

# Remerciements

18 mai 2009

Nous remercions :

- Monsieur Michel Meynard pour sa disponibilité et ses conseils.
- Monsieur Maneschi, qui a imprimé notre dossier.
- Madame Marie-Josée Bernard, secrétaire des Licences Informatiques, pour avoir relié notre rapport.
- QTSoftWare pour avoir créé une bibliothèque libre d'utilisation.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Généralités . . . . .	5
1.2	Le sujet . . . . .	5
1.3	Cahier des charges . . . . .	5
1.4	Syntaxe . . . . .	6
1.5	Analyse de quelques éditeurs web existants . . . . .	6
<b>2</b>	<b>Organisation du projet</b>	<b>8</b>
2.1	Organisation du travail . . . . .	8
2.2	Choix des outils de développement . . . . .	9
<b>3</b>	<b>Analyse du projet</b>	<b>10</b>
3.1	Analyse générale . . . . .	10
3.2	Première approche . . . . .	11
3.3	Editeur . . . . .	12
3.4	Hierarchie . . . . .	13
3.5	Site-Map . . . . .	14
3.6	WebBrowser . . . . .	15
3.7	FTPBrowser . . . . .	16
<b>4</b>	<b>Analyse technique</b>	<b>17</b>
4.1	Analyse du MainWindow . . . . .	17
4.1.1	Partie graphique . . . . .	17
4.1.2	Préférences . . . . .	18
4.2	Analyse de l'Editeur . . . . .	19
4.2.1	Analyse de Coloration . . . . .	19
4.2.2	Analyse de WidgetLigne . . . . .	20
4.3	Analyse du WebBrowser . . . . .	20
4.3.1	Introduction . . . . .	20
4.3.2	Analyse de l'historique . . . . .	21
4.3.3	Analyse des marques-pages . . . . .	21
4.4	Analyse du FTPBrowser . . . . .	21
<b>5</b>	<b>Développement du MainWindow</b>	<b>22</b>
5.1	Introduction . . . . .	22
5.2	Fonctions . . . . .	22
5.2.1	MainWindow : :gestionMenuBarre . . . . .	22
5.2.2	MainWindow : :gestionMenuActions . . . . .	22
5.2.3	MainWindow : :fichierNouveau . . . . .	23
5.2.4	MainWindow : :enregistrer . . . . .	24
<b>6</b>	<b>Développement de la barre de recherche</b>	<b>25</b>
6.1	Introduction . . . . .	25
6.2	Fonctions . . . . .	25
6.2.1	FindDialog : :remplacer . . . . .	25
<b>7</b>	<b>Développement de Editeur</b>	<b>26</b>
7.1	Introduction . . . . .	26
7.2	Fonctions de l' Editeur . . . . .	26
7.2.1	Editeur : :recupereUrl() . . . . .	26
7.2.2	WidgetLigne : :paintEvent() . . . . .	27
7.2.3	WidgetLigne : :chargement() . . . . .	28
7.2.4	Erreur : :ecritErreur() . . . . .	28

<b>8 Développement de Coloration</b>	<b>29</b>
8.1 Introduction . . . . .	29
8.2 Fonctions de Coloration . . . . .	29
8.2.1 Coloration : :ajouteMotCleFichier . . . . .	29
8.2.2 Coloration : :highlightBlock(const QString & text) . . . . .	31
8.2.3 Boucle de reconnaissance des mots clé en fonction du langage . . . . .	33
<b>9 Développement du Completer</b>	<b>35</b>
9.1 Introduction . . . . .	35
9.2 Fonctions . . . . .	35
9.2.1 Completer : :activeComplete . . . . .	35
9.2.2 Completer : :Completer . . . . .	36
9.2.3 Completer : :insertCompletion . . . . .	37
9.2.4 Completer : :modelFromFile . . . . .	37
<b>10 Développement de Syntaxe</b>	<b>38</b>
10.1 Introduction . . . . .	38
10.2 Fonctions principales de Syntaxe . . . . .	39
10.2.1 Sytaxe : :verification . . . . .	39
10.2.2 Sytaxe : :surligneBloc . . . . .	40
<b>11 Développement Indentation</b>	<b>42</b>
11.1 Introduction . . . . .	42
11.2 Fonctions . . . . .	42
11.2.1 void Indent : :texteChange(int pos, int ajout, int enleve) . . . . .	42
11.2.2 void Indent : :indenter(QString s) . . . . .	42
<b>12 Développement de Projets</b>	<b>44</b>
12.1 Introduction . . . . .	44
12.2 Fonctions . . . . .	45
12.2.1 Hierarchie : :parcours . . . . .	45
12.2.2 Projet : :parcours . . . . .	45
12.2.3 Hierarchie : :itemDoubleClic . . . . .	46
<b>13 Développement du plan du site</b>	<b>47</b>
13.1 Introduction . . . . .	47
13.2 Fonctions . . . . .	47
13.2.1 Fichier : :ajouter(string corps) . . . . .	47
13.2.2 Fichier : :prefixe(string extension) . . . . .	47
13.2.3 Arbo : :listerRepertoire(char* nom) . . . . .	47
<b>14 Développement du WebBrowser</b>	<b>49</b>
14.1 Introduction . . . . .	49
14.2 Choix d'implémentation . . . . .	50
14.3 Fonctions Principales . . . . .	50
14.3.1 WebBrowser : :chargerPage . . . . .	50
14.3.2 WebBrowser : :actionCharger . . . . .	50
14.3.3 Historique : :Restaurer . . . . .	51
14.3.4 Historique : :Enregistrer . . . . .	52
14.3.5 MarquePage : :StartDrag . . . . .	52

<b>15 Développement du FTPBrowser</b>	<b>54</b>
15.1 Introduction . . . . .	54
15.2 Fonctions Principales . . . . .	54
15.2.1 WidgetDistant : :connexion(int) . . . . .	54
15.2.2 WidgetDistant : :get(QString, QIcon) . . . . .	55
15.2.3 MessagesToolBar : :ajouterTelechargement(QIcon, QString, bool) . . . . .	56
<b>16 Utilisation du Logiciel</b>	<b>58</b>
<b>17 Perspectives et conclusion</b>	<b>60</b>
17.1 Perspectives . . . . .	60
17.2 Conclusions . . . . .	60
17.2.1 Fonctionnement de l'application . . . . .	60
17.2.2 Fonctionnement du groupe de travail . . . . .	60
<b>18 Bugs recensés</b>	<b>62</b>
18.1 WebBrowser . . . . .	62
18.2 FTPBrowser . . . . .	62
18.3 FrameAide . . . . .	62
18.4 Mainwindow . . . . .	62
18.5 WidgetLigne . . . . .	62
18.6 Coloration . . . . .	62
18.7 Syntaxe . . . . .	62
<b>19 Annexe</b>	<b>63</b>
19.1 ALPHA 0.1 20/03/09 . . . . .	63
19.2 ALPHA 0.2 23/03/09 . . . . .	63
19.3 ALPHA 0.3 24/03/09 . . . . .	63
19.4 ALPHA 1 25/03/09 . . . . .	63
19.5 ALPHA 1.1 28/03/09 . . . . .	63
19.6 ALPHA 1.2 28/03/09 . . . . .	64
19.7 ALPHA 1.3 31/03/09 . . . . .	64
19.8 Beta 0.1 02/04/09 . . . . .	65
19.9 Beta 0.2 15/04/09 . . . . .	66
19.10Beta 0.3 23/04/09 . . . . .	67
19.11Beta 0.4 14/05/09 . . . . .	69

# 1 Introduction

## 1.1 Généralités

Ce projet a été réalisé dans le cadre d'une unité d'enseignement, au cours du dernier semestre de la licence informatique à la Faculté des Sciences de Montpellier. Il a été proposé et encadré par le professeur et responsable de notre licence Michel Meynard. Notre groupe est composé de neuf étudiants répartis en sous-groupes.

Un projet d'une telle envergure nous a obligé à appliquer toutes les connaissances et méthodologies acquises en Programmation Objet tout au long de notre cursus universitaire. En outre, ce projet fait suite à l'unité d'enseignement Maîtrise des Médias et de la Communication dans laquelle nous avons abordé les bases du framework QT de Nokia. Enfin, la partie analyse syntaxique nous a fait mettre en pratique les concepts étudiés en cours d'Interprétation et Compilation.

## 1.2 Le sujet

Ce projet consiste à créer un éditeur de fichiers web gérant différents langages :

- HTML
- PHP
- Javascript
- CSS

Il nous a été imposé les fonctionnalités suivantes :

- Auto-complétion
- Coloration syntaxique
- Accès au manuel
- Squelette de site préexistant
- Validation HTML

## 1.3 Cahier des charges

Nous avons implémenté les fonctionnalités suivantes :

### Coloration syntaxique :

- Coloration des mots-clés du langage et des fonctions qui seront placés dans un fichier texte.
- Possibilité de laisser l'utilisateur changer les couleurs de coloration.
- L'utilisateur peut rajouter ou supprimer des mots clés dans les langages.
- La coloration peut être automatique, c'est-à-dire par reconnaissance des langages (balises), ou sélectionnée par l'utilisateur.

### Auto-complétion :

- Pour faire l'auto-complétion, nous utiliserons les fichiers textes créés pour la coloration syntaxique.
- Les variables ainsi que les fonctions créées par l'utilisateur seront aussi intégrées dans l'auto-complétion.

### Accès à tous les manuels :

- Cette partie est liée à l'auto-complétion et donc à la coloration syntaxique.
- Affichage d'une bulle d'aide lorsque la souris passera sur une fonction ou un mot clé.
- Présence dans la fenêtre d'un champ de texte permettant de chercher l'aide sur une fonction particulière.

### Plan de site :

- Fabrication d'un arbre de lien représentant l'arborescence du site créée à l'aide du logiciel.

### Web Browser :

- Création de notre propre navigateur internet gérant les marques-pages, l'historique, et étant appelé lors de l'accès aux manuels.

### Graphisme :

- Affichage du nombre de lignes et du nombre de caractères dans une page.
- Possibilité de marquer des lignes, avec différentes icônes, pour les mettre en avant.
- Gestion d'éditeurs par des onglets.
- Affichage de l'arborescence des projets contenus dans l'espace de travail.
- Ouverture d'une petite bulle lors de la reconnaissance d'un mot clé PHP ou Javascript sous la souris.

**FTP Browser :**

- Connexion FTP : envois et récupérations de fichiers.
- Connexion à l'aide d'un serveur proxy possible.
- Envoi d'un projet.

**Documentation :**

- Aide au développement de l'application, création d'une aide en ligne, explications des classes et des méthodes utilisées dans la fabrication du logiciel.

**A COMPLETER :**

- Validation W3C.
- Copie automatique des images du site dans un dossier afin de les référencer par un chemin relatif.
- Gestion de base de données, génération automatique de scripts MYSQL.
- Traduction du logiciel en plusieurs langues.

## 1.4 Syntaxe

Nous coderons en «français» c'est-à-dire que les attributs, les noms de classes, les méthodes, et les fonctions que nous utiliserons pour créer cet éditeur web auront des noms français.

**Classe :** majuscule puis minuscule et majuscule pour séparer les mots.

**Méthode :** minuscule et majuscule pour séparer les mots.

**Attribut :** minuscule puis underscore.

**indentation :** mettre le maximum d'accolades, même si il n'y a qu'une ligne de code après le for ou le if. Et les placer en dessous du for et de la dernière ligne de la boucle.

**Commentaires :** les mettre au dessus de chaque fonction, de chaque classe, dans le .h de préférence. Doxygen sera utilisé pour générer la documentation.

**Gestion des erreurs avec le format :** Erreur : courte description des erreurs, situées sur nom de la page, à telle ligne.

## 1.5 Analyse de quelques éditeurs web existants

**FrontPage :** Frontpage est un éditeur de la suite Microsoft Office, c'est donc un logiciel propriétaire. Cet éditeur est considéré comme étant plutôt destiné à des petites entreprises, notamment grâce à la gestion des tâches pour le travail par équipes. Il est de type WYSIWYG (What You See Is What You Get => ce qu'on voit est ce qu'on obtient), ce qui fait de lui un éditeur assez intuitif pour les amateurs. Il dispose entre autre d'un éditeur d'images et d'un client ftp intégrés. Néanmoins, pour des utilisateurs possédants des notions de langage html, le code généré par Frontpage est parfois superflu, et de moyenne qualité (le code n'est pas toujours conforme aux normes W3C). De plus, c'est un logiciel payant, et utilisable sous Windows (bien qu'une version Mac ait été créée).

**Adobe Dreamweaver :** Dreamweaver est aussi un éditeur de site web de type WYSIWYG possédant une licence propriétaire. C'est un des éditeurs les plus connus. Tout comme Frontpage, un utilisateur qui n'a aucune connaissance particulière concernant la conception de sites internet peut aisément utiliser Dreamweaver pour créer un site web élaboré, d'autant plus que Dreamweaver permet de faire des pages web dynamiques grâce à l'intégration d'outils PHP. Il possède aussi un client ftp. De plus, cet éditeur permet d'insérer des scripts préprogrammés qui permettent par exemple de mettre une horloge ou un compteur dans son site. Cependant, Dreamweaver produit parfois du code de mauvaise qualité, et coute plutôt cher.

**KompoZer** : KompoZer est un logiciel libre de type WYSIWYG, conçu pour être facile d'utilisation. Il est multiplateforme, et permet donc d'être installé sous Windows, Linux, ou Mac. Il possède de nombreuses fonctionnalités telles que : client ftp, gestion des feuilles de style CSS... Cependant, la dernière version de KompoZer comporte de nombreux problèmes de stabilité, ainsi qu'une interface graphique peu développée.

**Notepad++** : Notepad++ est un éditeur de texte qui gère la coloration du code html (mais aussi d'autres langages). Il est donc destiné aux personnes maîtrisant le langage HTML ou PHP. De plus, comme c'est un éditeur de texte, il ne possède pas de client ftp, mais il est gratuit.

**PHPEdit** : PHPEdit est un environnement de développement intégré, spécialisé dans le PHP. Il intègre la coloration syntaxique, ainsi que la complétion automatique des fonctions PHP et une interface personnalisable pour faciliter son usage. Néanmoins, c'est un éditeur de texte et il requiert donc certaines connaissances de langages. Enfin, il est également payant.

## 2 Organisation du projet

### 2.1 Organisation du travail

Lors de la première réunion nous avons pris connaissance du sujet de notre projet ainsi que de notre équipe. Il nous a paru nécessaire de s'organiser dès cette première réunion, c'est pour cela que nous avons décidé à l'unanimité d'élire Maneschi Romain au poste de chef de projet. Nous avons également décidé d'utiliser le langage C++ après une délibération avec tous les membres du groupe.

Enfin, afin de se rendre compte de la tâche que nous devons faire, nous avons analysé les différents éditeurs web existants sous différents systèmes d'exploitation. Après avoir pris conscience des fonctionnalités que nous voulions implémenter, nous nous sommes réunis une seconde fois pour répartir les tâches et réaliser le cahier des charges :

#### Editeur :

Coloration :

NOVAK Audrey

Syntaxe :

KÖNIG Mélanie

Auto-completion :

SAUVAN William

Auto-indentation :

AZRIA Julien

DURAND Romain

#### Accès à tous les manuels :

MAILLET Laurent

#### Plan de site :

BALIMA Dietrich

#### Web Browser, FTP Browser :

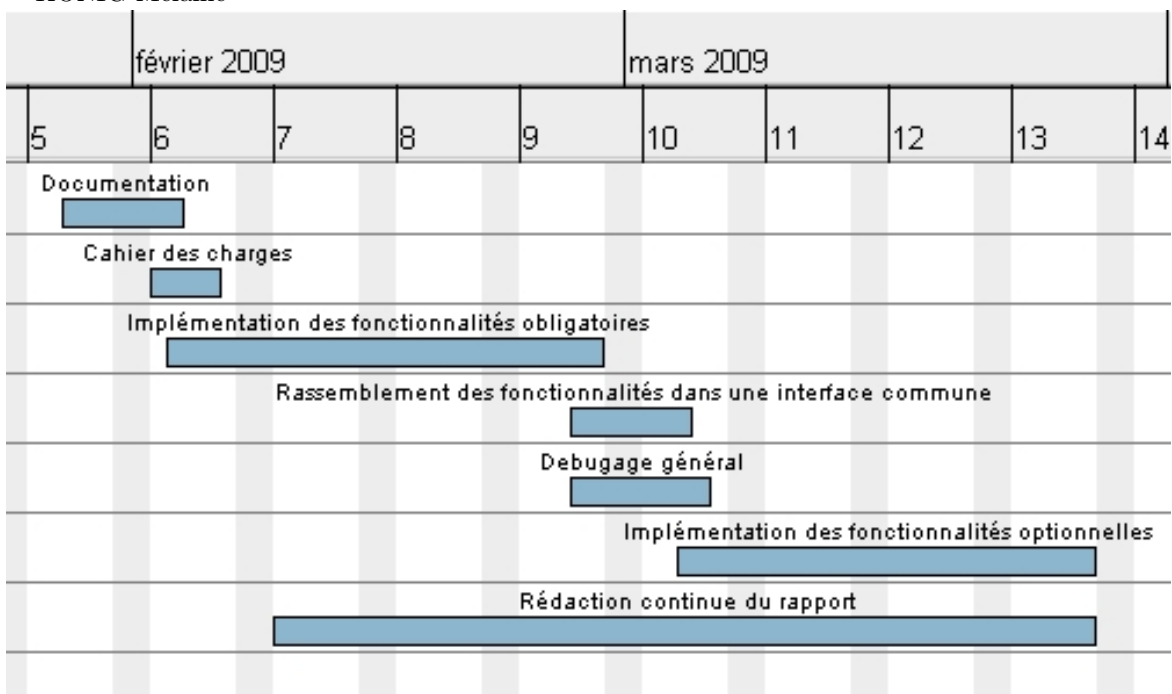
MANESCHI Romain (chef de projet)

#### Graphisme, gestion de fichiers :

FHAL Jonathan

#### Arborescence de l'espace de travail :

KÖNIG Mélanie





Après avoir réparti le travail nous avons décidé de nous voir au moins une fois par semaine. Même si tout le groupe ne pouvait être présent, ce petit rendez-vous permettait de nous tenir au courant sur l'avancée de nos parties mais aussi et surtout pour nous échanger les informations récoltées durant la semaine, sur l'utilisation de Qt. De plus cette réunion nous permettait de savoir à quel moment et comment appeler une fonction d'une autre classe.

Programmer en groupe est vite devenu notre plus gros problème : certains ne voyant pas la hiérarchie comme d'autres. Mais de petits schémas comme ceux présentés plus loin nous ont permis de clarifier la situation.

## 2.2 Choix des outils de développement

Nous avons délibéré au sujet du choix du langage. Deux choix se sont naturellement imposés à nous : C++ ou Java, les deux ayant leurs avantages et inconvénients :

Nous aurions pu choisir le langage Java en raison de sa portabilité sur les différents systèmes d'exploitation, sa gestion implicite des pointeurs et références et de son mécanisme de ramasse-miette. Cependant après avoir analysé nos besoins, nous avons choisi d'utiliser le langage C++. Notamment pour sa librairie QT qui nous offre la possibilité de créer un logiciel portable ainsi que d'utiliser de nombreux modules tel que :

- QTextEdit : un éditeur de texte nous permettant d'intégrer un colorateur syntaxique (QSyntaxHighlighter)
- QWebView : un moteur de rendu HTML (WebKit)
- QFTP : un module FTP permettant la connexion à un serveur FTP

En outre, QT Software nous propose QtCreator, un IDE spécialement développé pour cette librairie intégrant QtAssistant, la documentation officielle de QT, ainsi qu'un débogueur. Qt se charge également de gérer l'arborescence du projet au travers d'un fichier '.pro' recensant les sous-projets listés eux-mêmes dans des fichiers '.pri'.

Le chef de projet se chargeait de réunir tous les fichiers de chaque sous-groupes, afin d'en tester la compatibilité, et ainsi de renvoyer une version du projet mise à jour. Il s'occupait également d'envoyer à chacun la liste des modifications effectuées en globalité.

Pour la représentation du projet nous avons utilisé un logiciel WinDesign, conçu à la base pour réaliser des diagrammes UML. Mais notre projet étant trop conséquent, nous avons préféré ne pas respecter complètement la normalisation, afin de simplifier la modélisation, comme vous pourrez le voir sur les schémas plus loin.

## 3 Analyse du projet

### 3.1 Analyse générale

Qu'est ce qu'un éditeur web ? C'est avant tout un éditeur de texte permettant l'implémentation de pages nécessaires à la conception de sites internet. Après avoir réalisé une étude de l'existant, nous avons conclu qu'un éditeur web utile se doit de comprendre un certain nombre de fonctionnalités, telles que la coloration des langages du web(PHP, Javascript,HTML,etc), l'auto-complétion, une vérification de la syntaxe et une auto-indentation.

Mais il nous a aussi paru essentiel de rajouter des fonctionnalités supplémentaires telles que l'intégration d'un navigateur web, d'un client FTP, d'un espace de travail, d'un lien direct vers le validateur w3c, ainsi que la création du site-map automatique lié à un projet.

