relations de traçabilité entre les exigences des différents niveaux (SSS, SRS, SDD). Il joue un rôle clé dans la compréhension visuelle de l'arborescence des documents et dans la validation de la couverture des exigences.

La visualisation s'appuie principalement sur la classe FenetreGraphe, qui utilise QGraphicsView pour générer dynamiquement un graphe interactif. Ce graphe représente les liens hiérarchiques et logiques entre les exigences extraites.

Ce module intègre également les classes de style (Style) afin d'assurer une présentation cohérente et lisible de l'interface graphique. Ces styles permettent d'unifier l'apparence des widgets personnalisés tels que QListWidget_custom et QTreeWidget_custom.

Enfin, certaines statistiques peuvent également être affichées dans cette interface, à travers une classe dédiée (comme StatistiqueFichier), afin de compléter la visualisation par des

données numériques pertinentes (nombre d'exigences, taux de couverture, etc…).

Classe FenetreGraphe

Nom	**FenetreGraphe**	
Héritage	QGraphicsView	
Attributs	LimiteZoom : ***int***	Limite actuelle du zoom. Incrémente la variable si l'utilisateur glisse la molette vers le haut ou appuie sur le bouton "+" et décrémente dans le cas contraire.
	MAX_ZOOM : ***const int***	Limite maximale du zoom. Valeur maximale que la variable "limiteZoom" peut atteindre.
	MIN_ZOOM : ***const int***	Limite minimale du zoom. Valeur minimale que la variable "limiteZoom" peut atteindre.
Méthodes	zoomAvant()	Entrée: / Sortie: void Explication: Applique un zoom avant si limiteZoom est inférieur à MAX_ZOOM.
	zoomArriere()	Entrée: / Sortie: void Explication: Applique un zoom arrière si limiteZoom est supérieur à MIN_ZOOM.
	FenetreGraphe() (constucteur)	Entrée: QWidget* parent : widget parent, initialisé à nullptr par défaut
	wheelEvent()	Entrée: QWheelEvent *event Sortie: void Explication: Capte l'événement de la molette et le redirige vers la méthode Zoom_dezoom()
	Zoom_dezoom()	Entrée: QWheelEvent *event Sortie: void Explication: Applique un zoom avant ou arrière en fonction du

		mouvement de la molette.
Dépendance	/	

Classe Style

Nom	**Style**	
Héritage	/	
Méthodes *(toutes static)*	Style_Button_analyse()	Entrée: / Sortie: QString (style CSS) Explication: Retourne le style par défaut du bouton "analyse".
	Changer_bleu_bouton_nav()	Entrée: / Sortie: QString (style CSS) Explication: Retourne le style après avoir cliqué sur le bouton "analyse".
	Style_liste_Fichiers()	Entrée: / Sortie: QString (style CSS) Explication: Retourne le style du tableau “QListWidget_custom” et ses éléments.
	Style_arbre_Fichiers()	Entrée: / Sortie: QString (style CSS) Explication: Retourne le style de l’arbre “QTreeWidget_custom” et ses éléments.

	Style_progressBar()	Entrée: / Sortie : QString Explication: Retourne le style CSS pour la barre de progression (QProgressBar)
	Style_CheckBox_Critere_Style()	Entrée: / Sortie : QString Explication: Retourne le style CSS personnalisé pour les QCheckBox de critères
Dépendance	/	

Classe graphe

Nom	Graphe	
Attributs	• fichier : QVector<Sss*> ○ **Signification** : Contient les fichiers SSS liés au graphe. • nodes : QVector<NodeItem*> ○ **Signification** : Liste des nœuds du graphe. • links : QVector<LinkItem*> ○ **Signification :** Liste des liens entre les nœuds du graphe.	
Méthodes	existNode(NodeItem* node) : bool	vérifie si un nœud (NodeItem* node) existe dans le graphe. Elle retourne **vrai** si le nœud existe, **faux** sinon.
	ajouterNode() : void	Ajoute des nœuds au graphe à partir de la hiérarchie de fichiers. Cette méthode parcourt une liste de fichiers racines de type "Sss" et crée pour chacun un nœud "NodeItem" associé. Ensuite, elle parcourt leurs enfants de type "Srs", crée les nœuds correspondants, et établit les relations parent-enfant dans le graphe. Si un "Srs" possède un fichier enfant ("Sdd"), un nœud est aussi créé pour celui-ci et ajouté au graphe avec les connexions adéquates.
	organiserLignes() : void	Organise les nœuds du graphe en lignes selon leur type. Cette méthode dispose graphiquement les nœuds (`NodeItem`) selon trois niveaux : - Ligne supérieure : les fichiers de type `SSS` - Ligne centrale : les fichiers de type `SRS`