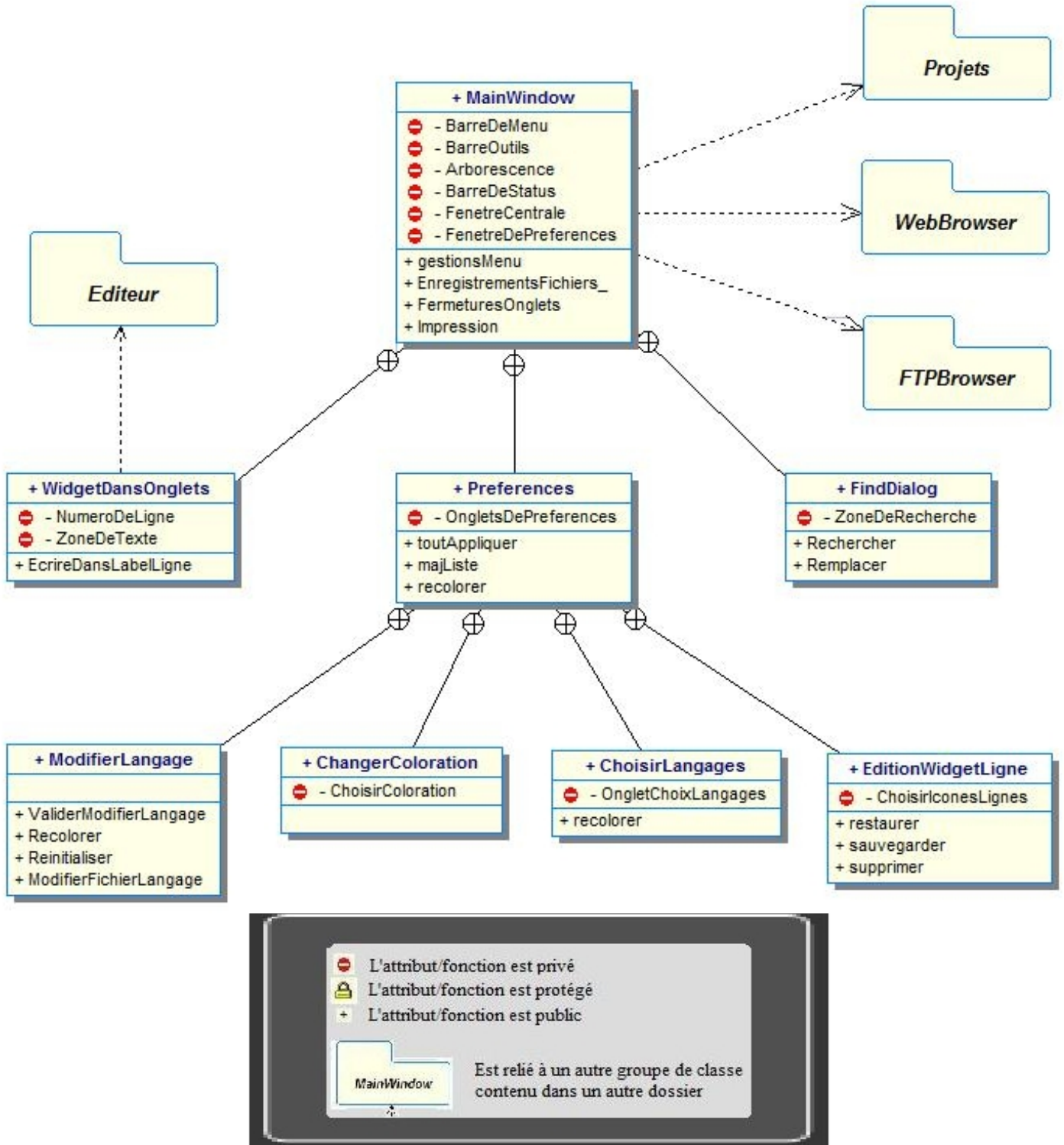
En effet, le navigateur web permet de visualiser l'évolution de la création de son site et de consulter l'aide en ligne. Un client FTP permet quant à lui de gérer directement ses fichiers sur son serveur et de rendre son site opérationnel. Le validateur w3c indique à l'utilisateur si son site est conforme aux standards du web. Le mappage du site est aussi bien utile pour les moteurs de recherche que pour le visiteur du site car il y recense tous les liens.

Ainsi, tous ces modules étant indispensables, en les réunissant au sein d'un même éditeur, ils permettent à l'utilisateur de n'avoir besoin que d'un seul logiciel pour créer son site internet.

Ci-dessous vous pourrez observer la modélisation que nous avons choisie afin de représenter nos classes. Comme vous pouvez le constater, ces schémas ressemblent fortement à de l'UML, nous nous en sommes inspirés. Cependant le projet étant assez conséquent, nous avons fait le choix de n'écrire que certaines méthodes et attributs qui nous semblaient être les plus importants. Ces diagrammes reflètent bien la hiérarchie de notre projet.

### 3.2 Première approche

Puisque nous réalisons un logiciel graphique, nous étions dans l'obligation d'implémenter une classe principale permettant l'affichage de toutes les fonctionnalités du programme. L'éditeur de texte étant la pièce maîtresse du programme, nous nous devons de le placer au centre de la fenêtre. Naturellement, un menu et des boutons permettront l'utilisation des fonctionnalités basiques d'un éditeur de texte (sauvegarde/chargement de fichier, quitter, copier/coller, nombres de lignes, etc.). Puisque nous avons décidé d'intégrer un espace de travail, nous le représenterons sous forme d'une arborescence dans une fenêtre accolée à celle de l'éditeur. Ce rôle sera attribué à la classe nommée `MainWindow`.

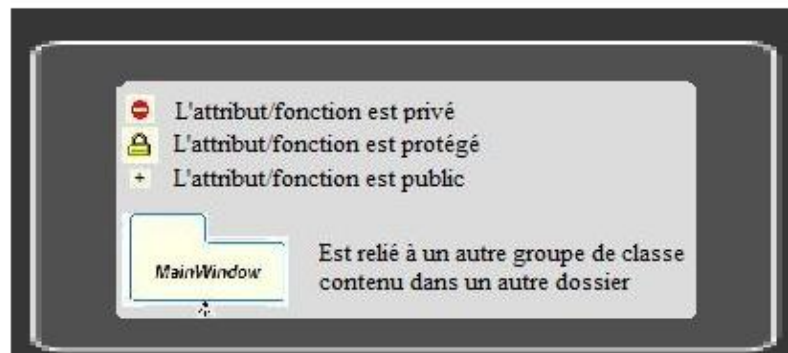
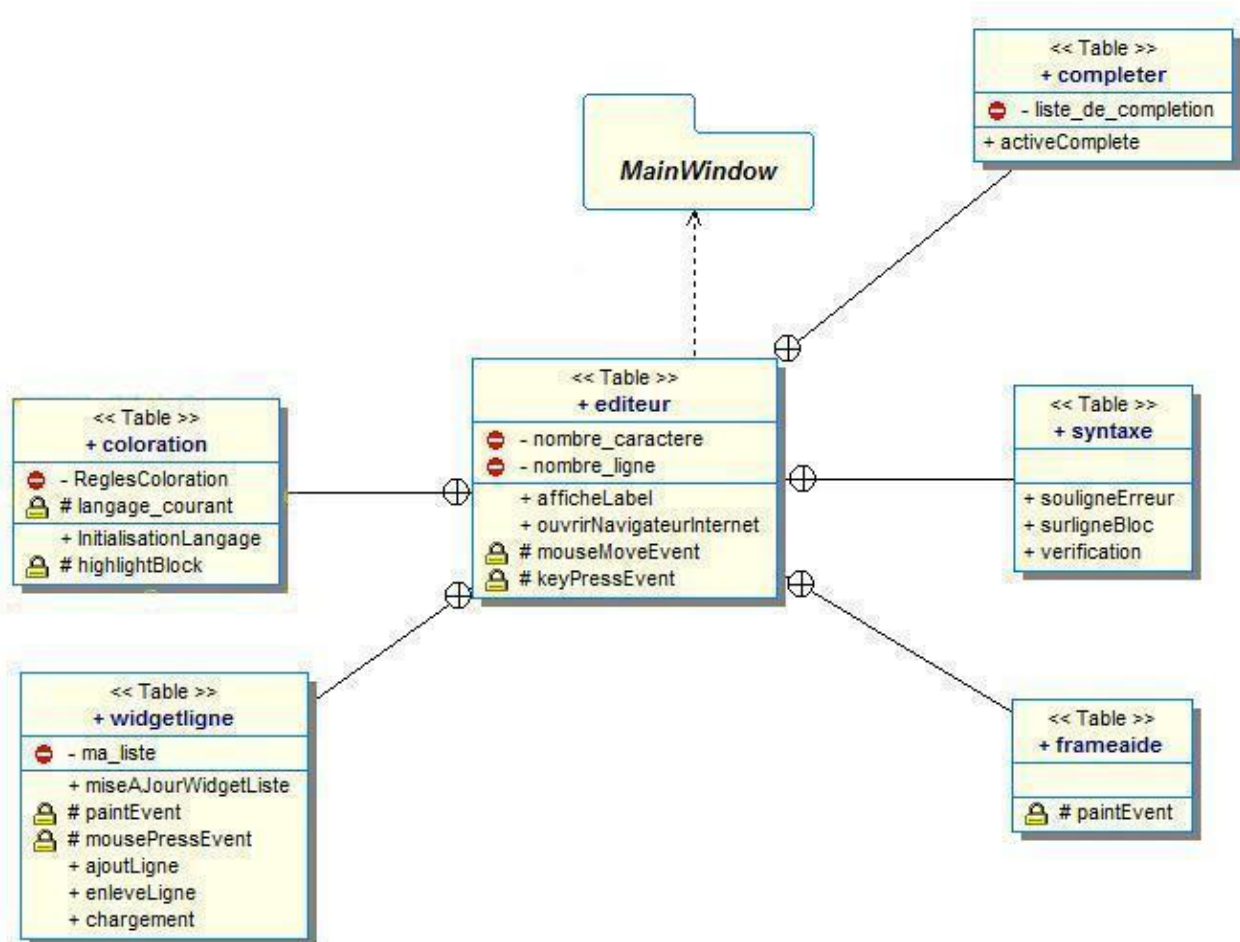


### 3.3 Editeur

L'éditeur permettra simplement d'accueillir le code tapé par l'utilisateur. Il se doit donc de contenir tous les modules facilitant la programmation de l'utilisateur dont la coloration syntaxique. Cette dernière permettra de mettre en évidence une liste de mots clés prédéfinis selon le langage de programmation en cours de saisie. Cette liste de mots clés devra être modifiable ainsi que la couleur liée à celle-ci. Le type de coloration du langage saisi pourra être modifié.

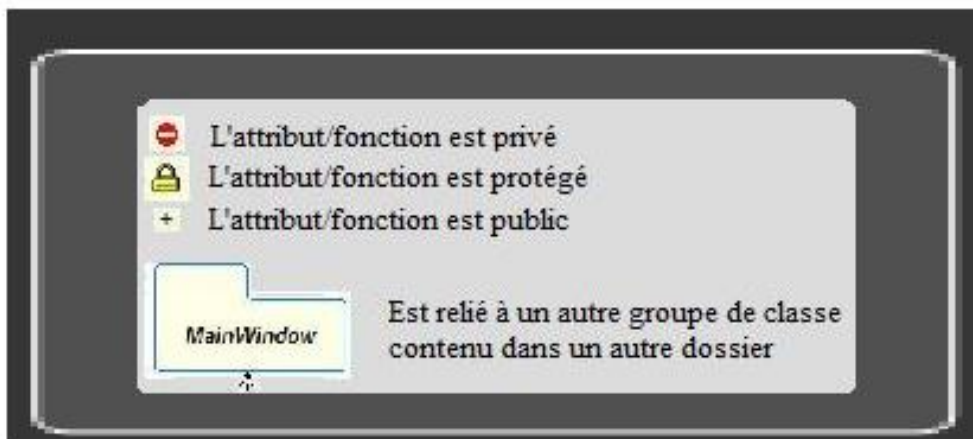
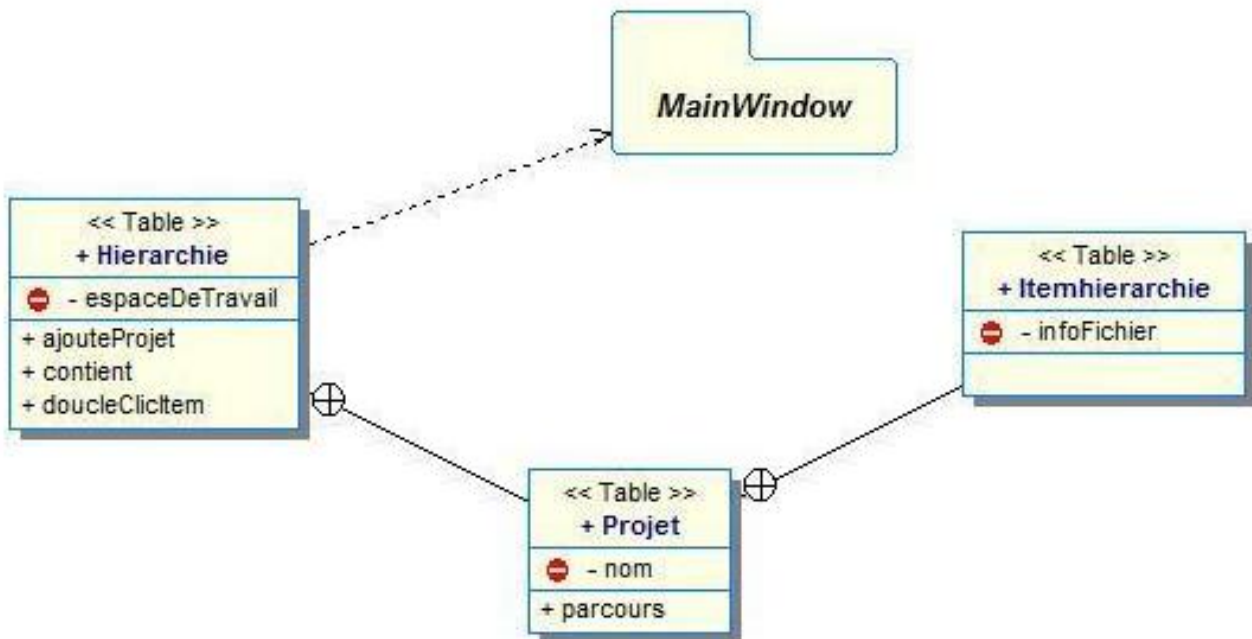
L'utilisateur aura aussi la possibilité d'utiliser un auto-compléteur, se servant des listes de mots clés prédéfinies et étant appelé au cours de la frappe. La complétion automatique enregistrera également les variables et les fonctions déclarées par l'utilisateur dans son code.

Notre éditeur se devra aussi d'avoir une vérification syntaxique minimale, par exemple une ligne n'ayant pas le bon nombre de parenthèses sera mise en évidence, probablement soulignée en rouge, à l'image d'autres éditeurs, ceci parmi les autres erreurs syntaxiques aisément repérables, telles que le nombre de paramètres d'une fonction ou les points-virgule en fin de ligne.



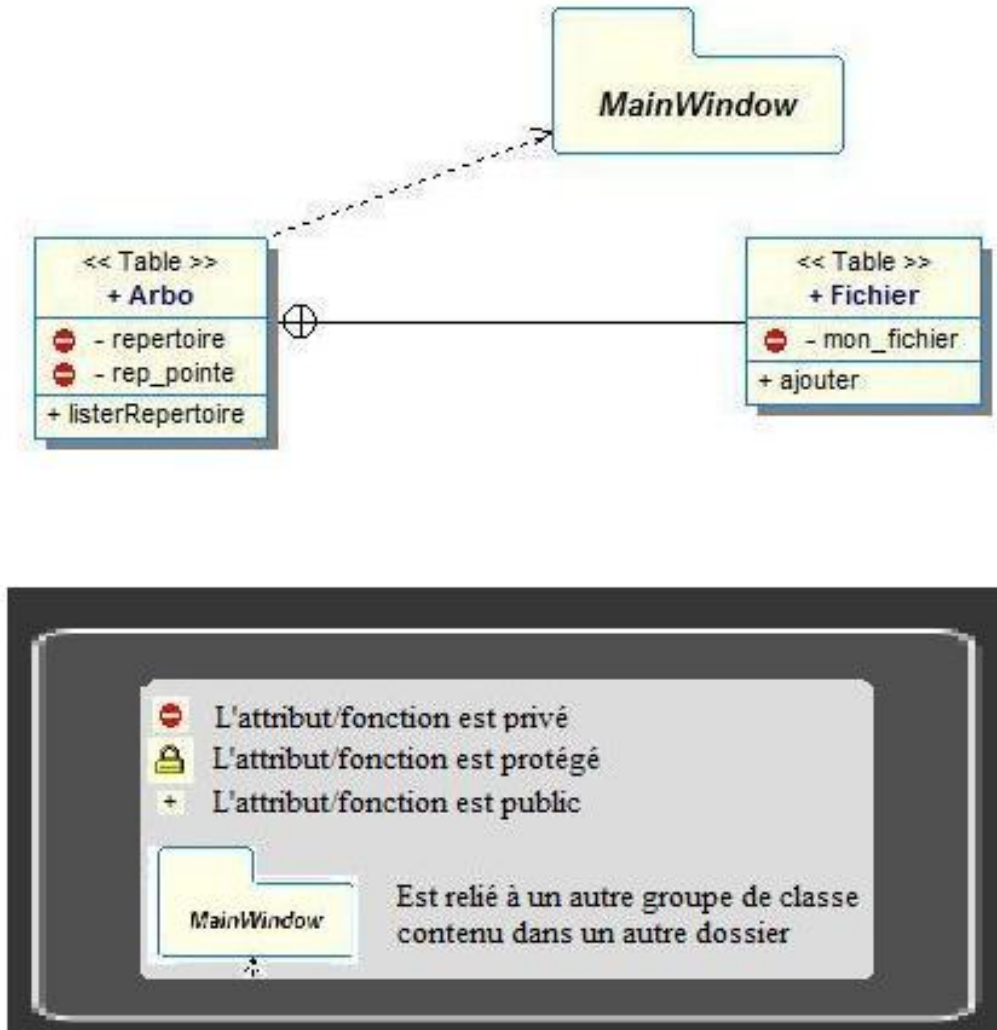
### 3.4 Hierarchie

Afin de pouvoir visualiser l'arborescence de l'espace de travail et de ne pas perturber l'utilisateur, nous utiliserons un affichage classique, listant les projets ainsi que leurs fichiers et dossiers. Il aura la possibilité de charger les fichiers directement dans l'éditeur. Cette arborescence sera contenue dans une fenêtre séparée permettant d'être redimensionnée, déplacée, ou cachée.



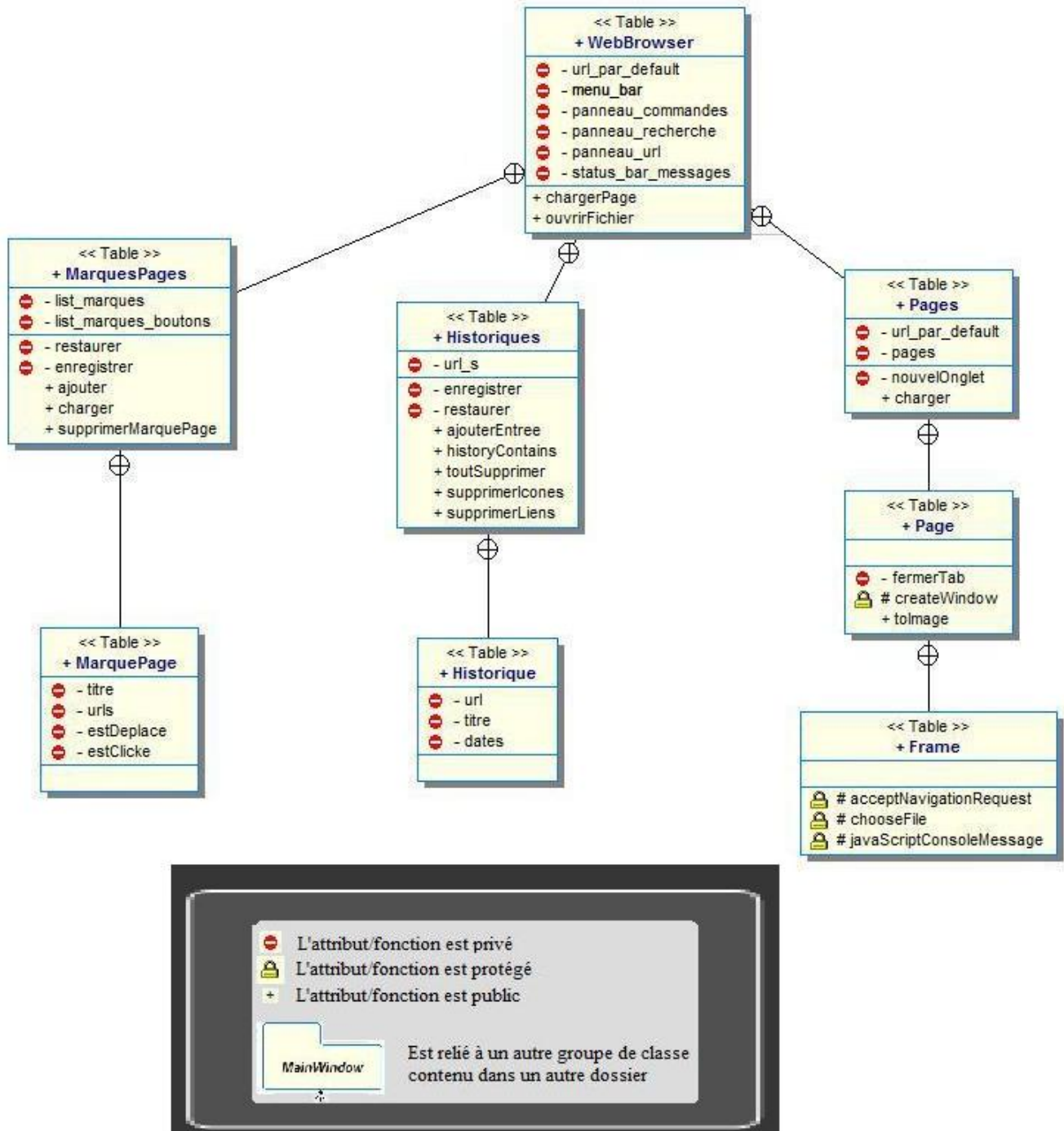
### 3.5 Site-Map

Le site-map s'appuiera sur la partie Hierarchie et créera automatiquement le fichier "siteMap.html" recensant les liens du projets. Il devra correspondre aux standards du web pour être analysé par un maximum de moteurs de recherche.



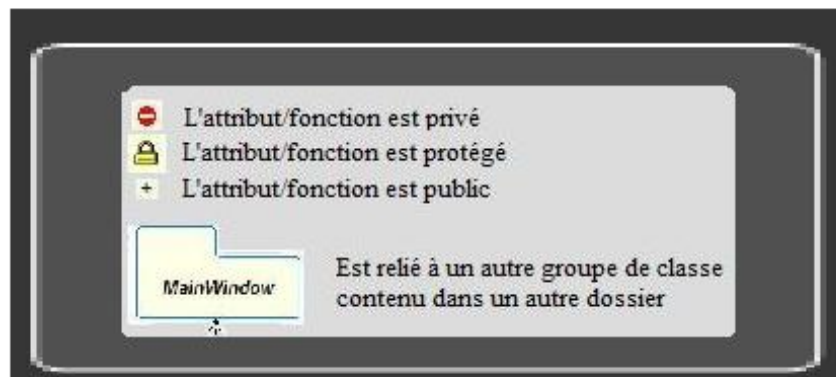
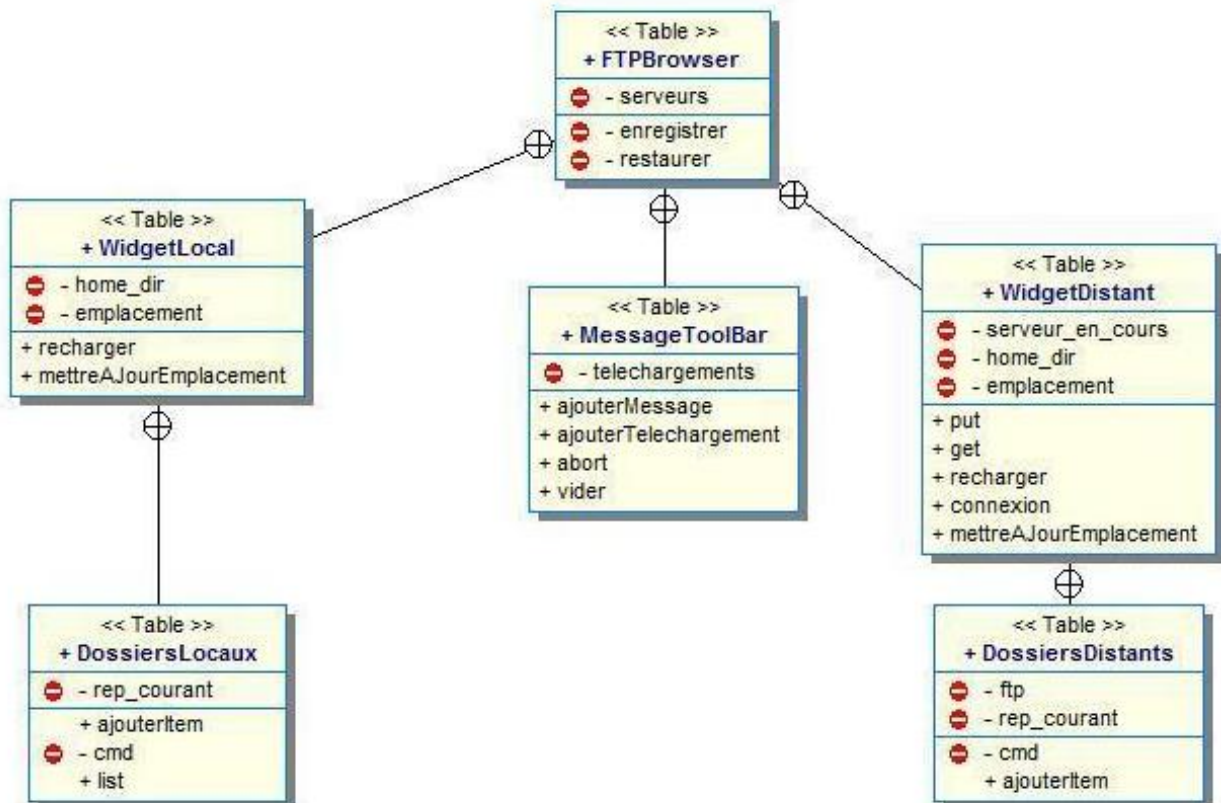
### 3.6 WebBrowser

Cet explorateur web servira essentiellement à diriger l'utilisateur sur les manuels d'aide en ligne des différents langages. Il devra donc être adapté aux normes du web. En outre, les fonctionnalités d'un navigateur internet traditionnelles seront implémentées (marques-pages, historique, gestion des cookies, etc ).



### 3.7 FTPBrowser

Ce client FTP sera la liaison entre l'espace de travail et le serveur FTP de l'utilisateur. Il aura la possibilité de se connecter au travers d'un serveur proxy. Et comme tout bon client FTP, il gèrera l'envoi et la réception de fichiers multiples.





## 4 Analyse technique

Afin de réaliser au mieux notre projet, nous avons tout d'abord identifié et séparé dans des sous-groupes les classes essentielles :

- `MainWindow` : Classe principale rassemblant les classes majeures des sous-groupes. De plus, elle gère tout l'aspect graphique, l'enregistrement, l'ouverture des fichiers et les préférences du logiciel.
- `Editeur` : Classe majeure du sous-groupe `Editeur`, incluant la coloration et la vérification syntaxique, l'auto-complétion, la frame permettant d'accéder à l'aide, la numérotation des lignes et l'ajout de signets accolés aux numéros de lignes.
- `Hierarchie` : Classe majeure du sous-groupe `Projets`, permettant de lister et d'afficher l'arborescence de l'espace de travail dans une fenêtre se trouvant à gauche de la fenêtre d'édition.
- `Arbo` : Classe majeure du sous-groupe `SiteMap`, générant un fichier `SiteMap.html` pour les moteurs de recherche internet.
- `WebBrowser` : Classe majeure du sous-groupe `WebBrowser`, incluant tous les fichiers nécessaires à la création de notre navigateur internet.
- `FTPBrowser` : Classe majeure du sous-groupe `FTPBrowser`, incorporant tous les fichiers nécessaires à la création de notre client FTP.

### 4.1 Analyse du `MainWindow`

#### 4.1.1 Partie graphique

Le `mainwindow` est divisé en 5 parties distinctes représentant les zones graphiques principales :

- La barre de menu, `QMenuBar`, permet d'appeler diverses fonctions gérant les préférences d'affichage, les gestions de fichiers, les actions principales que l'on retrouve dans tout éditeur de texte (copier/coller/couper, zoomer...). Un `QMenuBar` est composé de divers `QMenu`, servant à regrouper les différents thèmes. Ces `QMenu` contiennent des `QAction` qui appliquent les fonctions demandées.
- La barre d'outils, `QToolBar`, est également constituée de `QAction`. Elle regroupe des raccourcis vers les fonctions les plus utilisées. Le `MainWindow` contient un espace réservé spécialement aux `QMenuBar` et aux `QToolBar`. Ceux-ci une fois créés s'intègrent donc parfaitement en faisant appel aux fonctions correspondantes : `setMenuBar` et `addToolBar`.
- Le menu de gauche est un `QDockWidget` contenant un `QTreeWidget`, ce-dernier étant utile au listing de l'espace de travail. De ce fait les fichiers sont accessibles rapidement : d'un simple clic l'ouverture se fait dans un nouvel onglet.
- La partie principale de la fenêtre est un `QWidget` comprenant un `QTabWidget` générant des onglets. Chaque onglet incorpore un `widgetDansOnglet`, héritant de `QWidget`, possédant un `widgetLigne` et un `Editeur`. `widgetLigne` est la classe permettant de numéroter les lignes et de les marquer, en ajoutant des signets. L'ajout d'un signet se fait par un simple clic. Un `Editeur` est un `QTextEdit`, il sera spécifié plus loin. On trouve aussi un espace dédié à la recherche et au remplacement d'un mot dans le fichier courant. Ceci est géré par la classe `FindDialog`.
- La partie inférieure est un `QStatusBar` et affiche le nombre de lignes et de caractères du fichier de l'onglet courant.

Une section de la classe `MainWindow` est dédiée à la gestion des fichiers, plus précisément :

- L'ouverture des fichiers simples ou multiples. Il y a reconnaissance de l'extension du fichier matérialisé par une icône au niveau de l'onglet.
- L'enregistrement des fichiers se fait directement soit par l'utilisateur, soit lors de la fermeture du programme. Lors de la fermeture d'un fichier, si il contient un astérisque, alors un message prévient l'utilisateur qu'il n'a pas encore sauvegardé son travail et lui propose ainsi de l'enregistrer.
- Lors de la sauvegarde d'un nouveau fichier, une fenêtre de dialogue apparaît et permet à l'utilisateur de choisir le nom du fichier ainsi que l'endroit où il veut le sauvegarder.
- Le texte de l'onglet courant peut être imprimé grâce à la fonction prévue à cet effet.

### 4.1.2 Préférences

Cette classe rassemble dans un même widget toutes les options disponibles dans notre logiciel. En effet, elle est constituée de différents onglets contenant chacun un widget, ainsi qu'un bouton appliquer et un bouton quitter. Le bouton appliquer permet d'appliquer toutes les options en même temps pour chaque onglet d'édition ouvert.

#### Changer la coloration des mots clés :

Cette classe permet de changer la coloration des différents mots clés. Elle est intégrée dans le widget Préférences. `ChangerColoration` est construite à partir de quatre `QGroupBox` (boîtes permettant de contenir des boutons, du texte ...), une pour chaque type de langage. Ces `QGroupBox` contiennent chacune un texte correspondant à la liste de mots clés ou d'expressions régulières concernés, ainsi que pour chacune d'elle une frame colorée, un bouton G (Gras) et un bouton I (italique), qui correspondent au format associé à la liste de mots clés. La frame colorée est construite à partir de la classe `Couleur` qui hérite de `QFrame`, et qui prend comme couleur, la couleur des mots clés correspondant aux paramètres passés dans son constructeur. En cliquant sur la frame, on affiche une boîte de dialogue permettant de changer la coloration des mots clés du langage. Le widget `changerColoration` contient également un bouton valider qui permet d'appliquer tous les changements de format aux mots clés et ce pour tous les onglets d'édition ouverts. Les formats sont sauvegardés grâce à un `QSettings`, dans la base de registre pour windows, et dans un fichier `.conf` pour Linux, ils sont ainsi restaurés lors la réouverture du logiciel. Le bouton appliquer lance l'enregistrement des couleurs dans le fichier du `QSettings`, le changement automatique du format de chaque langage, la recoloration automatique de chaque mot clé, en faisant appel aux fonctions : `changerFormatPhp1(QColor coul, bool estGras, bool estItalique)` (respectivement javascript, html, css), de la classe `Coloration`. Les paramètres de ces fonctions correspondent à la couleur choisie, les booléens `estGras` et `estItalique` sont l'état des boutons du même nom, ainsi, si le bouton gras est "poussé", le booléen `est gras` sera true.

#### Modifier les listes de mots clés :

Afin de rendre notre logiciel utilisable à long terme, nous avons décidé de rendre les listes de mots clés complètement modifiables. En effet, dans le cas d'une mise à jour d'un langage, il nous a paru nécessaire que l'utilisateur puisse le faire facilement. C'est pour cela que nous avons créé ce widget. Il contient deux `QComboBox`, une zone de texte (un `QLineEdit`), un `QListWidget` et des `QPushBoutton`. Le premier `QComboBox` est une liste déroulante permettant de choisir le langage que l'on veut modifier, après avoir choisit le langage, une deuxième `QComboBox` s'affiche afin de sélectionner le fichier de mots clés auquel on pourra rajouter ou supprimer des mots clés. Le choix du fichier lancera l'affichage des mots contenus dans la liste correspondante, dans le `QListWidget`.

La coloration de mots clés répondant à des normes particulières, il a donc en fallut restreindre l'ajout en les traitant par leur expression régulière. Ainsi, avant d'afficher le bouton permettant d'ajouter, nous testons si l'expression régulière correspond à nos standards, et affichons une couleur rouge si ca ne correspond pas, et verte si c'est correct, nous affichons également un texte permettant d'aiguiller l'utilisateur sur le type d'expression attendu. L'utilisateur aura aussi la possibilité d'ajouter une liste de mots clés, séparés par des |. Les mots ainsi ajoutés se chargent dans la liste et le `QListWidget` se met à jour automatiquement, après avoir appuyé sur le bouton ajouter.

L'utilisateur pourra supprimer n'importe quel mot clé, simplement en cliquant dessus et en appuyant sur le bouton supprimer. La liste se mettra ainsi à jour et les mots clés supprimés ne se coloreront plus. Néanmoins, si l'utilisateur souhaite revenir à la configuration initiale, il a la possibilité de cliquer sur le bouton réinitialiser qui rechargera automatiquement la liste concernée. Les fichiers de réinitialisation sont situés dans un répertoire nommé : `fichiers_initiaux`.

Le clic sur le bouton appliquer appellera le slot `relancerInitialisationLangages()` de la classe `Coloration`, qui permet de vider les listes et de les recharger avec les nouveaux mots clés, et de recolorer automatiquement les mots contenus dans chaque onglet d'édition ouvert.

### Choisir le type de coloration :

Par défaut, le type de coloration syntaxique est automatique, c'est-à-dire que la coloration est appliquée en fonction des blocs détectés dans l'éditeur. Cependant, grâce à ce widget, il est possible de choisir de colorer automatiquement (par balises), et de rajouter la coloration des mots clés d'un autre langage (hors balises). Par exemple, on ouvre une balise php, on est alors dans l'état php, mais on souhaite aussi colorer les mots clés du javascript afin d'avoir une meilleure vue des différentes parties de notre code, il suffira de cocher automatique, et javascript. Il est aussi possible de choisir de colorer seulement les mots clés d'un certain langage, sans même avoir reconnu l'ouverture d'un bloc, il faudra alors décocher la case automatique, et cocher la case du langage désiré. Ce widget est constitué de QCheckBox, une pour chaque langage, une pour la coloration personnalisée et une pour la coloration automatique. En cochant la case personnalisée, on a accès aux QCheckBox des langages. La fenêtre choisir le type de coloration contient aussi un bouton appliquer qui permet d'appliquer toutes les modifications aux onglets ouverts et de recolorer automatiquement les mots en fonction des cases cochées, il fait appel aux fonctions setPhp(), et estPhp() (respectivement javascript, css et html) de la classe Coloration. Ces fonctions rappellent la boucle de reconnaissance de mots clés par rapport à leur expression régulière, dans le vecteur de règles correspondant.

### Les signets :

Lorsque l'utilisateur ouvre cet onglet, il arrive sur une fenêtre, de type QWidget, recensant tous les signets qu'il a sauvegardé sur son ordinateur. En face de chaque signet se trouvent des QCheckBox, lorsque l'utilisateur clique sur l'une d'elle, le signet est directement supprimé de son pc.

Il peut également ajouter des signets en leur donnant un nom et en désignant le chemin d'une image. Lors de l'ajout, la fonction EditionWidgetListe : :sauvegarder() est appelé. Elle permet de sauvegarder des données dans les fichiers spéciaux du système d'exploitation de l'utilisateur (ex : base de registre pour windows) en faisant appel à la classe QSettings. Ainsi il suffit uniquement de récupérer l'endroit où se trouve les signets sur l'ordinateur, pour les afficher dans la QWidget. Puis lors de l'ajout ou de la suppression d'un signet, une simple mise à jour du fichier contenant les signets est nécessaire.

## 4.2 Analyse de l'Editeur

La classe Editeur est la plus importante du projet car c'est à partir d'elle qu'ont lieu les différentes analyses sur le texte. Elle dérive de QTextEdit, c'est une zone de texte, elle appelle les fonctions nécessaires à l'analyse et la coloration syntaxique, l'auto-indentation ainsi que l'auto-complétion.

### 4.2.1 Analyse de Coloration

Comme son nom l'indique, la classe coloration permet de colorer syntaxiquement les mots clés et les caractères rentrés par l'utilisateur, au cours de la frappe. Les langages reconnus par notre éditeur seront : le Php, le Javascript, le Css et le Html, il reconnaitra également les chaînes de caractères (" ", ' '), les commentaires simples et les commentaires multiples, ainsi que les variables de type Php, les mots suivis immédiatement par une parenthèse seront considérés comme des fonctions, et seront colorés comme tel. Les mots clés sont contenus dans des fichiers textes correspondant aux différents langages et divisés en sous groupes, afin que pour un même langage, il y ait plusieurs couleurs, une pour chaque type de mots clés.

La classe coloration réimplémente la classe QSyntaxHighlighter de Qt, qui permet de définir des règles de coloration pour un QTextEdit (éditeur). Afin de pouvoir colorer du texte, il suffit de réécrire la fonction highlightBlock() en lui donnant des règles à respecter. Les règles de coloration de chaque langage sont des vecteurs de ReglesColoration, ReglesColoration est une structure qui contient une QRegExp (une expression régulière) à laquelle est associée un QTextCharFormat (format de mot pouvant contenir une couleur et éventuellement être en gras et/ou en italique). QSyntaxHighlighter reconnaît des expressions régulières qu'il faut au préalable définir dans les vecteurs de règles. Il a donc fallu utiliser la classe QRegExp de Qt, qui permet de gérer tout ce qui est expression régulière. L'assistant de Qt liste tout ce qui est nécessaire dans la création de celles-ci.

Nous avons décidé que l'utilisateur de notre éditeur pourrait rajouter ou supprimer des mots clés, c'est pour cela qu'il nous a paru judicieux de placer les mots clés dans des fichiers textes. Lors de la construction de l'éditeur, les fichiers de mots clés sont lus, et les mots qu'ils contiennent sont copiés dans des listes correspondant à leur langage.

Après avoir instancié tous les mots clés et les expressions régulières que l'on souhaite être reconnus, tout ce qui sera tapé dans l'éditeur sera analysé par cette classe au cours de la frappe.

Le type de langage sera défini par un entier (int) contenu dans une énumération nommée Etat. Cet entier correspond à l'état du bloc dans lequel on se trouve. Un bloc est constitué de texte construit sur plusieurs lignes et qui possède un certain état, un début de bloc (<?php pour le php) et une fin de bloc (?> pour le php). Un QTextEdit possède par défaut un état égal à -1, ce qui correspond à la non-présence de début de bloc, et donc aucun type de langage particulier.

Une fois que le bloc est ouvert, une boucle se charge de parcourir le vecteur de règle de coloration correspondant à son état, si ce que tape l'utilisateur coïncide avec une expression régulière du vecteur, on lui applique le format correspondant, et ce jusqu'à la fermeture du bloc. Lorsqu'un bloc est fermé, l'état revient à -1 (nous l'avons appelé état normal), dans lequel les commentaires et les chaînes de caractères seront colorés.

Cette classe contient aussi des méthodes utiles pour d'autres classes (entre autres : classe éditeur, classe préférences), telles que :

- Fonction utilisée pour l'affichage de la bulle d'aide sous la souris, qui renvoie le type de langage du mot présent sous la souris, en fonction de l'état du bloc dans lequel il se trouve.
- Les différentes fonctions permettant de changer les formats des mots clés, le type de coloration choisit, qui sont utilisées dans les modifications des options de notre éditeur ;

#### 4.2.2 Analyse de WidgetLigne

On trouve également une fonctionnalité permettant d'afficher le numéro des lignes. C'est simplement une QWidget dans laquelle on dessine le numéro de la ligne correspondant au bloc en face duquel on se trouve. En effet comme expliqué pour la coloration, un QTextEdit est composé de blocs. Chaque ligne du QTextEdit est un bloc. En calculant la position du bloc affiché à l'écran il est donc aisé de dessiner le numéro de la ligne correspondant au bloc. Cela se réalise dans la fonction paintEvent() de la classe WidgetLigne.

Enfin l'Editeur permet de récupérer le mot courant (selon si on se déplace avec la souris ou avec le clavier), de l'afficher et d'ouvrir une fenêtre d'aide si c'est un mot clé du Php ou du Javascript. Le QTextEdit permet de récupérer le curseur tout simplement grâce à des fonctions préexistantes. Il suffit pour cela d'appeler la fonction QTextEdit : :textCursor() si on se déplace au clavier, sinon si la souris est active, il faut alors récupérer la position de la souris (dans la fonction Editeur : :mouseMoveEvent(QMouseEvent\*) on peut récupérer la position de la souris) puis de lancer un appel à QTextEdit : :cursorForPosition(position\_souris). Une fois cela réalisé il est simple de récupérer le mot positionné sous le curseur. Le mot récupéré sera ainsi comparé à la liste des mots clés définie par la classe Coloration puis à l'état du bloc dans lequel il se trouve. Si c'est un mot de php ou de javascript, alors lors de l'appui sur la touche F1 le navigateur internet sera lancé sur la page d'aide correspondante au mot clé.