		- Ligne inférieure : les fichiers de type `SDD` (si présents)
	ajouterLien(NodeItem* parent, NodeItem* fils, float tauxTracabilite) : LinkItem*	Permet de créer et renvoie un lien entre 2 nœuds avec un le taux de tracabilité entre les 2 nœuds.
	connectNodeAndLink(NodeItem* node, LinkItem* link) : void	Connecter un noeud à un lien pour synchroniser les coordonnées des noeuds avec celles du lien pour dynamiser le graphe
	afficherGraphe(QGraphicsScene* scene, const QStringList& criteres) : void	Permet l'affichage du graphe (nœuds et liens) dans la scène (QGraphicsScene* scene).
	getFichiers() : QVector<Sss*>	Récupère la liste des fichiers associés au graphe.
Dépendances	Aucune	

Classe NodeItem

Nom	NodeItem	

Attributs	• file : File*
	  ○ **Signification** : Fichier associé au nœud.
	• label : QString
	  ○ **Signification** : Étiquette du nœud.
	• type : TypeFile
	  ○ **Signification :** Type de fichier représenté par le nœud.
	• parentNodes : QList<NodeItem*>
	  ○ **Signification** : Liste des nœuds parents
	• filsNodes : QList<NodeItem*>
	  ○ **Signification :** Liste des nœuds fils.
	• isRed : bool
	  ○ **Signification :** Indique si le nœud est rouge (état).
	• isBlinking : bool
	  ○ **Signification :** Indique si le nœud clignote.
	• blinkTimer : QTimer*
	  ○ **Signification :** Minuterie pour gérer le clignotement.	

Méthodes	addParent(NodeItem* parent) : void	Ajoute un nœud parent.

	getFils() const : QList<NodeItem* >	Retourne la liste des nœuds fils.
	addFils(NodeItem * fils) : void	Ajoute un nœud fils.
	hasFils(): bool	Vérifie si le nœud a des fils. **Vrai** si le nœud a des fils, **faux** sinon.
	getParents(): QList<NodeItem* >	Retourne la liste des nœuds parents associés à ce nœud.
	getType() : TypeFile	Retourne le type de fichier associé au nœud.
	boundingRect() : QRectF	Retourne la zone occupée par le nœud dans la scène. Cette méthode définit une zone fixe autour du centre de l'élément.
	makeBlinking() :	Active le clignotement du nœud.
	paint(QPainter*, const QStyleOptionGraphicsItem*, QWidget*) :	Dessine le nœud graphique sur la scène. Cette méthode est responsable du dessin du nœud sur la scène. Elle définit d'abord la couleur du nœud en fonction de son état de clignotement et de son type de fichier. Ensuite, elle dessine un cercle représentant le nœud à la position et la taille spécifiées par "boundingRect()".
Dépendances	**Héritage :** QGraphicsObject	

Classe LinkItem

Nom	LinkItem

Attributs	• sourceItem : QGraphicsItem*
	  ○ **Signification** : Élément source du lien.
	• targetItem : QGraphicsItem*
	  ○ **Signification** : Élément cible du lien.
	• tauxTracabilite : float
	  ○ **Signification :** Taux de traçabilité du lien.
	• line : QLineF
	  ○ **Signification** : Ligne représentant visuellement le lien.

Méthodes	boundingRect() const : QRectF	Retourne le rectangle englobant de l'élément graphique.
	paint(QPainter* painter, const QStyleOptionGraphicsItem* option, QWidget* widget) : void	Dessine le lien entre les deux items avec le taux de traçabilité.
	getTauxTracabilite() const : int	Renvoie le taux de traçabilité associé à ce lien.
Dépendances	**Héritage :** QGraphicsObject	

Classe FenetreStatistiques

Nom	FenetreStatistiques	
Attributs	/	
Méthodes	afficherHistogramme() : void	Affiche un histogramme animé représentant les taux de traçabilité dans un widget Qt
	initialiserTranches (): void	Répartit les taux dans 11 tranches de 10% (0–9%, 10–19%, ..., 100%)
	dessinerAxes	Dessine les axes X et Y de l'histogramme dans la scène graphique
	ajouterTitresAxes	Ajoute les titres des axes (X : % de traçabilité, Y : nombre de liens) à la scène
	ajouterBarres	Ajoute les barres représentant les tranches et anime leur affichage
	afficherVue	Insère la scène graphique dans un QGraphicsView et l'intègre dans le layout du widget donné
Dépendances	**Héritage :** QGraphicsObject	

Objectif

— Générer un graphe interactif des liens de traçabilité.