### 4.3 Analyse du WebBrowser

#### 4.3.1 Introduction

Créer un WebBrowser n'est pas chose facile. La preuve étant Internet Explorer, l'un des navigateur internet le plus utilisés au monde, qui ne passe le test du W3CValidator qu'à 33%, et contient un nombre incroyable de bugs. Heureusement Qt met à disposition, depuis sa version 4.5, le module WebKit : un moteur de rendu libre et puissant. Ainsi en quelques classes nous pouvons créer un navigateur internet simple mais d'autant plus puissant.

En effet ce module regroupe en quelques classes tout un éventail d'options. Il ne nous reste qu'à réimplémenter ces classes pour les adapter à nos besoins. Nous noterons entre autre, un widget QWebView permettant d'afficher graphiquement une page. La fonction QWebView\* createWindow(QWebPage : :WebWindowType) ; est appelée lorsqu'une page demande la création d'une fenêtre (pop-up, alert(), ...), puisque WebKit a besoin de cette fonction nous devons la réécrire. De plus, la classe QWebPage permet d'ouvrir une fenêtre pour choisir un fichier (pour les formulaires) ou encore de gérer les cookies en accédant aux requêtes envoyées et

reçues.

Dans un autre registre nous remarquerons la présence de la classe `QWebHistoryInterface` permettant de gérer un historique en appelant `void addHistoryEntry(const QString&);` lors d'un chargement de page, ou encore `bool historyContains(const QString&) const;` vérifiant l'existence d'une url. Mais ces deux fonctions n'étant pas assez développées pour notre utilisation nous avons décidé de passer par d'autres fonctions. Néanmoins nous avons gardé l'héritage pour de futures mises-à-jour du WebKit.

Pour le côté graphique, il n'y a aucun problème, puisque Qt est là pour ça. Nous pensons tout naturellement à un `QTabWidget` pour afficher plusieurs `QWebView` par un système d'onglets.

#### 4.3.2 Analyse de l'historique

Plutôt que d'utiliser un banal tableau pour afficher l'historique nous avons décidé d'essayer de le représenter sous forme d'une frise. Dès lors il a fallu penser à créer une icône d'une page visitée, afin de pouvoir l'afficher. Heureusement Qt met à notre disposition une fonction `render()` sur tous ces widgets ce qui permet de réaliser un screen du widget. De plus nous pouvons gérer ces images comme des `QLabel`, qui sont des enfants de `QFrame`, qui peuvent être réimplémentés, pour reconnaître les clics et autres survols de la souris. Ainsi un petit système de frise fût rapidement créé.

#### 4.3.3 Analyse des marques-pages

Un marque page n'est ni plus ni moins qu'un bouton permettant de rejoindre un site fréquemment visité sur un simple clic. Mais de nos jours c'est aussi et surtout un système permettant de glisser et de déposer ces boutons. Qt gère parfaitement le glisser-déposer pour tous ces widgets en réimplémentant simplement certaines méthodes de celui-ci. Ainsi `mousePressEvent(QMouseEvent*)` permet de savoir si un utilisateur clique sur notre widget, puis `startDrag(QMouseEvent*)` permet de le lancer visuellement en créant une image de l'élément sous la souris. Enfin `dropEvent(QDropEvent*)` permet de savoir si l'utilisateur relâche la souris et veut donc poser l'élément glissé, dans la zone pointée par la souris.

Ces marques-pages devront être sauvegardés en dur, car ils seront réutilisés à chaque ouverture du WebBrowser. Pour ce faire Qt nous propose un moyen puissant et facilement utilisable, les `QSettings`. Cette classe permet de sauvegarder des variables en dur, qu'elles soient d'un type primitif ou complexe. Mais ce système repose sur la classe `QVariant` pour encoder et décoder les données, notre structure devra donc contenir des structures contenues dans `QVariant`. Ainsi un marque-page peut-être vu comme un titre et une url. Mais il faut pouvoir aussi créer un dossier de marques-pages donc notre structure deviendrait un titre et un ensemble de titres et d'urls. Pour ce faire, nous utiliserons un `QString` pour le titre et un `QMap` pour les titres et urls. Ainsi un marque-page peut être simple, si le `QMap` contient la clé titre, sinon c'est un dossier. Il suffira de créer un `QSetting` pour stocker tous nos marques-pages.

### 4.4 Analyse du FTPBrowser

Tout comme le module WebKit fournit par Qt pour afficher une page internet, un module `Ftp` permet de se connecter, de lister un répertoire et d'envoyer ou recevoir un fichier très facilement. Ainsi en créant un objet `QFtp` et en lui passant l'identifiant, le mot de passe et d'autres informations, celui-ci se charge de presque tout.

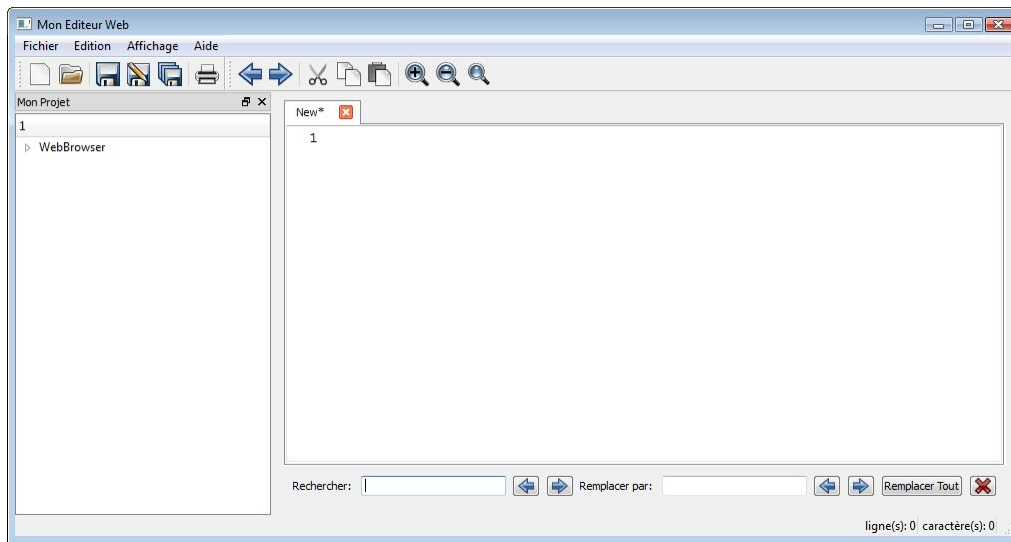
Lorsque l'utilisateur sélectionne un seueur dans sa liste, on lance la connexion, puis en connectant le signal `listInfo(const QUrlInfo &)` de la classe `QFtp` avec un de nos slots, nous récupérerons élément par élément le contenu du dossier distant. De plus, un signal `stateChanged(int)` nous permet de savoir si la connexion a réussi ou non. Enfin les méthodes `put(QFile, QString)` et `get(QFile, QString)` permettent de récupérer ou d'envoyer un fichier.

Ce client ftp se veut minimaliste tout en étant pratique, c'est la raison de l'absence d'une barre de menu ou encore d'options superflues. Pour cela certains boutons sont réutilisés pour plusieurs actions, car dans un ftp on ne peut réaliser certaines actions qu'en fonction de l'état dans lequel on se trouve. Cela est possible par la méthode `disconnect()` présente dans tous les `QObject`, qui détruit les relations signaux->slots avant de pouvoir les remplacer. Ensuite, une simple modification de l'image et du texte du bouton permet à l'utilisateur de ne pas se perdre.

## 5 Développement du MainWindow

### 5.1 Introduction

Le développement du MainWindow se fait sur toute la durée du projet. Il est complètement réalisé en collaboration avec tous les membres du groupe puisqu'à chaque nouvelle fonctionnalité, il faut ajouter des fonctions, des slots ou des attributs dans la classe et donc réorganiser le tout. Nous avons deux manières d'aborder la fenêtre principale : soit utiliser un QWorkspace permettant de gérer une fenêtre par fichier ouvert, soit utiliser un QMainWindow permettant d'afficher les fichiers dans une seule fenêtre centrale. Nous avons opté pour le QMainWindow.



### 5.2 Fonctions

#### 5.2.1 MainWindow : :gestionMenuBarre

**Données :**

- Attributs : menu\_bar, menu\_fichier, menu\_edition, menu\_affichage, menu\_aide

**Résultat :** cette fonction est basique, elle sert à ajouter des menus dans la barre de menu. On fait de même pour les fonctions gestionMenu, gestionBarreOutils et gestionBarreOutilsItems.

```
void MainWindow::gestionMenuBarre () {
    menu_bar->addMenu(menu_fichier);
    menu_bar->addMenu(menu_edition);
    menu_bar->addMenu(menu_affichage);
    menu_bar->addMenu(menu_aide);
    menu_bar->show();
    setMenuBar(menu_bar);
}
```

#### 5.2.2 MainWindow : :gestionMenuActions

**Données :**

- Attributs : action\_menu\_nouveau, action\_menu\_ouvrir...

**Résultat :** cette fonction permet d'ajouter des actions lorsque l'utilisateur clique sur un item du menu ou de la barre d'outils. Le clic appelle une fonction 'SLOT' particulière. Ici, les items comportent un icône, et un raccourci clavier.

```

void MainWindow::gestionMenuActions () {
    // FICHIER
    action_menu_nouveau = new QAction(QIcon(":/Menu/nouveau.png"),
                                         "Nouveau",
                                         menu_fichier);
    action_menu_nouveau->setShortcut(QKeySequence("Ctrl+N"));
    QObject::connect(action_menu_nouveau, SIGNAL(triggered()),
                    this, SLOT(fichierNouveau()));

    // Pareil pour toutes les autres actions comme pour ouvrir
    // un fichier, l'enregistrer...
}

```

### 5.2.3 MainWindow : :fichierNouveau

#### Données :

– Attributs : WidgetDansOnglet

**Résultat** : cette fonction sert, comme son nom explicite l'indique, à créer un nouveau fichier. Elle gère les noms des onglets comme par exemple 'New\*', New(2)\* grâce notamment aux expressions régulières... Et ajoute cet onglet dans le QTabWidget et met à jour le nombre de caractères et de lignes dans la barre de statut.

```

void MainWindow::fichierNouveau () {
    WidgetDansOnglet * newInterieurOnglet = new WidgetDansOnglet(tab_widget);
    QRegExp rx("New\\*|New\\([0-9]*\\)\\*");

    // Parcours tous les onglets ouverts
    int nbOnglets = tab_widget->count(); // Compte le nb d'onglets
    int nbNew = 0; // compteur du nb de nouveau fichier
    for(int i=0; i<nbOnglets; i++) {
        if(rx.exactMatch(tab_widget->tabText(i))) {
            nbNew++;
        }
    }
    // Si aucun porte le nom "New*" alors
    if(nbNew==0) {
        tab_widget->addTab(newInterieurOnglet, "New*");
    }
    else {
        QString str;
        str.setNum(nbNew);
        int page_courante = tab_widget->addTab(newInterieurOnglet,
                                             "New("+str+")*");
        tab_widget->setCurrentIndex(page_courante);
        newInterieurOnglet->getTextEdit()->getColoration()->
            setJavascript(mes_preferences->getChoisirLangage()->
                          estJavascriptSelectionne());
        newInterieurOnglet->getTextEdit()->getColoration()->
            setCss(mes_preferences->getChoisirLangage()->
                  estCssSelectionne());
        newInterieurOnglet->getTextEdit()->getColoration()->
            setPhp(mes_preferences->getChoisirLangage()->
                  estPhpSelectionne());
        newInterieurOnglet->getTextEdit()->getColoration()->

```

```

        setAutomatique(mes_preferences->getChoisirLangage()->
estAutomatiqueSelectionne());
    }
QObject::connect(tab_widget, SIGNAL(currentChanged(int)), this,
    SLOT(quandChangeOngletCourant(int)));
QObject::connect(newInterieurOnglet->getTextEdit(),
    SIGNAL(textChanged()), this, SLOT(modifNomTab()));
QObject::connect(newInterieurOnglet->getTextEdit(),
    SIGNAL(textChanged()), this, SLOT(modifNbLignes()));
QObject::connect(newInterieurOnglet->getTextEdit(),
    SIGNAL(textChanged()), this, SLOT(modifBarreStatus()));
}

```

#### 5.2.4 MainWindow : :enregistrer

##### Données :

– Attributs : QFileInfo, QFile

**Résultat** : cette fonction permet d’enregistrer dans un fichier le contenu de l’onglet courant. Si ce fichier est un fichier importé, alors on écrase l’ancien fichier et on enlève l’astérisque montrant que le fichier n’est pas enregistré. Sinon on appelle la fonction enregistrerSous() permettant de choisir l’endroit où on veut l’enregistrer.

```

void MainWindow::enregistrer() {
    interieur_onglet = dynamic_cast<WidgetDansOnglet*> (this->tab_widget->
currentWidget());

    if(interieur_onglet)
    {
        // On recupere le chemin du fichier pour voir si ce
        // fichier a ete importe
        QFileInfo* fichier = interieur_onglet->getInfoFichier();

        if(fichier != NULL && fichier->absoluteFilePath()!="") {
            QFile sauvegarde(fichier->absoluteFilePath());
            if(sauvegarde.open(QFile::WriteOnly | QIODevice::Text))
            {
                QTextStream out(&sauvegarde);
                out << interieur_onglet->getTextEdit()->toPlainText();
                // on eleve l'etoile car l'enregistrement est reussi
                changeNom(QString());
            }
        }
        else {
            enregistrerSous();
        }
    }
}

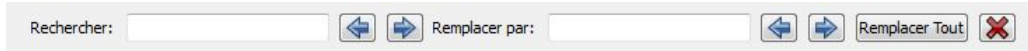
```



## 6 Développement de la barre de recherche

### 6.1 Introduction

La barre de recherche permet de rechercher un mot dans l'onglet courant avec la possibilité de le remplacer. Nous avons décidé de l'incorporer directement à la fenêtre principale et non d'en ouvrir une indépendante pour que l'utilisateur puisse voir le mot directement dans le fichier ouvert. La barre se trouve sous les onglets.



### 6.2 Fonctions

#### 6.2.1 FindDialog : :remplacer

**Données :**

- Paramètres : QString mot

**Résultat :** Cette fonction permet de remplacer le mot recherché par le mot passé en paramètre. On sélectionne le mot sous le curseur, on l'efface puis on insère le nouveau mot.

```
void FindDialog::remplacer(QString mot) {
    if(this->getLineEditRechercher()->text()!="") {
        parent = dynamic_cast<MainWindow*> (this->getParent());
        if(parent) {
            QTextCursor curseur = parent->getInterieurOnglet()->
                getTextEdit()->textCursor();
            if(curseur.hasSelection()) {
                curseur.select(QTextCursor::WordUnderCursor);
                curseur.removeSelectedText();
                curseur.insertText(mot);
            }
        }
    }
}
```

## 7 Développement de Editeur

### 7.1 Introduction

Comme nous l'avons vu le module Editeur regroupe tout ce qui est en rapport avec l'analyse du texte : la coloration et l'analyse syntaxique, l'auto-complétion, ou encore l'auto-indentation. Ainsi au moment d'aborder la partie pratique de ce projet nous nous sommes heurtés à plusieurs problèmes.

Dès le début de l'implémentation de la classe Editeur, un souci s'est posé : quels moyens utiliser pour afficher correctement les lignes sur le côté ? En effet la classe QTextEdit telle quelle, ne permet pas d'y ajouter dans son sein d'autres fenêtres ce qui rendait la tâche difficile.

Après quelques recherches nous avons trouvé un moyen efficace de résoudre ce problème : en effet un QTextEdit est composé de blocs sur chacune des lignes qui le compose. Ainsi en parcourant ces blocs, en retrouvant leurs positions dans la zone de texte et en s'aidant de la barre verticale servant à faire défiler le texte il est possible de retrouver le numéro des lignes. A partir de cela, il a donc été décidé de séparer la zone de texte et la zone des lignes. Il suffit juste de lier la zone de texte avec la zone contenant les lignes afin de la mettre à jour.



```
51         while($donnee=mysql_fetch_array($resultat))
52         {
53             $change = $_POST['id'].$donnee['id'];
54             $id = $_POST['valeur_id'].$donnee['id'];
55             $sql="UPDATE $type SET
agrandissement='$change' WHERE id='$id'";
56             mysql_query($sql);
57         }
58         $reussite = 'Insertion reussie';
59         mysql_close($lien); //fermeture mysql
60     }
61     elseif(isset($_POST['marque_voiture_neuve']))
62         $erreur = 'Elements manquants dans le formulaire';
63
64     ?>
65
66     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
67         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
68     <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
69         <head>
70             <title>Bienvenue sur nomdusite</title>
71             <meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-15" />
72             <link rel="stylesheet" type="text/css"
href="../styles.css">
73         </head>
74         <body>
75     <?php
76         include_once('entete.php');
```

### 7.2 Fonctions de l' Editeur

#### 7.2.1 Editeur : :recupereUrl()

Données :

- Attributs :
  - QString selection : texte sélectionné lorsque le curseur ou la souris est immobile.

Résultat : Récupère le type du mot sélectionné, on construit une url à partir de ce mot et on la renvoie. Elle sera récupérée et permettra de lancer le navigateur sur la page désirée.

```
// expression reguliere correspondant a une variable
QRegExp expression_reguliere (" [a-zA-Z0-9_]* ");
QString url = selection; // on recupere le texte selectionne
QString tmp = selection;
```

```

QString type;
tmp.prepend("\\b").append("\\b");
if(this->hasFocus() &&
    !selection.isEmpty() &&
    expression_reguliere.exactMatch(selection) )
{
    // fonction nous renvoie le type de langage auquel appartient le mot cle
    type = this->coloration->langage(tmp, getBlockSousSouris());
    if(type == "php")
        url.prepend("$:");
    else if(type == "javascript")
        url.prepend("js:");
    else
        url = "";
}
else
url = "";
return url;

```

### 7.2.2 WidgetLigne : :paintEvent()

#### Données :

- Paramètres :
- QPainter \*e
- Attributs :
- int m\_lineNumber : numéro de la ligne à afficher
- QTextBlock block : bloc de texte
- int contentsY : haut de la page
- qreal pageBottom : bas de la page

**Résultat** : Affiche les lignes correspondantes : On parcourt le texte du premier bloc du texte jusqu'au dernier ou jusqu'à ce qu'on dépasse la page visible. Pour chaque bloc on cherche à savoir s'il est visible dans la page, s'il ne l'est pas on passe au bloc suivant. Une fois qu'un bloc correspond aux critères il suffit d'y dessiner en face le numéro de la ligne : position nous donne cet élément. Ensuite il ne reste plus qu'à faire correspondre la taille de la ligne à dessiner avec la position du bloc, la taille du texte, etc.

```

QPainter p(this);
int m_lineNumber = 1;
const QFontMetrics fm = fontMetrics(); //police
const int ascent = fontMetrics().ascent();
// on recupere le premier bloc
QTextBlock block = editeur_texte->document()->begin();
int contentsY = editeur_texte->verticalScrollBar()->value();
qreal pageBottom = contentsY + editeur_texte->viewport()->height();
//on parcourt les bloc
for ( ; block.isValid(); block = block.next(), ++m_lineNumber )
{
    QTextLayout* layout = block.layout();
    // point en haut a gauche du bloc
    QPointF position = layout->position();
    // rectangle contenant le texte
    const QRectF boundingRect = layout->boundingRect();
    // le bloc n'est pas encore visible
    if ( position.y() + boundingRect.height() < contentsY )
        continue;
}

```

```

//le bloc n'est plus visible il est en dessous dont on arrete
if ( position.y() > pageBottom )
    break;
const QString txt = QString::number( m_lineNumber );
// on dessine le texte
p.drawText(position_texte , qRound(position.y()) - contentsY + ascent , txt);
}
p.end();

```

### 7.2.3 WidgetLigne : :chargement()

**Données :** Aucunes

**Résultat :** On récupère les signets conservés sur l'ordinateur de l'utilisateur et on les ajoute automatiquement dans la liste des signets.

```

//ouverture des donnees correspondante sur l'ordinateur
QSettings settings("MMW", "WidgetListe");
// on lit l'array WidgetListe contenant les signets
int size = settings.beginReadArray("WidgetListe");
for (int i = 0; i < size; i++) {
    settings.setArrayIndex(i);
    // on recupere le texte du signet numero i
    QString texte = settings.value(QString("texte")).toString();
    // on recupere le chemin de l'image du signet numero i
    QString chemin_image = settings.value(QString("chemin_image")).toString();
    ajoutWidgetListe(new QAction(this), texte, chemin_image);
}
settings.endArray();

```

### 7.2.4 Erreur : :ecritErreur()

**Données :**

- Paramètres :
- QString nom\_classe : nom de la classe où s'est produit l'erreur
- QString nom\_fonction : nom de la fonction où s'est produit l'erreur
- int ligne : numéro de la ligne où s'est produit l'erreur

**Résultat :** On construit une phrase à partir des données envoyés à la fonction et on l'insère dans le fichier log.txt

```

QDateTime date = QDateTime::currentDateTime();
QString tmp = QString(systeme_relation_fichier).append("Erreurs/log.txt");
QFile fichier(tmp);
if(fichier.open(QFile::WriteOnly | QFile::Append))
{
    QTextStream out(&fichier);
    out<<date.toString("Le_dd.MM.yyyy_a_hh:mm:ss_")<<"Erreur_sur_le_fichier_"
        <<nom_classe<<"_avec_la_fonction_"<<nom_fonction;
    if(ligne != -1)
        out<<"_a_la_ligne_"<<ligne;
    out<<endl;
    fichier.close();
    return true;
}
return false;

```

## 8 Développement de Coloration

### 8.1 Introduction

Cette classe contient toutes les méthodes permettant de construire des expressions régulières qui seront colorées en fonction du format qui leur sera appliqué. Notre première idée fut de mettre à profit nos cours de compilation dans lesquels nous avons utilisé Flex, un outil d'analyse lexicale, qui pourrait reconnaître les expressions régulières à colorer. Cependant, après avoir fait plusieurs recherches, nous nous sommes aperçu que Qt proposait un exemple simple de coloration syntaxique de code c++, basé sur l'implémentation de la classe `QSyntaxHighlighter`. Cette classe reconnaît des mots sous forme d'expression régulière. Il a alors fallu apprendre et s'adapter à la déclaration des expressions régulières de Qt. Nous nous sommes donc aidé de cet exemple pour construire notre propre classe de coloration. Le principal problème rencontré a été le nombre de langage que nous traitons. En effet, dans une même fenêtre d'édition, il est possible de colorer du php, du javascript, du css, et du code html (dans des blocs non imbriqués). C'est ici que la notion de bloc a été très utile. De plus, comme nos mots clés sont stockés dans des fichiers textes, nous avons dû aussi créer des fonctions permettant de lire et de transformer ses mêmes mots clés en expression régulière afin de les mettre dans des `QStringList` (liste de mot clé) c'est ce que fait la fonction `ajouteMotCleFichier(const QString &chemin, QStringList &liste, QTextCharFormat format)`.