Interfaces et Visibilité

- GenererGraphe(Liste<Exigence> liens) : Graphe
- AfficherGraphe(Graphe graphe) : void

Algorithmes Utilisés

- Utilisation de QGraphicsView pour l'affichage des graphes.

3.5. Module de calcul de taux de traçabilité

Ce module a pour objectif de mesurer la traçabilité des exigences entre les différents niveaux de documentation (SSS, SRS, SDD) et de générer un rapport synthétique. Il permet d'évaluer si chaque exigence définie en amont est bien prise en compte à tous les niveaux du projet. Le calcul du taux de traçabilité repose sur la comparaison entre les exigences du document parent (par exemple SSS) et celles du document enfant (SRS), en vérifiant les correspondances. Les exigences non tracées sont automatiquement détectées et listées.

Les résultats sont affichés sous forme de taux (%) et peuvent être exportés dans un fichier CSV, afin de faciliter l'analyse.

Objectif

- Filtrer les exigences selon certains critères.
- Calculer le taux de traçabilité.

Classe TRACABILITE

Méthodes statiques

Nom de méthode	taux_traca_sss_srs
Type de retour	int
Paramètre	vector<Exigence> exigences_pere, vector<ExigenceSrs> exigences_fils
Description	Calcul du taux de traçabilité entre un fichier SSS et un fichier SRS

Nom de méthode	taux_traca_sss_sdd
Type de retour	int
Paramètre	vector<ExigenceSrs> exigences_pere, vector<Exigence> exigences_fils
Description	Calcul du taux de traçabilité entre un fichier SRS et un fichier SDD

Nom de méthode	taux_traca
Type de retour	float
Paramètre	const vector<std::string> pere, const std::vector<std::string>& fils
Description	Calcule le taux de traçabilité entre deux ensembles d'identifiants d'exigences

3.6. Module de génération de rapport

Ce module permet de générer un fichier .csv qui résume le graphe. Une ligne du rapport correspond à la traçabilité d'une exigence d'un document SSS.

Lorsqu'une exigence est tracée (repérée) dans un document SSS, SRS ou SDD, le nom du fichier et le nom de l'exigence seront rempli sur la ligne.

Objectif

- Repérer facilement les exigences non tracées
- Avoir un document qui permet de résumer et d'enregistrer la traçabilité à un moment précis du projet.

Classe Rapportligne

Attributs

Attributs	nameSSS
Type	QString
Valeur par défaut	Null
Description	nom du fichier SSS tracé

Attributs	nameSRS
Type	QString
Valeur par défaut	Null
Description	nom du fichier SRS tracé

Attributs	nameSDD
Type	QString

Valeur par défaut	Null
Description	nom du fichier SDD tracé

Attributs	exigenceSSS
Type	string
Valeur par défaut	Null
Description	nom de l'exigence du SSS tracé

Attributs	exigenceSRS
Type	string
Valeur par défaut	Null
Description	nom de l'exigence du fichier SRS tracé

Attributs	exigenceSDD
Type	string
Valeur par défaut	Null
Description	nom de l'exigence du fichier SDD tracé

Constructeurs

Constructeur	Rapportligne
Paramètre	QString nameSSS,QString nameSRS,QString nameSDD,string exigenceSSS,string exigenceSRS,string exigenceSDD
Description	Construit un objet rapportligne rempli par les paramètres donnés

Constructeur	Rapportligne
Paramètre	-
Description	Construit un objet rapportligne vide

Méthodes

Nom de méthode	setNameSSS
Type de retour	void
Paramètre	QString nameSSS
Description	setter du nom du fichier SSS

Nom de méthode	setNameSRS
Type de retour	void
Paramètre	QString nameSRS
Description	setter du nom du fichier SRS

Nom de méthode	setNameSDD
Type de retour	void
Paramètre	QString nameSDD
Description	setter du nom du fichier SDD

Nom de méthode	setExigenceSSS
Type de retour	void
Paramètre	QString exigenceSSS
Description	setter du nom de l'exigence du fichier SSS

Nom de méthode	setExigenceSRS
Type de retour	void
Paramètre	QString exigenceSRS
Description	setter du nom de l'exigence du fichier SRS

Nom de méthode	setExigenceSDD
Type de retour	void

Paramètre	QString exigenceSDD
Description	setter du nom de l'exigence du fichier SDD