### 8.2 Fonctions de Coloration

#### 8.2.1 Coloration : `:ajouteMotCleFichier`

##### Données :

- Paramètres : Cette fonction prend en paramètre le chemin du fichier texte, la liste dans laquelle ajouter les mots clés, et le format à leur appliquer.
- Autre fonction appelée : `renvoieListe(QStringList)` cette fonction renvoie sous forme de `QString` le type de la liste passée en paramètre.

**Résultat** : Les fichiers textes sont de la forme : `mot-clé1|mot-clé2|...|mot-cléN`, cette fonction parcourt le fichier texte passé en paramètre et réalise un découpage dès qu'elle trouve le caractère `|` grâce à la fonction `split(QString)` de Qt. Puis elle rajoute devant et après chaque mot : `"b"`. Cet ajout définit les bornes du mot, c'est à dire que grâce à ça, le `QSyntaxHighlighter` sait qu'il doit reconnaître seulement les mots en tant que tel.

```
include          aaaincludeaaa
aaa<html>aaa|
```

Dans cet exemple on se rend bien compte que le mot `include` est coloré car il a été précédemment "borné", et que le mot `"aaaincludeaaa"` n'est quant à lui pas coloré. Pour une balise html, le principe de borne est différent, en effet, en html, on doit pouvoir écrire des caractères accolés aux balises, ainsi, il ne faut pas rajouter de bornes lors de la lecture des fichiers de mots clés html (seule la fonction d'ajout de mots clés non html est décrite ici, celle de l'ajout de mots html est de la même forme, sauf qu'elle ne contient pas la ligne de remplacement de `QString` par un autre `QString` contenant les bornes). Après que chaque mot clé ait été "borné", ils sont insérés dans leurs listes respectives (`QStringList`). Enfin, une boucle parcourt ces mêmes listes et ajoute les mots clés dans le vecteur de coloration concerné puis on leur applique un format.

```
void Coloration :: ajouteMotCleFichier(const QString &chemin ,
                                       QStringList &liste ,
                                       QTextCharFormat format)
{
    QFile fichier(chemin);
    if( !fichier.exists() )1.00
    {
```

```

        cout<<"le_fichier_n'existe_pas."<<endl;
    }
    if( !fichier.open(QIODevice::ReadOnly) )
    {
        cout<<"impossible_d'ouvrir_ce_fichier."<<endl;
    }
    // ici on enleve les caracteres de fin de ligne du fichier texte et on
    // stocke sous forme de QString la chaine de caractere ainsi lue dans
    // le fichier texte.
    QString fic= fichier.readAll().trimmed();

    // creation d une liste temporaire contenant tous les mots du fichier texte
    QStringList list = fic.split("|");
    QString ajout = "\\b"; // permettra de borner les mots cles
    foreach(QString s, list) // pour chaque mot dans la liste temporaire
    {
        s.replace(s,ajout+s+ajout); // on rajoute les bornes au mot
        // on rajoute ces mots modifies dans la liste passee en parametre
        liste.push_back(s);
    }

    // permet de donner les regles de coloration a la liste
    if(renvoieListe(liste)=="php")
    {
        // ici chaque mot de la liste est transforme en QRegExp,
        // on les ajoute aux vecteur de regle de coloration,
        // on leur ajoute un format.
        foreach (QString expression_reg, liste)
        {
            rule_php.expression_reg = QRegExp(expression_reg);
            rule_php.format = format;
            regles_coloration_php.append(rule_php);
        }
    }
    else if(renvoieListe(liste)=="javascript")
    {
        foreach (QString expression_reg, liste)
        {
            rule_javascript.expression_reg = QRegExp(expression_reg);
            rule_javascript.format = format;
            regles_coloration_javascript.append(rule_javascript);
        }
    }
    else if(renvoieListe(liste)=="css")
    {
        foreach (QString expression_reg, liste)
        {
            rule_css.expression_reg = QRegExp(expression_reg);
            rule_css.format = format;
            regles_coloration_css.append(rule_css);
        }
    }
    else if(renvoieListe(liste)=="attributs_html")
    {

```

```

        foreach (QString expression_reg, liste)
        {
            rule_html.expression_reg = QRegExp(expression_reg);
            rule_html.format = format;
            regles_coloration_html.append(rule_html);
        }
    }
else
{
    foreach (QString expression_reg, liste)
    {
        rule.expression_reg = QRegExp(expression_reg);
        rule.format = format;
        regles_coloration.append(rule);
    }
}
fichier.close();
}

```

Après avoir ainsi instancié toutes les expressions régulières que souhaitons être colorées, il nous a fallu réimplémenté la fonction `highlightBlock` de la classe `QSyntaxHighlighter`.

### 8.2.2 Coloration : `:highlightBlock(const QString & text)`

**Données :** Cette fonction prend en paramètre le texte que l'utilisateur est en train de taper.

**Résultat :** Cette fonction est appelée automatiquement, dès lors qu'une touche du clavier est pressée. C'est dans cette fonction que le type de langage sera déterminé, et ce grâce à l'étude des caractères contenus dans l'éditeur, avec laquelle nous saurons si nous sommes en php ou en javascript par exemple. Effectivement, avec l'aide d'une boucle qui parcourt la ligne courante, on étudie chaque caractères tapés, lorsqu'un début de bloc est découvert, on passe de l'état par défaut à l'état concerné. Ainsi, lorsque l'on est en état php, seuls les mots du langage php sont colorés, il est alors impossible de colorer des mots html ou javascript (cette fonctionnalité sera possible grâce aux options de l'éditeur), les mots clés reconnus peuvent être colorés sur plusieurs lignes tant que le bloc n'est pas fermé. Après reconnaissance d'une ouverture de bloc, on assiste à un changement d'état, la coloration des mots clés peut alors commencer.

Etant donné la taille importante de la fonction `highlightBlock`, nous ne la mettrons pas en entière ici. Cependant, afin d'en expliquer son fonctionnement, nous présentons ici, une fonction simplifiée permettant de colorer du code php et du code javascript :

```

void Coloration::highlightBlock(const QString &text)
{
    // la variable etat est un entier qui recupere l'etat du bloc precedent
    etat = previousBlockState();

    int debut = 0; // debut de la ligne (position du curseur)

    // i est la position du curseur, il augmente au fur et a mesure que
    // l'utilisateur tape une lettre sur le clavier
    // dans cette boucle, on teste d'abord si on est dans un etat particulier,
    // sinon on etudie les caracteres tapes
    // par l'utilisateur, jusqu'a trouve l'ouverture d'un bloc.
    for (int i = 0; i < text.length(); ++i)
    {
        if (etat == php) // si l'etat du bloc est en php
        {

```

```

        // en "coupant" la chaine de caractere 2 caracteres par 2 caracteres ,
        // si on trouve ?> alors le bloc php est ferme
        if (text.mid(i, 2) == "\\?>")
        {
            etat=normal; // on ferme le bloc php et on passe en etat normal.
        }
    }
    else if (etat == javascript)
    {
        // fin_javascript est une QRegExp definie dans le constructeur de
        // la classe coloration : QRegExp fin_javascript=QRegExp("</SCRIPT>");
        if (fin_javascript.exactMatch(text.mid(i, 9)))
        {
            etat = normal; // la fin du bloc javascript etant trouvee ,
                // on retourne dans l'etat normal
        }
    }
    // si on est ni en javascript, ni en php, on continue de parcourir
    // les lignes tant qu'on a pas trouve de caracteres correspondant
    // au debut d'un bloc php ou javascript.
    else
    {
        // l'utilisateur vient de taper : <?php
        if (debut_php.exactMatch(text.mid(i,5)))
        {
            // la variable debut prend alors la valeur de i,
            // c'est a dire la position courante
            debut = i;
            // donne un etat a un QTextBlock* bloc_php(pour la frame aide)
            bloc_php->setUserState(php);
            // le debut de bloc etant reconnu comme etant du php,
            // on met dans etat la valeur de php
            etat=php;
        }
        // l'utilisateur vient de taper le debut d'un bloc javascript,
        // on realise les memes operation que pour le php
        else if (debut_javascript.exactMatch(text.mid(i, 28)))
        {
            debut = i;
            bloc_javascript->setUserState(javascript);
            etat = javascript;
        }
    }
} // fin de la boucle de definition des etats
// si l'etat est definit comme etant php, on lance la
// boucle de reconnaissance de mots cle (expliquee plus bas)
if (etat == php)
{
    foreach (ReglesColoration rule, regles_coloration_php)
    {
        QRegExp expression(rule.expression_reg);
        int index = text.indexOf(expression);
        while (index >= 0)

```



```

        {
            int length = expression.matchedLength();
            setFormat(index, length, rule.format);
            index = text.indexOf(expression, index + length);
        }
    }
}
if (etat == javascript)
{
    foreach (ReglesColoration rule, regles_coloration_javascript)
    {
        QRegExp expression(rule.expression_reg);
        int index = text.indexOf(expression);
        while (index >= 0)
        {
            int length = expression.matchedLength();
            setFormat(index, length, rule.format);
            index = text.indexOf(expression, index + length);
        }
    }
}
// on donne au bloc courant, la ou est le curseur la valeur de l'etat courant.
setCurrentBlockState(etat);
}

```

La coloration du langage html est cependant différente, en effet, les balises html se colorent sur une seule ligne et non dans un bloc de plusieurs lignes. C'est pour cela qu'il suffit seulement d'appeler la boucle de reconnaissance des mots clés html pour colorer ceux-ci.

### 8.2.3 Boucle de reconnaissance des mots clé en fonction du langage

**Données** : Cette boucle est située dans la fonction highlightBlock, elle est appelée lorsqu'on se situe dans un état, avec le vecteur de règles correspondant. Par exemple si l'état est Php, la boucle va chercher si le texte tapé par l'utilisateur correspond à une expression régulière dans le vecteur : reglescolorationphp.

**Résultat** : Lorsque le mot que l'utilisateur a entré est identifié par la boucle, il apparaît alors coloré dans la fenetre d'édition.

```

// pour chaque structure regle contenue dans le vecteur de regles_coloration_php
foreach (ReglesColoration regle, regles_coloration_php)
{
    //pour chaque structure, on recupere l'expression reguliere
    QRegExp expression(regle.expression_reg);

    // renvoie l'index de la premiere occurence correspondant a
    // l'expression reguliere dans la ligne courante de l'editeur.
    // si l'expression ne correspond pas au text contenu dans l'editeur,
    // index prends la valeur -1. si le mot correspond
    // index = position du curseur dans l'editeur en entier
    int index = text.indexOf(expression);

    while (index >= 0)    // un mot cle a ete tape
    {
        // on recupere le nombre de lettres du mot cle
        int length = expression.matchedLength();
    }
}

```

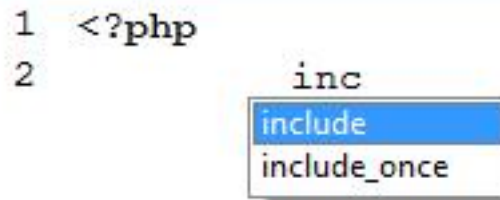
```
// on lui applique le format correspondant a son expression
// reguliere associee
setFormat(index , length , regle.format);

// on change l'index afin que si on tape plusieurs fois
// le mot sur la meme ligne , toutes les occurences seront colorees.
index = text.indexOf(expression , index + length);
}
}
```

## 9 Développement du Completer

### 9.1 Introduction

La librairie QT contenait déjà une classe nommé QCompleter. En la rattachant à un QTextEdit, cette dernière permettait d'y lier une liste de mots. Le premier problème rencontré fut la trop grande limitation de l'utilisation de fichier texte. Cette méthode nécessitait la présence d'une multitude de fichier contenant des listes de mots classés selon leur langage. Et à cela s'ajoutait le fichier gérant les mots ajoutés. Par la suite, il s'avéra aussi que le compléteur ne pouvait contenir qu'une seule liste de mot, passée en paramètre sous forme d'une QStringList. Il était donc impossible de charger la liste en une seule fois. De plus, il fallait traiter la gestion des événements des touches, afin de pouvoir gérer l'activation ou non du compléteur selon la touche pressée, ainsi que l'ajout automatique de mots inexistant dans la liste et d'un raccourci activant la complétion. Lors de la frappe du raccourci, le problème qu'il a fallut traiter, fût de ne pas faire afficher son caractère dans le texte (exemple : si le raccourci est ctrl+espace, il ne fallait pas afficher l'espace).



### 9.2 Fonctions

#### 9.2.1 Completer : :activeComplete

**Données** : Si la touche pressée doit activer la complétion, cette fonction affiche la liste de mots possibles commençant par la chaîne déjà écrite.

- Paramètre : l'évènement du clavier.
- Attributs : le préfixe du mot saisi et la liste de caractères ne pouvant pas faire partis d'un mot.

**Résultat** : La fonction récupère l'évènement du clavier (la touche pressée) et le préfixe du mot actuellement saisi. Ensuite elle vérifie, dans le cas où il s'agit de la fin d'un mot, si elle doit l'ajouter en tant que variable ou nom de fonction. Enfin, si la touche n'est pas un raccourci et si le préfixe est supérieur à 3 caractères, la fonction dessine un carré contenant une liste de mots possibles.

```
void Completer::activeComplete(QKeyEvent *e)
{
    bool est_touche_de_raccourci = ((e->modifiers() & Qt::ControlModifier)
                                     && e->key() == Qt::Key_Space);
    QString completion_prefixe = this->textUnderCursor();
    static QString fin_de_mot(" ~!@#%&*()_+{|: \ "<>? ,./; '[]\ \ - = ");
    if(((e->key()==Qt::Key_Space)&&!est_touche_de_raccourci) ||
        ((e->key()==Qt::Key_Enter) &&
         !(fin_de_mot.contains(e->text().right(1)))) ||
        fin_de_mot.contains(e->text().right(1))&&!Qt::ControlModifier)
    {
        if(!(this->getWords().contains(ancien_mot))&&
            ancien_mot.count()>3)
        {
            if((parent->getColoration()->getEtat()==4)&&
                (ancien_mot.left(1)=="$"))
            {
```

```

        this->ajouteVariable(completion_prefixe);
    }
    else if(verifVariableOuFonctionJS())
    {
        this->ajouteVariable(completion_prefixe);
    }
}
}
const bool est_ctrl_ou_shift = e->modifiers() &
    (Qt::ControlModifier | Qt::ShiftModifier);
if (!this || (est_ctrl_ou_shift && e->text().isEmpty()))
{
    return;
}
bool hasModifier = (e->modifiers() != Qt::NoModifier) &&
    !est_ctrl_ou_shift;
if (!est_touche_de_raccourci && (hasModifier || e->text().isEmpty() ||
    completion_prefixe.length() < 3 ||
    fin_de_mot.contains(e->text().right(1))))
{
    this->popup()->hide();
    return;
}
if (completion_prefixe != this->completionPrefix())
{
    this->setCompletionPrefix(completion_prefixe);
    this->popup()->setCurrentIndex(this->completionModel()->
        index(0, 0));
}
QRect carre_de_completion = QRect(parent->cursorRect());
carre_de_completion.setWidth(this->popup()->sizeHintForColumn(0) +
    this->popup()->verticalScrollBar()->sizeHint().width());
this->complete(carre_de_completion);
}

```

### 9.2.2 Completer : :Completer

**Données** : Initialise le compléteur.

- Paramètre : Parent est le QTextEdit lié au compléteur.
- Attributs : Chemin\_fichier\_variables contient le chemin du fichier stockant les variables.

**Résultat** : Initialise le compléteur en lui donnant une liste de mots, connecte aussi son activation à l'insertion du mot et initialise le chemin du fichier de variables.

```

Completer::Completer(Editeur * parent) : QCompleter(parent)
{
    this->parent = parent;
    this->setModel(modelFromFile());
    this->setModelSorting(QCompleter::CaseInsensitivelySortedModel);
    this->setCaseSensitivity(Qt::CaseInsensitive);
    this->setWrapAround(false);
    this->setWidget(parent);
    this->setCompletionMode(QCompleter::PopupCompletion);
    this->setCaseSensitivity(Qt::CaseInsensitive);
    QObject::connect(this, SIGNAL(activated(const QString&)), this,

```

```

        SLOT(insertCompletion(const QString&));
    chemin_fichier_variables = QString(systeme_relation_fichier).
        append("variables.txt");
}

```

### 9.2.3 Completer : :insertCompletion

**Données** : Insère le mot sélectionné dans la zone de texte.

- Paramètre : completion est le mot choisi dans la liste de complétion.
- Attributs : tc stockera le curseur récupéré de la zone de texte pour pouvoir y écrire.

**Résultat** : La fonction met la fin du mot sélectionné dans la zone de texte en évitant d'écrire deux fois le début.

```

void Completer::insertCompletion(const QString& completion)
{
    if (this->widget() != parent)
        return;
    QTextCursor tc = parent->textCursor();
    int extra = completion.length() - this->completionPrefix().length();
    tc.movePosition(QTextCursor::Left);
    tc.movePosition(QTextCursor::EndOfWord);
    tc.insertText(completion.right(extra));
    parent->setTextCursor(tc);
}

```

### 9.2.4 Completer : :modelFromFile

**Données** : Charge la liste de mot du compléteur via la liste de fonction du langage courant ainsi que via le fichier de variables/fonctions

- Attributs : Words est la liste de mot qui sera retournée pour le compléteur.

**Résultat** : La fonction met dans words la liste de mot que le compléteur devra proposer et retourne un QStringListModel composé de cette liste. La liste sera fonction du langage courant.

```

QAbstractItemModel *Completer::modelFromFile()
{
    #ifndef QT_NO_CURSOR
        QApplication::setOverrideCursor(QCursor(Qt::WaitCursor));
    #endif
    if (!parent->getColoration()->getListe().isEmpty())
    {
        words.append(parent->getColoration()->getListe());
    }
    majListeVariables();
    words.sort();
    #ifndef QT_NO_CURSOR
        QApplication::restoreOverrideCursor();
    #endif
    return new QStringListModel(words, this);
}

```

## 10 Développement de Syntaxe

### 10.1 Introduction

Pour faire la vérification syntaxique, il a d'abord fallu choisir quelles étaient les vérifications les plus utiles à faire. Le choix s'est évidemment porté sur le parenthésage, et comment le rendre le plus lisible possible. La classe syntaxe se compose donc de deux grosses parties, la vérification du parenthésage et le surlignement d'un bloc parenthèse lorsqu'on positionne le curseur à côté.

Pour la vérification du parenthésage, nous avons utilisé, sur le modèle de l'automate à pile, une pile dans laquelle, lorsqu'on arrive sur une ouverture de bloc ( '[', '(', '"' ), on empile la fermeture de bloc correspondante ( ']', ')', '"' ). Puis, lorsqu'on arrive sur une fermeture de bloc si elle est identique à celle du début de pile on dépile sinon on marque une erreur.

Pour le surlignage des blocs, nous avons préféré créer une fonction la plus générale possible (qui peut être utilisée sur tous types de bloc). En effet, elle prend en paramètre deux expressions régulières, une pour reconnaître l'ouverture du bloc et l'autre la fermeture du bloc, deux entiers, un pour la longueur de l'expression d'ouverture et l'autre pour celle de fermeture ainsi qu'un entier pas, pour savoir si on cherche en amont ou en aval du curseur. Ainsi, en ne changeant que les paramètres on peut appeler la même fonction pour reconnaître par exemple un bloc `<?php?>` ou un bloc `.`. En revanche, même si elle fonctionne très bien, sur de gros fichiers la vérification du parenthésage ralentit le processus. Cela est dû au fait que la fonction de vérification est appelée à chaque modification du texte sur l'intégralité du texte. Pour résoudre ce problème, plusieurs solutions ont été envisagées, d'abord nous avons essayé de mettre un compteur pour n'appeler la vérification que tous les  $n$  fois mais ceci ne change pas la complexité du nombre d'appel donc cela a un effet très limité. Ensuite, nous avons envisagé de sauvegarder la position des blocs bien parenthésés grâce à une liste de curseur pour ne faire la vérification qu'au niveau du changement effectué, du bloc correct précédent au bloc correct suivant. Mais cette solution a échoué pour deux raisons liées au signal utilisé pour lancer la vérification. En effet, à chaque modification du texte le signal `contentsChange` nous transmet trois paramètres, la position du changement, le nombre de caractères enlevés et le nombre de caractères ajoutés. Mais ceux-ci ne sont pas corrects, et donc l'utilisation du paramètre `position` non valable donnait des résultats incohérents. De plus, lors des suppressions de blocs, il aurait fallu mettre à jour la liste des blocs corrects mais comme le paramètre des caractères enlevés n'est pas juste, il est impossible de différencier grâce à lui un ajout et une suppression et donc de réaliser cette solution.