Nom de méthode	isInSRS
Type de retour	bool
Paramètre	Exigence* exi, Srs docSrs, ExigenceSrs& rtr
Description	vérifie si l'exigence exi est dans le document docSrd

Nom de méthode	isInSDD
Type de retour	bool
Paramètre	Exigence* exi, Sdd docSdd
Description	vérifie si l'exigence exi est dans le document docSdd

Classe Rapport

Attributs

Attributs	listeDocSss
Type	QVector<Sss*>
Valeur par défaut	Null
Description	Liste qui contient les documents SSS à traite

Attributs	lignesRapport
Type	list<Rapportligne>
Valeur par défaut	Null
Description	Liste qui contient les informations sur chaque ligne du rapport

Constructeur

Constructeur	Rapport

Paramètre	QVector<Sss*> listeDocSss
Description	Construit un objet rapport, avec sa liste des documents SSS

Méthodes

Nom de méthode	getLignesRapport
Type de retour	list<Rapportligne>
Paramètre	-
Description	Récupère la liste des lignes du rapport

Nom de méthode	generation
Type de retour	void
Paramètre	QVector<Sss*> listeDocSss
Description	rempli lignesRapport en traitant les exigences de chaque SSS

4. Classes utilitaires utilisées

XmlParserUtils

Nom	XmlParserUtils	
Attributs	Aucun	
Méthodes	Check_filtres(pugi::xml_node & paragraphe, Exigence_Srs& exigences) : void	Cette fonction analyse un paragraphe XML pour extraire et filtrer des informations relatives aux exigences (les critères de filtrage). Elle extrait des informations en fonction des mots-clés tels que "Cible", "Nécessaire à", et "Développé", et met à jour exigences.
	extract_all_styles(const std::string& path) : vector<StyleFile>	Extrait tous les styles utilisés dans un document Word à partir du fichier XML des styles. Cette fonction analyse le fichier XML "styles.xml" associé au document Word pour en extraire les informations relatives aux styles utilisés dans le document. Ces informations sont stockées dans un vecteur de structures "StyleFile" et

		renvoyées à la fin de la fonction.
	nettoyage_exigence_srs(const std::string& line_exigence_srs) : list<string>	Nettoie une ligne d'exigences SRS et extrait chaque exigence sous forme de liste. Cette fonction prend une ligne de texte représentant une ou plusieurs exigences SRS, supprime le préfixe avant le premier caractère ":", puis extrait chaque exigence séparée par des virgules. Elle supprime également les espaces en début et fin de chaque exigence avant de l'ajouter à la liste retournée.
	Search_Sss_Exigence(const std::string& path, const StyleFile& userStyle) : : vector<string>	Recherche et extrait les exigences dans un document SSS Word en fonction d'un style. Cette fonction ouvre un fichier au format Word, extrait son contenu XML, puis parcourt chaque paragraphe pour détecter ceux ayant le style passé en argument. Si le style correspond, le texte du paragraphe est considéré comme une exigence et est ajouté à la liste.
	ExtractExigenceExcel(const std::string& PathExcelDoc) : vector<string>	Extrait les exigences d'un document Excel. Elle utilise la bibliothèque pugixml pour analyser le contenu XML.
	afficherStyles(QDialog* styleExigence, const QVector<Sss*>& sss_files, MainWindow* mainwindow) : void	Affiche les styles détectés dans les documents SSS, SRS et SDD dans une boite de dialogue.
	get_used_styleIds(const std::string& path_doc_xml) : vector<string>	Récupère les identifiants de style utilisés dans un document XML Word. Cette fonction charge un document XML Word spécifié par le chemin "path_doc_xml" et parcourt tous les paragraphes du corps du document. Pour chaque paragraphe, elle recherche la balise "<w:pStyle>" et en extrait l'attribut "w:val", qui représente l'identifiant du style appliqué au paragraphe. Les identifiants de style extraits sont ajoutés à un vecteur, qui est retourné à la fin de la fonction.
	extract_exigence_sss(const std::string& path, const StyleFile& userStyle) : vector<string>	Extrait les exigences d'un fichier SSS, quel que soit son format. Cette fonction détecte automatiquement le type de fichier en se basant sur son extension, puis appelle la fonction d'extraction appropriée.
	ajouterStyleAListe(QListWidget* listWidget, const StyleFile& style) : void	Ajoute un style à une QListWidget sous forme d'un item contenant une case à cocher et un label.