Pour finir nous avons envisagé de créer grâce à des threads un processus fils qui gèrerait en parallèle du processus principal la vérification mais nous n'avions plus le temps nécessaire pour mettre en place cette solution.

```
6
7 int main(int argc, char *argv[])
8 {
9     QApplication a(argc, argv);
10    MainWindow w;
11    w.show();
12    Arbo * ar = new Arbo();
13    ar->listerRepALaQt();
14    return a.exec();
15 }
16
17 void mauvaisParenthésage();
```

Grise les blocs de parenthèse quand le curseur est positionné à côté.

Souligne les erreurs de parenthésage

## 10.2 Fonctions principales de Syntaxe

### 10.2.1 Sytaxe : :verification

Données :

- Paramètres : QString s : texte à vérifier  
int deb : indice de début à partir duquel vérifier  
int fin : indice de fin jusqu'auquel vérifier
- Attributs : QList<QTextCursor> listeSouligne : liste de curseur contenant en selection les zones de texte soulignées en erreur

Résultat : souligne en rouge ondulé les endroits du texte où le parenthésage est incorrect.

```
void Syntaxe::verification(QString s, int deb, int fin)
{
    QStack<QChar> attendu;
    QTextCursor curseur(editeur->document());

    //enlever tous les soulignements
    QTextCharFormat souligne;
    souligne.setFontUnderline(false);
    QTextCursor item;
    while(!listeSouligne.isEmpty())
    {
        item = listeSouligne.first();
        item.mergeCharFormat(souligne);
        listeSouligne.removeFirst();
    }

    curseur.setPosition(deb);
    for(int i = deb; i<fin; i++)
    {
        QChar c = s[i];

        if(c=='(')
        {
            attendu.push('(');
        }
        else if(c=='{')
        {
            attendu.push('{');
        }
        else if(c=='[')
        {
            attendu.push('[');
        }
        else if(c==')' || c=='}' || c==']')
        {
            if(attendu.isEmpty() || attendu.top() != c)
            {
                souligneErreur(curseur);
            }
            else
            {
                attendu.pop();
            }
        }
    }
}
```

```

    }
    curseur.setPosition(curseur.position()+1);
}
}

```

### 10.2.2 Syntaxe : :surligneBloc

#### Données :

- Paramètres : QTextCursor\* curseur : Copie du curseur courant de l'éditeur
- int pas : sens dans lequel incrémenter le curseur pour trouver l'extrémité du bloc,-1 si on est en fin de bloc, 1 si on est en début
- QRegExp debut : Expression régulière permettant d'identifier le début de bloc
- QRegExp fin : Expression régulière permettant d'identifier la fin de bloc
- int longueur\_deb : longueur de l'expression qui ouvre le bloc
- int longueur\_fin : longueur de l'expression qui ferme le bloc
- Attributs : QTextCursor\* surligne : curseur contenant en sélection la zone de texte grisée précédemment

**Résultat :** Grise le bloc à côté duquel se trouve le curseur.

```

void Syntaxe::surligneBloc(QTextCursor* curseur, int pas, QRegExp debut,
                          QRegExp fin, int longueur_deb, int longueur_fin)
{
    QString contenu = editeur->toPlainText();
    curseur->setPosition(editeur->textCursor().position());
    int ind = curseur->position();
    int bon = -1;
    bool pastrouve = true;
    while(ind <= contenu.length() && ind >= 0 && pastrouve)
    {
        if(fin.exactMatch(contenu.mid(ind, longueur_fin)))
        {
            if(bon == 0)
            {
                pastrouve = false;
                if(pas < 0)
                {
                    ind++;
                }
                else
                {
                    ind = ind + longueur_fin - pas;
                }
            }
            else
                bon--;
        }
        else if(debut.exactMatch(contenu.mid(ind, longueur_deb)))
        {
            bon++;
        }
        ind += pas;
    }
    if(!pastrouve)
    {

```



```
    curseur->setPosition(ind, QTextCursor::KeepAnchor);
    QTextCharFormat grise;
    grise.setBackground(Qt::gray);
    curseur->mergeCharFormat(grise);
    surligne = curseur;
}
}
```

# 11 Développement Indentation

## 11.1 Introduction

Cette classe nous permet d'indenter automatiquement le code rentré par l'utilisateur en cours de la frappe. L'indentation automatique fonctionne sur deux langages le Html et le Php. Après vérification si le texte se trouve entre deux balise Php, l'indentation s'effectuera sur les " ", sinon l'indentation sera basé sur les balises Html. Pour rajouter l'indentation, on utilise un expression régulière QRegExp qui stockera une tabulation et l'ajoutera en début de ligne et nous permet ainsi d'obtenir une indentation.

La particularité ici, c'est l'automatisation de l'indentation, pour cela on va utiliser un signal qui pour tout changement dans le contenu de l'éditeur, appelle le slot avec la fonction "textechange" et lance la vérification syntaxique du texte. SIGNAL(contentsChange(int, int,int)), this, SLOT(texteChange(int, int, int));

## 11.2 Fonctions

### 11.2.1 void Indent : :texteChange(int pos, int ajout, int enleve)

**Données** : la variable pos, correspond à la position du curseur , les deux dernières variables vérifie l'ajout ou le retrait de caractère dans l'éditeur.

**Résultat** : La fonction verifie donc s'il y a eu des changements dans l'éditeur c'est à dire, l'ajout ou le retrait de caractères, si c'est le cas elle appelle la fonction indenter

```
void Indent::texteChange(int pos, int ajout, int enleve)
{
    if(ajout != 0 || enleve != 0) //verifie si le texte a changer
    {
        QString s = editeur->toPlainText();
        //verifie que le contenu de s recuperer depuis l'editeur n'est pas vide
        if(!s.isEmpty())
        {
            indenter(s); //execute la fonction indenter.
        }
    }
}
```

Pour différencier les langages Php et Html, on utilise une variable bool par défaut cette variable qui a pour valeur : "false", si l'utilisateur ouvre une balise php (<?php) alors la variable sera modifié en "true" et logiquement si l'utilisateur ferme une balise php(??) alors la variable redevient false. On met en place également une variable indent qui comptabilisera le nombre d'indentation à effectuer. En effet les tabulations doivent augmenter en même temps que les ouvertures des balises ou accolades. Donc on va compter le nombre d'ouverture de balises/accolades ainsi que le nombre de balises/accolades fermées , une fois ceci effectuer la variable indent reçoit le nombre d'ouverture – le nombre de fermetures. Comme notre indentation repose sur une tabulation à travers un QregExp , on vérifie que l'utilisateur n'a pas lui même effectué une tabulation ou espace en début de lignes. Si c'est le cas alors on les supprime avant de placer l'indentation.

### 11.2.2 void Indent : :indenter(QString s)

**Données** : QString s reçoit directement le contenu du QTextEdit(correspondant à l'éditeur).

**Résultat** : On utilise une boucle for avec la variable d'indentation comme arrêt qui va appliquer l'indentation sur les strings Dans le cas du Html, on prends en compte le cas spéciale des balises qui sont fermés directement. Ex : <br/> qui ne nécessite pas d'indentation.

```
{
    int indent = 0; //variable d'indentation
    bool php = false; //variable Php
```

```

//position actuelle du curseur
QTextCursor curseur(editeur->document());
//decoupage de la string en plusieurs a chaque entree
QStringList lignes = s.split("\n");
//pour chaque nouvelle strings
for (int i=0; lignes.size();i++) {
    //suppression d'eventuel espaces tabulation ou autre
    lignes.replaceInStrings(QRegExp("^\\t"), "");
    lignes.replaceInStrings(QRegExp("^_"), "");
    int p = lignes.at(i).count("<?php"); //detection de balise php
    int fp = lignes.at(i).count(">"); //detection de fin de php
    if (p>0)php = true;
    if (fp>0)php = false;
    //on rajoute indent tabulation en debut de ligne
    for(int j= 0; j<= indent; j++)lignes.replaceInStrings(QRegExp("^"), "\\t");
    //declaration de variable de comptage Php et Html
    int acolO=0,acolF=0,ouver=0,ferme=0,vide=0;
    if (php){
        //On compte les '{' dans le cas php
        acolO = lignes.at(i).count("{");
        acolF = lignes.at(i).count("}");
    }
    else {
        //On compte les balises dans le cas Html.
        ouver = lignes.at(i).count("<");
        ferme = lignes.at(i).count("</");
        //Cas speciale des balises fermes directement.
        vide = lignes.at(i).count("//>");
    }
    indent = indent + (ouver-(vide+ferme)) + (acolO - acolF);
}
s = lignes.join("\n");// on rassemble les strings par des entrees
}

```

## 12 Développement de Projets

### 12.1 Introduction

Cet ensemble de classes se charge d'afficher l'arborescence de l'espace de travail choisi par l'utilisateur et de faire le lien entre les double-clics sur les items de cette arborescence et l'affichage des fichiers correspondants dans l'éditeur.

Cette partie est composée de trois classes :

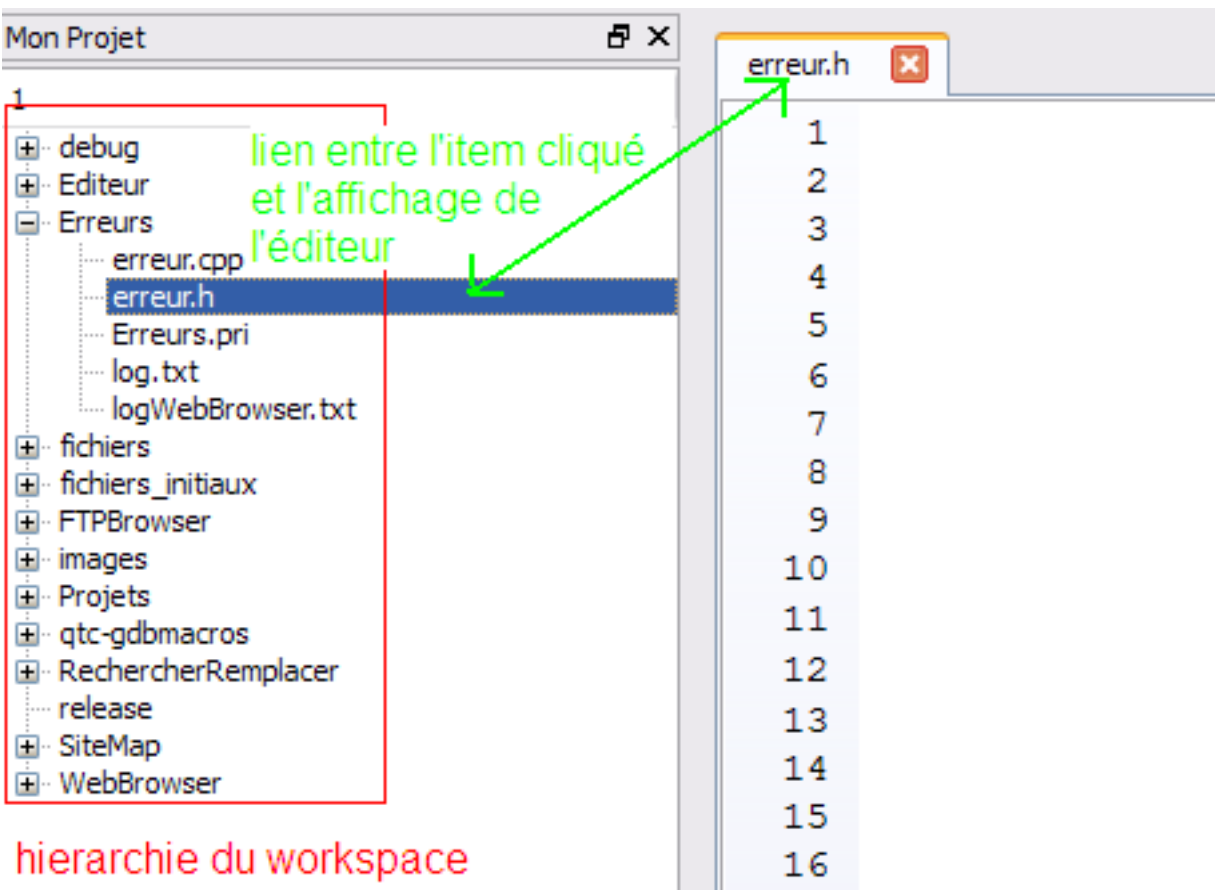
Hierarchie qui correspond à l'espace de travail et représente la racine de l'arbre, elle hérite de `QTreeWidget` qui gère l'arborescence.

ItemHierarchie qui correspond aux fichiers et représente les feuilles de l'arbre, elle hérite de `QTreeWidgetItem` qui implémente les fils de `QTreeWidget` dans l'arborescence.

Projet, héritant de `itemHierarchie`, correspond aux dossiers et représente les noeuds de l'arbre. Dans l'arborescence, a pour fils des `itemHierarchie` ou des projets.

Très simplement, la classe hiérarchie liste son contenu et crée pour chaque repertoire qu'elle contient une instance de la classe projet qu'elle charge de se lister. Ainsi, chaque projet liste son contenu et crée pour chaque fichier qu'il contient une instance de la classe `ItemHierarchie` et pour chaque dossier une instance de la classe projet qu'il charge de se lister.

Il n'a pas été fait de tri sur les fichiers à afficher en fonction de l'extension, en effet les types de fichiers que peut ouvrir notre éditeur n'étant pas définie, nous avons préféré ne pas mettre de restrictions sur les fichiers à afficher dans l'arborescence.



## 12.2 Fonctions

### 12.2.1 Hierarchie : :parcours

**Données :**

- Paramètres : aucun
- Attributs : QDir\* espaceDeTravail : contient les informations de l'espace de travail que l'utilisateur a choisi

**Résultat :** Parcourt l'espace de travail et pour chaque dossier qu'il contient crée un projet qu'il fait se lister.

```
void Hierarchie::parcours()
{
    QFileInfoList contenu = espaceDeTravail->entryInfoList();
    Projet* projet;
    for(int i=2; i<contenu.length(); i++)//debut a 2 pour ne pas regarder . et ..
    {
        if(contenu.at(i).isDir())
        {
            QFileInfo * infoProjet = new QFileInfo(contenu.at(i));
            projet = new Projet(infoProjet, infoProjet->baseName());
            this->ajouteProjet(projet);
        }
    }
    this->setColumnCount(1);
}
```

### 12.2.2 Projet : :parcours

**Données :**

- Paramètres : QDir\* dossier : contient les informations du dossier à lister.
- Attributs : aucun

**Résultat :** Parcourt le dossier, pour chaque dossier qu'il contient crée un projet qu'il fait se lister et pour chaque fichier crée un itemHierarchie.

```
void Projet::parcours(QDir* dossier)
{
    //on recupere le contenu de notre dossier
    QFileInfoList contenu = dossier->entryInfoList();
    for(int i=2; i<contenu.length(); i++)//on le parcourt
    {
        if(contenu.at(i).isDir() && !contenu.at(i).isSymLink())
            //si c'est un dossier on ajoute un niveau a l'arborescence
            // et on le parcourt
        {
            QDir* sousDossier = new QDir(contenu.at(i).absoluteFilePath());
            QFileInfo* sousInfo = new QFileInfo(contenu.at(i));
            Projet* sousItem = new Projet(sousInfo, sousInfo->baseName());
            this->addChild(sousItem);
            sousItem->parcours(sousDossier);
        }
        else if(contenu.at(i).isFile())
            //si c'est un fichier on l'ajoute a l'arborescence
        {
            QFileInfo* infoFichier = new QFileInfo(contenu.at(i));
            ajouteFichier(infoFichier, infoFichier->fileName());
        }
    }
}
```

```

    }
}

```

### 12.2.3 Hierarchie : :itemDoubleClic

**Données :**

- Paramètres : QTreeWidgetItem \* it : l'item qui a été double-cliqué.
- Attributs : QWidget\* parent : la fenêtre contenant la hierarchie.

**Résultat :** ouvre dans l'éditeur le fichier qui correspond à l'item double-cliqué.

```

void Hierarchie::doucleClicItem ( QTreeWidgetItem * it )
{
    ItemHierarchie * item = dynamic_cast<ItemHierarchie*>(it);
    MainWindow * fenetre = static_cast<MainWindow*>(this->parent);
    if(item)
    {
        if(fenetre)
        {
            fenetre->ouvreFichier(item->getInfoFichier(), false);
        }
    }
}

```

## 13 Développement du plan du site

### 13.1 Introduction

Ici on va répertorier l'architecture du site afin de permettre à l'internaute d'accéder rapidement aux pages qui l'intéressent. On va également permettre aux robots d'indexation de référencer l'ensemble des documents disponibles sur le site. Tout ceci ne s'est pas fait sans heurts et des problèmes ont été rencontrés. Parfois on a eu à faire des changements d'implémentations. L'utilisation de QT et le parcours du répertoire racine et de ses sous-répertoires ("www" en général), en listant uniquement les fichiers web, a fait apparaître des bugs propres à QT. La maîtrise de ce langage n'étant pas évidente, il a fallu d'abord comprendre son concept.

### 13.2 Fonctions

#### 13.2.1 Fichier : :ajouter(string corps)

**Données :**

- Paramètres : ofstream Ecriture;

**Résultat :** écrit le corps du fichier siteMap.html

```
if(Ecriture)
{
    //Ecriture : fichier ouvert en ecriture
    Ecriture << corps;
}
```

#### 13.2.2 Fichier : :prefixe(string extension)

**Données :**

- Paramètres : string mon\_fichier;
- Attributs : string pref;

**Résultat :** on lui donne l'extension qu'on veut enlever et elle renvoie le nom d'un fichier sans son extension pendant le listage.

```
cout<<" taille_mon_fichier_:"<<mon_fichier.length()<<endl;
cout<<" taille_extension_:"<<extension.length()<<endl;
if( mon_fichier.substr(mon_fichier.length() - extension.length() ,
    extension.length() ) == extension )
{
    pref = mon_fichier.substr(0, mon_fichier.length() -
        extension.length() );
    cout<< pref<< endl;
}
return pref;
```

#### 13.2.3 Arbo : :listerRepertoire(char\* nom)

**Données :**

- Paramètres : DIR\* repertoire, struct dirent\* rep\_pointe;

**Résultat :** lister le contenu du répertoire nom

```
repertoire = opendir(nom);
if( repertoire == NULL )
{
    cout<<" erreur "<<endl;
```

```
}  
else  
{  
    rep_pointe = readdir(repertoire);  
    while( rep_pointe != NULL)  
    {  
        cout<< rep_pointe->d_name;  
    }  
    closedir(repertoire);  
}
```



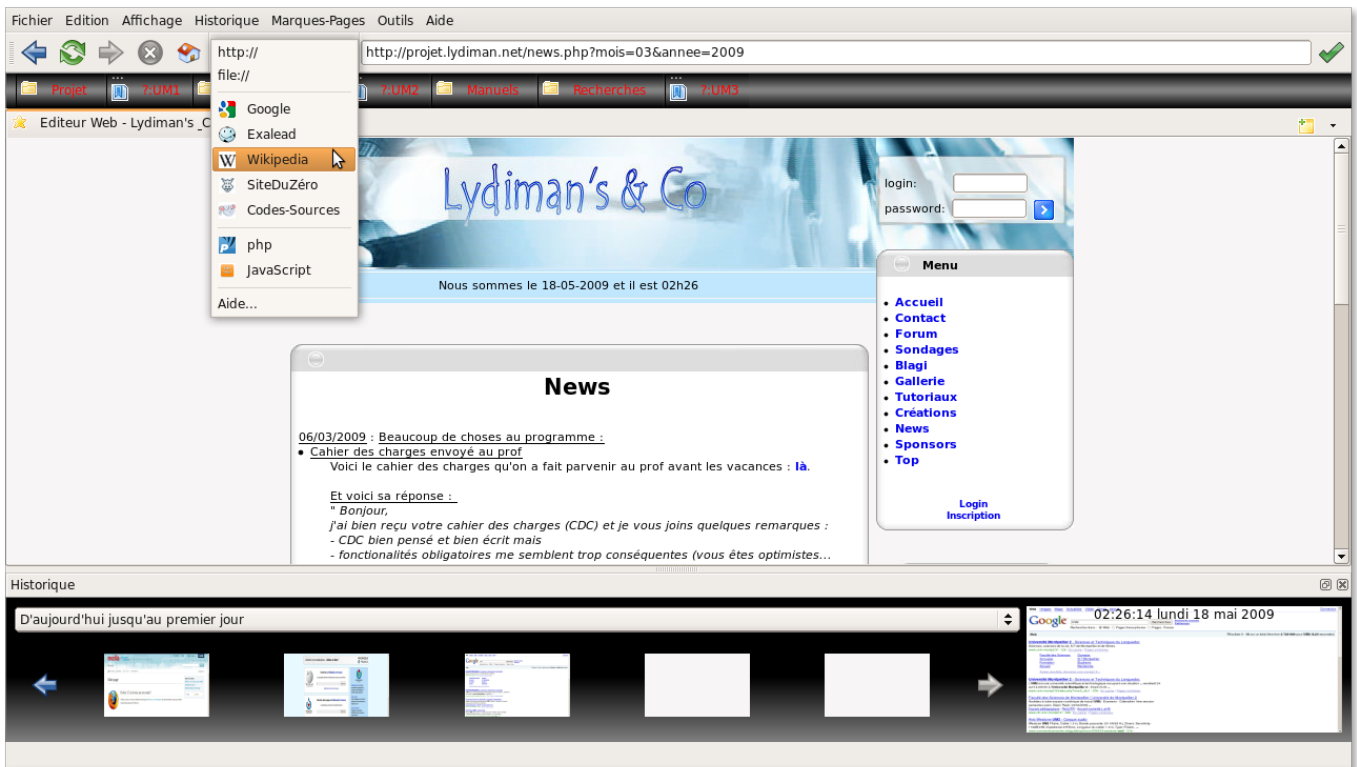
# 14 Développement du WebBrowser

## 14.1 Introduction

Le module WebBrowser est un explorateur internet permettant à l'utilisateur de pouvoir effectuer facilement des recherches sur des sites pré-sélectionnés par nos soins en rapport avec le monde du web. Ce module a été développé de façon autonome au projet, il peut donc tout à fait être utilisé en dehors du projet. Mais il peut charger des pages en effectuant des recherches sur les sites spécialisés dans la documentation, ou la création de scripts internet, ce qui l'intègre parfaitement dans l'éditeur web.

Un explorateur web se divise en deux parties : un moteur de rendu HTML et une interface. Qt nous offre la possibilité d'utiliser WebKit, le moteur de rendu de Safari (édité par Apple) ou encore Chrome (édité par Google). Il passe à 100% les tests du W3C Validator et peut donc être perçu comme l'un des meilleurs moteurs de rendu HTML du moment. Cependant notre navigateur web ne permet pas d'obtenir de si bons résultats. Le moteur interne de WebKit et surtout l'intégration du moteur par Qt n'étant pas terminés beaucoup de fonctionnalités ne sont pas accessibles. De ce fait l'utilisation complète du javascript n'est pas opérationnelle. Il en est de même pour l'utilisation des cookies, et la gestion des téléchargements. Les requêtes envoyées et reçues sont masquées à la fois par WebKit puis par Qt ce qui rend leur traitement très obscure car trop difficile à décrypter. Mais nous espérons que nous aurons trouvé la solution d'ici peu. Plusieurs tests ont été réalisés afin de comprendre le fonctionnement des requêtes internes mais sans résultats significatifs.

Pour le côté interface du navigateur internet, Qt dispose de tous les composants nécessaires à sa complète réalisation, c'est donc pour cela que la première partie est moins développée dans ce projet. Nous avons donc réalisé la plupart des modules d'un explorateur web traditionnel tel que l'historique ou encore les marques-pages sans oublier les autres fonctions basiques telles que l'ouverture de plusieurs pages (par onglets), copier, coller du texte ou encore afficher une image. Toutes les fonctionnalités implémentées peuvent être paramétrées par l'utilisateur afin de rendre son utilisation la plus conviviale possible.



## 14.2 Choix d'implémentation

Pour ce module l'implémentation n'a pas eu une place très importante dans l'analyse car le module WebKit nous invite, en réimplémentant ses classes, à suivre une hiérarchie bien particulière. En effet un QWebView, permet d'afficher une page internet, elle contient une QWebPage qui permet à une page internet de communiquer avec l'explorateur internet afin d'afficher une fenêtre avec un alert() javascript par exemple. Enfin cette QWebPage contient des QWebFrame affichant les différentes frames du site internet.

La hiérarchie est donc évidente à comprendre : un QMainWindow pour la fenêtre principale du WebBrowser, qui contient un QTabWidget représentant les différentes pages en cours de visualisation, ces pages héritent de QWebView afin de voir la page en elle même, bien sûr une page contient une frame permettant le relais entre le navigateur et le système.

Concernant l'historique et les marques-pages nous avons essayé d'implémenter notre code en suivant le plus possible le pattern MVC modèle-vue-controlleur permettant ainsi une grande réutilisation du code, notamment pour le côté graphique facilement modifiable. Ainsi chaque sous-module est formé d'une classe permettant le contrôle de la classe et d'au moins une autre permettant l'affichage.

## 14.3 Fonctions Principales

### 14.3.1 WebBrowser : :chargerPage

**Données :**

- Paramètres : QUrl u l'url du site à charger, int mode mode=0 dans l'onglet courant ou mode=1 dans un nouvel onglet
- Attributs : QLineEdit \* edition\_url permet à l'utilisateur d'éditer l'url, QLineEdit \* multi\_pages représente les onglets contenant les pages en cours d'utilisation

**Résultat :** Cette méthode permet de charger et d'afficher une page internet à partir de son url. Après avoir vérifié le mode elle va soit modifier la zone d'édition pour l'utilisateur et charger le lien inscrit, soit ajouter un nouvel onglet avant de modifier la zone d'édition pour l'utilisateur et charger le lien inscrit.

```
void WebBrowser::chargerPage(QUrl u, int mode)
{
    if(mode == 0)
    {
        edition_url->setText(u.toString());
        actionCharger();
    }
    else
    {
        multi_pages->charger(u);
        edition_url->setText(u.toString());
        actionCharger();
    }
}
```

### 14.3.2 WebBrowser : :actionCharger

**Données :**

- Attributs : QLineEdit \* edition\_url permet à l'utilisateur d'éditer l'url, QLineEdit \* multi\_pages représente les onglets contenant les pages en cours d'utilisation

**Résultat :** Cette méthode permet de charger et d'afficher une page internet à partir de son url obtenue par la zone d'édition de l'utilisateur. Elle va rechercher dans la chaîne de caractères entrée par l'utilisateur pour savoir s'il veut se rendre sur un site ou s'il veut effectuer une recherche. S'il trouve une recherche alors il appelle la fonction load() de la page courante (héritée de QWebView qui s'occupe de récupérer la page) avec l'url correspondant au moteur de recherche du site en question, menant directement aux réponses trouvées.

```

void WebBrowser::actionCharger ()
{
    QString s(edition_url->text ());
    if(s.contains("Google_:", Qt:: CaseInsensitive) ||
        s.contains("Google:", Qt:: CaseInsensitive) ||
        s.contains("?:_:")) || s.contains("?:")
    {
        s = s.section(":", 1, 1);
        multi_pages->getPageCourante()->
            load(QUrl(s.prepend("http://google.fr/search?q=")));
    }
    else if(s.contains("Exalead_:", Qt:: CaseInsensitive) ||
        s.contains("Exalead:", Qt:: CaseInsensitive) ||
        s.contains("Exa_:", Qt:: CaseInsensitive) ||
        s.contains("Exa:", Qt:: CaseInsensitive) ||
        s.contains("?:_:")) || s.contains("?:")
    {
        s = s.section(":", 1, 1);
        multi_pages->getPageCourante()->
            load(QUrl(s.prepend("http://www.exalead.fr/search/results?q=")));
    }
    ...

    else if(s.contains("http://") ||
        s.contains("https://") ||
        s.contains("file://"))
    {
        multi_pages->getPageCourante()->load(QUrl(s));
    }
    else if(s.contains("www.", Qt:: CaseInsensitive))
    {
        multi_pages->getPageCourante()->
            load(QUrl(s.prepend("http://")));
    }
    else
    {
        multi_pages->getPageCourante()->
            load(QUrl(s.prepend("http://google.fr/search?q=")));
    }
}

```

### 14.3.3 Historique : :Restaurer

#### Données :

- Attributs : QList<struct\_historique> contenant les informations de l'historique

**Résultat :** Cette méthode permet de charger l'historique de l'utilisateur. Elle utilise le système de QSettings de Qt. Ce système permet de stocker des variables de types complexes (il suffit qu'il soit accepté par QVariant) très facilement et s'adaptant au mieux suivant la plateforme de l'utilisateur, sous Windows il le stocke dans la base de registres, sous les autres systèmes dans un fichier situé dans le répertoire /home/pseudo/.config. Ainsi on peut le restaurer très facilement tout en étant dans des classes très différentes.

Nous remplissons donc la QList<struct\_historique> contenant les informations de l'historique en les récupérant par le système des QSettings.

```

void Historique::restaurer ()
{
    QSettings settings ("MW", "WebBrowser");

    int size = settings.beginReadArray ("Historique");

    for (int i=0; i<size; i++)
    {
        settings.setArrayIndex (i);
        struct _historique temp;
        temp.url = settings.value (QString ("url")).toString ();
        temp.titre = settings.value (QString ("titre")).toString ();
        temp.date = settings.value (QString ("date")).toList ();
        url_s->append (temp);
    }
    settings.endArray ();
}

```

#### 14.3.4 Historique : :Enregistrer

**Résultat** : Cette méthode permet d'enregistrer l'historique de l'utilisateur. Elle utilise le système de QSettings de Qt tout comme Recharger(). Vous remarquerez que nous utilisons la même déclaration pour le QSettings, afin de regrouper les informations par logiciel, module, fichier.

```

void Historique::enregistrer ()
{
    QSettings settings ("MW", "WebBrowser");

    settings.beginWriteArray ("Historique");
    for (int i=0; i<url_s->count (); i++)
    {
        if (!url_s->value (i).url.isEmpty ())
        {
            settings.setArrayIndex (i);
            settings.setValue (QString ("url"), url_s->value (i).url);
            settings.setValue (QString ("titre"), url_s->value (i).titre);
            settings.setValue (QString ("date"), url_s->value (i).date);
        }
    }
    settings.endArray ();
    settings.sync ();
}

```

#### 14.3.5 MarquePage : :StartDrag

**Données** :

- Paramètres : QMouseEvent event l'événement permettant de récupérer la position de la souris pour le déplacement.
- Attributs : bool estDeplace, estClicke ces deux booléens permettent de savoir si la source du déplacement (un marque-page) est en mouvement.

**Résultat** : Lance un drag permettant un drop avec un marque-page. Nous remarquerons que Qt utilise un QByteArray (un tableau de bits) pour stocker les données du drag, puis avec une simple image de l'élément sélectionné pour créer un drag.

```

void MarquePage::startDrag(QMouseEvent * event)
{
    QByteArray itemData;
    QDataStream dataStream(&itemData, QIODevice::WriteOnly);

    dataStream << this->text() << this->accessibleDescription()
                << place_dans_liste;

    QMimeData * mimeType = new QMimeData;
    mimeType->setData("text/html", itemData);

    QDrag *drag = new QDrag(this);
    drag->setMimeData(mimeType);

    QPixmap pixmap(this->size());
    this->render(&pixmap,
              QPoint(0, 0),
              QRegion(0, 0,
                     this->width(),
                     this->height()),
              RenderFlags());

    drag->setPixmap(pixmap);
    drag->setHotSpot(event->pos());

    if (drag->exec(Qt::CopyAction, Qt::IgnoreAction) == Qt::CopyAction)
    {
        estDeplace = false;
        estClicke = false;
    }
    else
    {
        parent->supprimerFleche();
        estDeplace = false;
        estClicke = false;
    }
}

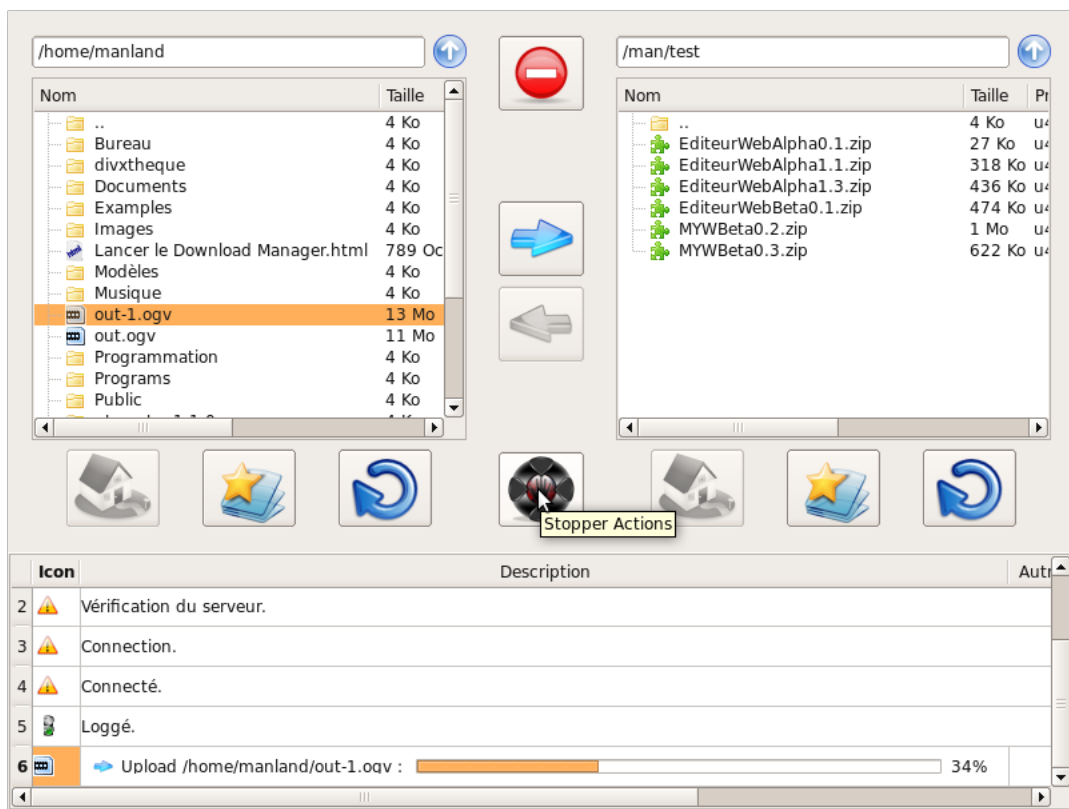
```

## 15 Développement du FTPBrowser

### 15.1 Introduction

Le FTPBrowser est un module faisant parti des fonctionnalités optionnelles, il a donc été développé en dernier. De ce fait son développement fût rapide car nous avons assimilés les principes du framework Qt. De plus comme il rassemble des parties déjà développées telles que l’affichage hiérarchique d’un répertoire, ou encore l’envoi de requêtes réseaux vus avec le WebBrowser.

Cependant, nous avons rencontré deux gros problèmes : la gestion des états relatifs au serveur ftp et la gestion de téléchargements multiples (en upload et/ou en download). En effet la gestion des états ftp passe entièrement par des signaux Qt, le problème des signaux est que ceux-ci appellent les slots dans un ordre qui ne peut être défini. Ainsi un serveur ftp basique renvoie l’état connecté en même temps que l’état login ce qui entraînait de gros soucis dû à une première implémentation. Mais ce problème fût rapidement réglé, ce n’est par contre pas le cas pour la gestion des téléchargements. Les téléchargements gèrent parfaitement les multiples uploads mais pour une raison qui nous échappe à l’heure actuelle, il est impossible d’ajouter plusieurs downloads.



### 15.2 Fonctions Principales

#### 15.2.1 WidgetDistant : :connexion(int)

Données :

- Paramètres : i le numéro de la place dans la liste des serveurs, du serveur auquel on souhaite se connecter
- Attributs : DossiersDistants \* distant est la classe contenant le QFtp

Résultat : Connecte le client à un serveur ftp, au travers d’un serveur proxy si c’est nécessaire, à partir de la place du serveur, dans la liste des serveurs déjà enregistrés par l’utilisateur.

```

void WidgetDistant::connexion(int i)
{
    //structure regroupant toutes les informations
    //nécessaires pour une connexion
    struct_serveur temp = parent->getServeurFTP(i);
    if(!temp.proxy.isEmpty()) //s'il y a un serveur proxy
        pour la connexion
    {
        distant->getFtp()->setProxy(temp.proxy, temp.port_proxy);
    }
    distant->getFtp()->connectToHost(temp.url, temp.port);
    distant->getFtp()->login(temp.login, temp.pass);
    home_dir = QString(temp.home); //Lorsque l'état passera à connecté on
        chargera cette page
    serveur_en_cours = i;
}

```

### 15.2.2 WidgetDistant : :get(QString, QIcon)

#### Données :

- Paramètres : QString chemin absolu du fichier à télécharger, QIcon icon du fichier à télécharger
- Attributs : bool verifier\_existe\_deja permet de savoir si l'utilisateur veut vérifier si le fichier existe ou s'il préfère le remplacer à chaque fois

**Résultat** : Place dans la liste des téléchargements, le download souhaitant être réalisé

```

void WidgetDistant::get(QString nom/*chemin absolu*/, QIcon icon)
{
    QString nom_fichier = nom.split("/").last();

    if(distant->getFtp()->state() == QFtp::LoggedIn)
    {
        //Methode permettant de savoir si un fichier existe deja
        bool existe_deja = parent->getLocal()->existeDeja(nom_fichier);
        if(existe_deja && verifier_existe_deja)
        {
            //Lance une fenetre permettant de prevenir l'utilisateur
            //si un fichier est deja present
            widgetRemplacementFichier(parent->getLocal()->getRepCourant()
                +"/"+nom_fichier, icon);
        }
        if(!existe_deja || !verifier_existe_deja)
        {
            struct_put_get temp; //structure regroupant des uploads
                //et des downloads
            //creation du fichier receveur
            temp.file = new QFile(parent->getLocal()->getRepCourant()
                +"/"+nom_fichier);
            temp.nom = nom;
            get_s->append(temp);

            parent->getMessages()->ajouterTelechargement(icon,
                parent->getLocal()->getRepCourant()+"/"+nom_fichier,
                false /*download*/); //ajoute le telechargement
        }
    }
}

```

```

}
else
{
    //Ajoute un message provenant l'utilisateur
    qu'il_n'est pas connecte
    parent->getMessages()->ajouterMessage
        (QIcon(":/FTPImages/mauvais.gif"),
         trUtf8("Vous_n'etes_connectez_a_aucun_serveur."),
         new QLabel(trUtf8("Erreur")));
}
}

```

### 15.2.3 MessagesToolBar : :ajouterTelechargement(QIcon, QString, bool)

#### Données :

- Paramètres : QIcon icon l'icone du fichier, QString text le nom du fichier, bool b true = upload, false = download
- Attributs : QTableWidgetItem \* table est le conteneur et l'afficheur des messages

#### Résultat :

```

void MessagesToolBar::ajouterTelechargement(QIcon icon, QString text, bool b)
{
    l = new QLabel();
    l->setPixmap(icon.pixmap(QSize(50, 50)));
    int i = table->rowCount();
    table->setRowCount(i+1);
    table->setCellWidget(i, 0, l);

    QWidget * w = new QWidget(this);
    QHBoxLayout * layout = new QHBoxLayout(w);

    if(b)
    {
        QLabel * t = new QLabel();
        t->setPixmap(QIcon(":/FTPImages/upload.png").pixmap
            (QSize(15, 15)));

        layout->addWidget(t);
        layout->addWidget(new QLabel(trUtf8("Upload_")+
            text+
            trUtf8("_:_"")));

        QProgressBar * progress = new QProgressBar(this);
        progress->setAccessibleName("upload");
        progress->setAccessibleDescription("vide");
        progress->setMinimum(0);
        progress->setValue(0);
        telechargements->append(progress);
        layout->addWidget(progress);
    }
    else
    {
        idem mais avec download
    }
}

```



```

if(telechargements->value(telechargement_en_cours)
    ->accessibleDescription() == "vide"
    && telechargements->value
        (telechargement_en_cours)->value() == 0)
{
    //Creation de la barre de progression
    QProgressBar * temp =
        telechargements->value(telechargement_en_cours);

    //Permet de mettre a jour la barre de progression
    connect(parent->getDistant()->getFtp(),
        SIGNAL(dataTransferProgress(qint64, qint64)),
        this,
        SLOT(dernierTelechargement(qint64, qint64)));
    if(temp->accessibleName() == "upload")
    {
        parent->getDistant()->lancerPut(telechargement_en_cours);
    }
    else if(temp->accessibleName() == "download")
    {
        parent{textblock*}->getDistant()->lancerGet(telechargement_en_cours);
    }
}

table->setCellWidget(i, 1, w);
table->setCellWidget(i, 2, 0);

table->setCurrentCell(i, 0);

table->resizeColumnToContents(0);
table->resizeColumnToContents(2);

resize(this->size());

//un telechargement est en cours on doit donc l'arreter avant de vider
    la corbeille
parent->getCommandes()->trashToStop();
}

```

## 16 Utilisation du Logiciel

Pour commencer, l'utilisateur peut :

- Créer un nouveau fichier en cliquant sur FICHER -> NOUVEAU (ou via l'icône prévu à cet effet dans la barre d'outils).
- Ouvrir un ou plusieurs fichiers existants en cliquant sur FICHER -> OUVRIR (ou via l'icône prévu à cet effet dans la barre d'outils).