	getCheckedIndex(QListWidget * listWidget) : int	Retourne l'index du premier élément coché dans un QListWidget contenant des QCheckBox.
	extractFromCsv(const std::string& path) : vector<string>	Extrait les exigences à partir d'un fichier CSV. Cette fonction lit un fichier CSV ligne par ligne, supprime les guillemets (""") de chaque ligne, puis découpe la ligne en éléments en utilisant ";" comme séparateur. Chaque élément est ajouté à un vecteur de chaînes représentant les exigences.
	remplirCriteres(QFrame* frame, const QStringList& liste_criteres, QObject* receiver, std::function<void()> updateCallback) : void	Remplit dynamiquement un QFrame avec une liste de critères sous forme de cases à cocher (QCheckBox). Chaque élément de la QStringList passée est ajouté sous forme de QCheckBox dans la liste. Une connexion est faite entre chaque QCheckBox et un callback fourni pour réagir aux changements d'état.
	getCriteresCoches(QFrame* frame) : QStringList	Récupère la liste des critères sélectionnés dans les QListWidget. Cette fonction parcourt récursivement tous les QListWidget présents dans le QFrame donné en paramètre, et extrait les textes des QCheckBox qui sont cochées. Elle retourne la liste de ces textes comme une QStringList.
	RGB_to_Color(const std::string &rgb) : string	Convertit une couleur en format RGB en son nom de couleur correspondant. Cette fonction prend une chaîne de caractères représentant une couleur en format RGB, la convertit en un objet "QColor", puis cherche le nom de cette couleur dans la liste des couleurs. Si la couleur existe dans cette liste, le nom est retourné sous forme de chaîne. Si elle n'est pas trouvée, le code hexadécimal de la couleur est retourné.
	extraireExigencesApresSelection(const QVector<Sss*>& sss_files, QListWidget* listWidgetSSS, QListWidget* listWidgetSRS, QListWidget* listWidgetSDD, const std::vector<StyleFile>& styles_sss, const std::vector<StyleFile>& styles_srs, const std::vector<StyleFile>& styles_sdd, MainWindow *mainwindow) : void	Extrait les exigences à partir des fichiers SSS/SRS/SDD en fonction des critères sélectionnés. Cette fonction est déclenchée après la sélection de critères de filtrage par l'utilisateur. Elle applique les filtres sélectionnés aux classes Sss, Srs et Sdd, réinitialise les filtres, puis extrait les exigences de manière asynchrone à l'aide de QtConcurrent, tout en affichant une boîte de progression. À la fin de l'extraction, elle déclenche le traitement du graphe dans la fenêtre principale ("MainWindow::Bouton_graphe").
Dépendance	Aucune	

FileUtils

Nom	XmlParserUtils	
Attributs	Aucun	
Méthodes	CreateCopy(const std::string& FilePath) : string	Crée le path d'une copie d'un fichier et le renvoie.
	CreateZipFolder(const std::string& FilePathCopy) : string	Crée le path du dossier zip à partir de ''FilePathCopy'' et le renvoie.
	ExtractXmlFolder(const std::string& FilePath) : string	Renvoie le dossier contenant les fichiers xml d'un fichier dont le path est FilePath.
	delete_extra(std::string& path) : void	supprime les fichiers, dossiers supplémentaires créés
Dépendance	Aucune	

5. Conclusion

Ce document détaille l'architecture et les composants clés du projet. Il servira de référence pour l'implémentation et la validation du logiciel.

En décrivant précisément chaque module, ses responsabilités, les classes associées, les méthodes, ainsi que les algorithmes utilisés, cette conception détaillée permet de garantir la cohérence globale du développement. Elle facilite également la collaboration entre les membres de l'équipe en offrant une vision commune et structurée du système à réaliser.

De plus, cette conception détaillée assure la traçabilité entre les exigences fonctionnelles exprimées dans le cahier des charges et les composants techniques du logiciel, ce qui est fondamental pour une validation rigoureuse. Enfin, ce document joue un rôle clé dans la maintenance et l'évolution future du projet, en offrant une documentation technique claire, réutilisable et facilement consultable.

6. Références

— **Documentation technique :**

- pugixml "Light-weight C++ XML processing library" : https://pugixml.org/docs/manual.html
- Qt Documentation "Qt 6.5 C++ GUI framework" : https://doc.qt.io/ qt-6/

— **Tutoriel et formations :**

- **Tutoriels sur Qt/Python** (chaînes diverses sur YouTube).
- **LinkedIn learning** : C++ avancé et développement d'applications avec Qt.
- **OpenClassroom** : Programmation orienté objet C++.

— **Outils d'assistance :**

- **ChatGPT** : Aide pour le choix des bibliothèques selon le besoin.