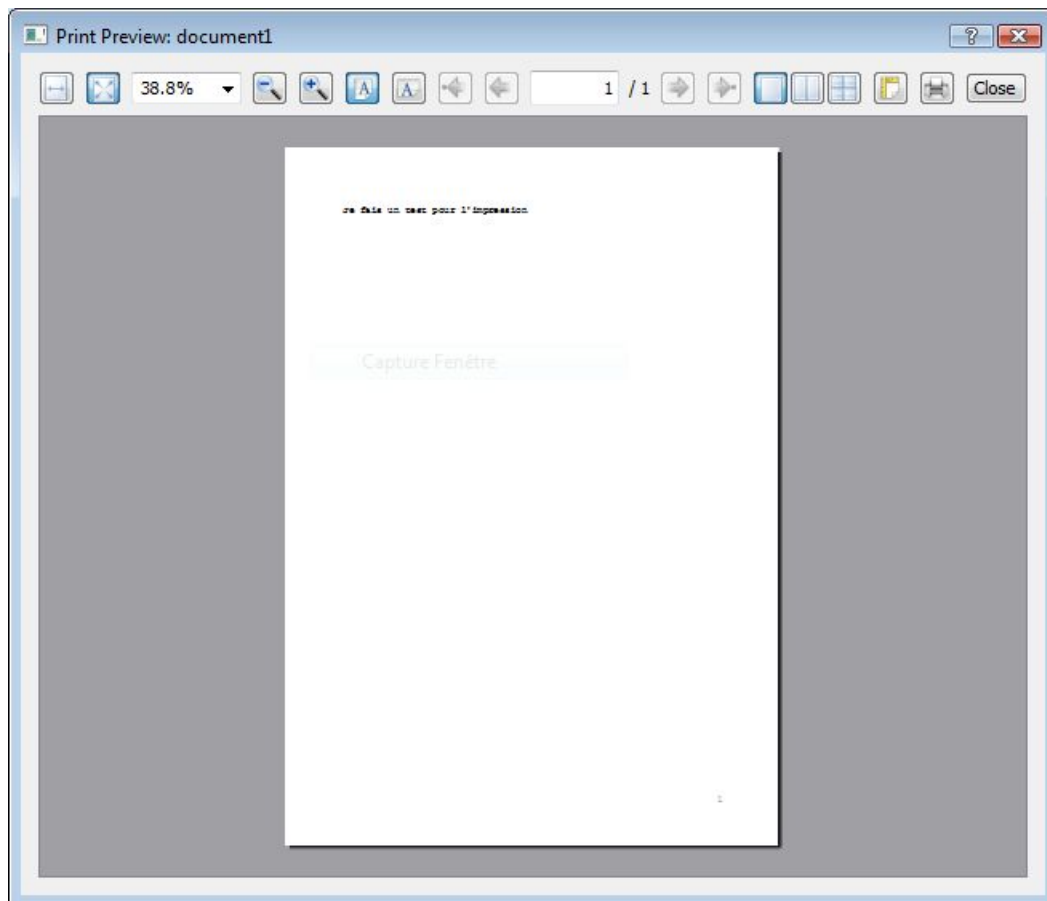
Une fois un document ouvert dans un onglet, il peut être modifié afin de commencer la création du site. Les fonctions liées à l'utilisation du champ de texte fonctionnent automatiquement. Les mots se colorent dès qu'ils sont saisis, la correction s'active à chaque caractères tapés et l'auto-complétion s'active au delà de 3 caractères tapés ou lorsque le raccourci l'activant est tapé (CTRL+espace).

Puis l'utilisateur peut enregistrer dans un fichier le contenu de l'onglet courant :

- FICHER -> ENREGISTRER (enregistre le fichier).
- FICHER -> ENREGISTRER SOUS (demande où sauvegarder le fichier).
- FICHER -> ENREGISTRER TOUT (enregistre tous les onglets de la fenêtre).

L'onglet courant peut être imprimé en cliquant sur :

- FICHER -> IMPRIMER (ou via l'icône prévu à cet effet dans la barre d'outils). Une fenêtre de prévisualisation s'ouvre puis vous pouvez confirmer ou non l'impression.



Il faudra parfois ouvrir un navigateur web afin de tester son site ou pour pouvoir afficher l'aide. Dans ce cas, le navigateur inclu dans le logiciel est fortement conseillé. Pour l'utiliser :

- FICHER -> WEB BROWSER

De même, il sera nécessaire d'utiliser un logiciel basé sur le protocole de transfert de fichiers afin de mettre son site en ligne ou autre. Pour ce faire :

- FTP : FICHER -> FTP BROWSER.

L'utilisateur peut également fermer :

- un fichier en cliquant sur la croix située sur l'onglet correspondant.
- le logiciel entier : il suffit de cliquer sur la croix située sur la fenêtre du programme, ou de faire FICHER -> QUITTER.

Toutes les manipulations de base sont disponibles soit dans la barre d'outils soit dans le menu EDITION :

- EDITION -> COUPER, EDITION -> COPIER, EDITION -> COLLER, EDITION -> SELECTIONNER TOUT.

Si la taille par défaut des caractères ne convient pas, alors on peut la modifier à volonté en utilisant la barre d'outils ou :

- EDITION -> ZOOM PLUS, EDITION -> ZOOM MOINS.

Afin de rechercher rapidement une chaîne de caractère pour pouvoir la remplacer aisément ou simplement la localiser, l'outil de recherche est disponible via :

- EDITION -> RECHERCHER/REEMPLACER

- le raccourci clavier CTRL+F

- l'icône de la barre d'outils. Une barre s'affiche sous les onglets.

Un panneau de préférences permet de personnaliser le logiciel à sa convenance via :

- EDITION -> PREFERENCES.

Ce panneau propose un florilège d'options permettant de rendre le logiciel plus personnalisé et plus agréable pour l'utilisateur :

- personnalisation des langages en ajoutant ou supprimant des mots d'un langage.
- reconnaissance des mots clés automatique(par le biais des balises) ou en choisissant un langage particulier.
- changement de couleurs des listes des mots clés.
- ajout et suppression des icônes de signets qui se placent à côté des numéros de lignes.

En outre, le logiciel permet aussi :

- d'afficher le numéro de ligne via F5 ou AFFICHAGE -> NUMERO DE LIGNE.
- de mettre en plein écran via F11 ou AFFICHAGE -> PLEIN ECRAN.
- d'afficher les barres d'outils et les projets via AFFICHAGE -> BARRE D'OUTILS ...

En cliquant sur le bouton central de la souris et sur les numéros de ligne, un menu s'affiche permettant de choisir l'image correspondant au signet de la ligne concernée. Dans ce menu, vous trouverez également la possibilité d'inverser les numéros de ligne et les icônes des signets.

## 17 Perspectives et conclusion

### 17.1 Perspectives

Ce logiciel a été développé en troisième année de Licence Informatique, il est donc réutilisable selon les critères que nous avons vu en cours : les méthodes sont en virtual pour l'héritage et disposent d'un nombre conséquent d'accesses. Ainsi on peut envisager de pouvoir, au choix, réimplémenter des classes pour modifier le logiciel ou d'ajouter des modules au MainWindow sans problème.

Il demeure cependant de nombreux bugs qu'il faudrait régler avant de penser à développer de nouvelles fonctionnalités. En particulier, le ralentissement dû à l'ouverture de gros fichiers, ou encore le manque d'options pour l'utilisateur. Mais il est déjà bien avancé et permettrait à un programmeur de réaliser son éditeur web en peu de temps.

Un module qui serait des plus intéressant est un SQLBrowser permettant d'afficher les bases, tables et contenu d'un serveur SQL. De plus, il manquerait quelques exemples de codes tels que des pages php ou html avec l'ossature de base d'une page internet.

Nous ne pensons pas qu'un tel projet puisse sortir du lot des nombreux éditeurs web déjà existant mais nous l'avons développé en étant nous même webmaster et donc en y ajoutant un maximum d'options facilitant le développement web. De plus il regroupe un explorateur internet et un client ftp ce qui lui permet de faciliter la vie de l'utilisateur. L'accès aux différents manuels d'aide à la programmation web est très intuitive et permet en un clic d'y accéder.

### 17.2 Conclusions

#### 17.2.1 Fonctionnement de l'application

Ce projet rassemble au final près de 100 fichiers dont 45 classes, créant ainsi MYW, un logiciel utilisé pour bénéficier d'une assistance complète pour coder les pages destinées à la création d'un site web. MYW permet non seulement la gestion d'un nombre quasi illimité de pages, chacune pouvant être sauvegardée, chargée, créé ou détruite. Elles bénéficient toutes d'une coloration syntaxique, d'une vérification de parenthèse, d'une auto-complétion et d'un affichage de lignes. Une arborescence est présente sur la gauche afin de rendre la structure de son projet bien plus claire. Plusieurs options sont aussi disponibles, telles que l'ajout/suppression de mots clefs, la personnalisation des couleurs de mots, et bien sur la plupart des fonctions découlant du copier (coller, couper, tout sélectionner, ...). Il est aussi possible de rechercher et remplacer un mot particulier, et de zoomer dézoomer dans le champ de texte. De plus, le logiciel permet de générer des fichiers html recensant les pages du site pour le site map. Enfin, MYW inclu deux modules essentiels dans son programme : son propre navigateur web, offrant ainsi une compatibilité optimale avec le logiciel et permettant aux utilisateurs de consulter directement l'aide en ligne, sans passer par un logiciel complémentaire ainsi que son propre logiciel de gestion du protocole FTP, donnant aux utilisateurs toutes les cartes dont ils ont besoin pour créer rapidement et facilement un site internet, et le mettre en ligne avec un seul et même logiciel.

#### 17.2.2 Fonctionnement du groupe de travail

Dans l'ensemble, l'équipe était plutôt bien organisée, chacun s'étant adapté au travail de groupe, se partageant avec efficacité les tâches selon le niveau de chacun, et restant un maximum en contact les uns les autres afin de s'aider et ainsi faire avancer l'ensemble. Dès le début, 4 groupes furent ainsi créés : l'apparence graphique générale, la gestion d'un site map, la zone de texte et toutes les fonctions qui y sont rattachées et la création du navigateur web (la création du FTP Browser s'est ajoutée après avoir terminé la partie sur le navigateur). Le groupe entier se rassemblait environ une fois par semaine. Ces réunions étaient faites pour tenter de trouver ensemble des solutions aux divers problèmes qui apparaissaient, puis de planifier les prochaines étapes de développement. Ainsi, personne n'était en retard et chacun avait quelque chose à faire, tout en gardant une compatibilité maximum entre les classes, à condition bien sûr, d'être présent lors de ces réunions. Puis, suite à la première mise en commun de toutes les classes, nous les avons rassemblées dans un seul et même projet, ainsi, après chaque réunion, une nouvelle version du projet était créée et envoyée par mail à chacun des programmeurs, ce qui permettait de cette façon, à tout le monde de travailler à partir d'une même version, tout en essayant au maximum de n'avoir qu'une seule personne travaillant par classe à la fois.

Le résultat de cette organisation aboutit enfin à créer MYW, et malgré les problèmes de communications avec les personnes chargées de l'indentation, nous sommes fier du travail qui a été accompli.

## 18 Bugs recensés

### 18.1 WebBrowser

- Un double clic sur un onglet place l'image en position 0, 0 du widgetParent.
- si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.

### 18.2 FTPBrowser

- Un clic sur fichier, puis une connexion ne met pas upload/download à `setEnabled(true)` dans `CommandeToolBar`.

### 18.3 FrameAide

- Le fait de déplacer le curseur avec le clavier ne fait pas apparaître le mot (entre le mot et les parenthèses).

### 18.4 Mainwindow

- Undo et redo ne fonctionne plus.
- Il faudrait griser `QAction->setEnabled(false)` les actions non réalisables.
- Lorsque l'on ouvre un fichier, une étoile apparaît lorsque l'on clique dans l'éditeur, même sans modifications.

### 18.5 WidgetLigne

- Les signets ne bougent pas lorsque l'on insère du texte avec le bouton coller de `MainWindow`.
- On ne peut pas cacher le `widgetLigne`

### 18.6 Coloration

- La fonction `highlightBlock` était contruite avec deux boucles `while` imbriquées qui parcouraient deux fois la zone de texte il y avait alors de gros ralentissement. Cette boucle a été remplacée par un `for`, il y a donc moins de ralentissement (encore présent pour les gros fichiers).
- Avec cet ajout du `for`, le document n'est parcouru qu'une seule fois et donc il est impossible de colorer du code php, javascript ou css sur une même ligne (sauf pour le html car c'est une coloration sur une ligne est donc une méthode différente). Cependant, nous avons préféré faire ce choix plutôt que de bloquer l'utilisation de notre éditeur.
- Problème de gestion des `/* */` et `"` et `”`. Dans un gros fichier, si on ouvre un commentaire multiple, et qu'on le ferme au milieu du fichier, la recoloration des mots clés est très lente.

### 18.7 Syntaxe

- Ralenti considérablement tout le projet car la fonction de vérification du parenthésage est appelée à chaque modification du texte sur tout le texte.
- Si on insère du texte juste avant un bloc, il sera grisé car l'éditeur prend la même police et couleur que le texte qui suit pour les insertions. Or comme on est juste devant un bloc celui-ci est grisé et donc le texte que l'on insère devant aussi. Nous n'avons pas réussi à résoudre ce problème car nous n'avons pas réussi à trouver comment empêcher l'éditeur d'avoir ce comportement.

## 19 Annexe

Cette annexe contient l'évolution des différentes versions du logiciel. On peut ainsi y observer les ajouts successifs ainsi que la description des bugs rencontrés et de leurs résolutions.

### 19.1 ALPHA 0.1 20/03/09

La première version contient :

Le mainWindow, contient la fenetre principale avec un QTextEdit  
Récupération d'un mot de l'éditeur selon la position du curseur ou de la souris  
base de la coloration syntaxique

### 19.2 ALPHA 0.2 23/03/09

Ajout du WebBrowser, pour lancer une recherche appuyez sur F1 dans l'éditeur  
Ajout de la variable `systeme_relation_fichier` dans le fichier `mesConfig.h` qui est définie à `../` pour windows  
et à `./` pour les autres systèmes.

### 19.3 ALPHA 0.3 24/03/09

Découverte et rectification d'un bug au niveau de l'historique et des marques pages qui empechait le lancement du WebBrowser

### 19.4 ALPHA 1 25/03/09

**Modifications :**

Rangement du projet WebBrowser est dans un dossier WebBrowser, fichiers textes pour l'auto-complétion et auto-coloration dans un dossier fichiers, images dans de le dossier images et ajout de fichiers qrc pour l'intégration à l'exécutable  
Il n'y a plus besoin de modifier la valeur de la variable `systeme_relation_fichier` dans le fichier `mesConfig.h`, cela est automatique.  
Modification des marques pages internes (non visuelles) pour ne plus avoir d'erreurs de segmentations.  
Ajout de belles icônes et plusieurs boutons sur `monmainwindow` mais aussi sur le WebBrowser.  
Coloration du css, prise en compte des commentaires simples et des " ".

### 19.5 ALPHA 1.1 28/03/09

**Modifications :**

Ajout de la classe Hiérarchie qui gèrera la partie de gauche montrant l'architecture du projet en cours + ajout du `TreeView` pour fichier et dossiers  
Ajout de la classe `Projet` qui gèrera le ou les projets en cours  
Modification de la taille du `dockWidget` dans le fichier `mesConfigs.h`  
Derniers réglages pour les marques pages et l'historique du WebBrowser  
Ajout de marques pages en double cliquant sur l'onglet du site voulu (passez la souris sur l'icône)  
Construction du menu pour le WebBrowser  
Enlèvement de la frame aide quand l'editeur perd le focus  
Vous pouvez désormet utiliser l'aide pour cela placez votre souris sur un mot et suivant le langage il vous renverra sur la bonne page  
Modification de `monmainwindow` pour afficher l'architecture du workspace  
Ajout du `completer`.

**Bugs :**

Problème d'intégration du compléteur, une réorganisation du projet est à l'étude.

## 19.6 ALPHA 1.2 28/03/09

### Modifications :

Suppression du compléteur car il n'est pas encore compatible avec le reste du projet

## 19.7 ALPHA 1.3 31/03/09

Début de la réflexion sur le nom du logiciel.

### Modifications :

Mise en place du fichier readMe répertoriant les différentes versions du projet

Définition d'un workspace

Enregistrer sous dans mainWindow avec son icône dans le menu et dans la barre d'outil

Nouvelle classe modifierLangage permettant d'afficher un widget qui permet de supprimer/ajouter/réinitialiser les mots clés d'un langage

Ajout de fichiers textes initiaux pour les mots clés des langages dans le répertoire fichiers\_initiaux

Suppressions de fichiers de mots clés inutiles

Ajout de la classe completer qui contient le compléteur des mots clés des langages

ItemHierarchie

Ajout de la variable workspace dans mesConfig.h

Implémentation du menu du WebBrowser

Implémentation d'enregistrer, enregistrer tout et ouvrir fichier dans mainWindow

Le widgetLigne (l'indicateur du numéro de ligne) est presque fini

Rangement de coloration.cpp

Meilleure prise en charge du html (non complète)

Réécriture presque complète du compléteur afin de regrouper ses propres fonctions dans sa propre classe

Implémentation du double clic pour ouvrir un fichier à partir de l'arborescence

Modification de editeur.cpp pour placer les méthodes appelées par Completer tout en bas du fichier

### Bugs :

#### WebBrowser :

Double Clic sur onglet place l'image au 0, 0 du widgetParent

Marques-Pages dans dossier lance startDrag au mauvais moment [corrigé]

Si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.

Cookie

Javascript

#### MainWindow :

Undo et redo ne fonctionnent plus

Ajouter le nombre de caractères et de lignes d'un fichier

Ouvrir plusieurs fichiers

il faudrait griser Qaction->setEnabled(false) les actions non réalisables (ex : enregistrer un fichier sans nom)

Cacher widgetLigne ne fonctionne pas

#### Coloration :

Lors de l'ouverture d'une balise, reste dans la même couleur même si la balise n'appartient plus aux mots clés



Doublons lors de l'affichage du dernier mot dans modifierLangage()

Non prise en charge de la re-coloration des mots après ajout ou suppression [corrigé]

NE SURTOUT PAS AJOUTER UN MOT VIDE dans modifierLangage()

**Syntaxe :**

On la soupçonne de ralentir le projet

## 19.8 Beta 0.1 02/04/09

Début du vote pour le nom du projet. Passage de la version Alpha à la version Béta pour la première présentation au tuteur

**Modifications :**

Preferences.cpp et .h pour gérer les préférences générales

Statistiques nbLignes & nbCaractères du document en cours dans la barre des statuts [à finir]

Ajout des signets (a côté des lignes) permettant sur le clic gauche d'apparaître/disparaître, sur le clic droit de modifier l'image et sur le clic central de faire apparaître le menu pour gérer les fiches

Les modifications portent sur la hierarchie pour bien tout synchroniser quand on ouvre un fichier même extérieur au workspace pour qu'ils apparaissent quand même

Marques-Pages dans dossier lance startDrag au mauvais moment [corrigé/a vérifier]

Doublons lors de l'affichage du dernier mot dans modifierLangage() [corrigé]

Non prise en charge de la recoloration des mots après ajout ou suppression [corrigé]

**Bugs :**

**WebBrowser :**

Double Clic sur onglet place l'image au 0, 0 du widgetParent

Si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.

Cookie

Javascript

Réimplémenter panneau\_url

**Mainwindow :**

Undo et redo ne fonctionnent plus

Ouvrir plusieurs fichiers

il faudrait griser QAction->setEnabled(false) les actions non réalisables (ex : enregistrer un fichier sans nom)

Cacher wigdetLigne ne fonctionne pas

**Coloration :**

Lors de l'ouverture d'une balise, reste dans la même couleur même si la balise n'appartient plus aux mots clés

NE SURTOUT PAS AJOUTER UN MOT VIDE dans modifierLangage()

**Syntaxe :**

On la soupçonne de ralentir le projet

## 19.9 Beta 0.2 15/04/09

Le nom du projet a été choisi, MyW l'acronyme de Make your Web (à prononcer my way).

### Modifications :

Kinetic et branch animationBranch pour des essais

Module FTPBrowser

Plus qu'un seul fichier QRC

Classe erreur transformée en module Erreur

Pleins de petites choses liées au menu générale du webBrowser entre autres : ajout de dossier de marques-pages, suppression de l'historique et des marques pages avec plusieurs options...

Rajout d'une classe choisirLangage, qui permettra de choisir le type de coloration, qui par défaut sera automatique, on pourra choisir automatique, et / ou personnalisée, c'est-à-dire colorer des mots de javascript dans une balise php par exemple.

Demande d'enregistrement lorsque l'on ferme un onglet du mainWindow

Le dock widget devient redimensionnable

Ouverture multiple de fichiers

Confirmation lorsque l'on désire quitter l'application

Afficher/cacher dockWidget projet

Ajout de tous les accesseurs du MainWindow

Ajout d'icônes dans les onglets (après sélection d'un type php, javascript grace à l'extension)

Modification du QTabWidget préférence en QDialog

Ajout de fonctions permettant d'utiliser la classe choisirLangage

Modification de la fonction getListe() qui renvoie une liste de mots simples

Correction du bug qui faisait disparaître la coloration lors d'un saut de ligne

La fonction qui renvoie le type de langage à la FrameAide, lorsqu'on choisie coloration php + javascript dans les options, la souris "récupère" aussi les mots de javascript

Coloration de la QLineEdit en fonction des mots clés qu'on rentre : si les mots clés sont bien rentrés alors coloration verte, sinon rouge

Si le mot clé est mal rentré, c'est-à-dire s'il ne correspond pas à l'expression régulière qu'on attend (ex : html : <[a-zA-Z0-9]\*) alors il y a en plus de la coloration rouge, ainsis que l'affichage d'un texte qui permet d' "aider" l'utilisateur

### Bugs :

#### WebBrowser :

Double Clic sur onglet place l'image au 0, 0 du widgetParent

Si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.

Cookie

Javascript

Réimplémenter panneau\_url

#### FTPBrowser :

Clic sur fichier puis connexion ne met pas la flèche upload/download à setEnabled(true)

QLineEdit de widgetDistant ne revient jamais blanc après une erreur

Téléchargements à régler ils ne démarrent pas au bon moment (il faut en lancer 2) et les connecter avec les QProgressBar

#### Mainwindow :

Undo et redo ne fonctionnent plus

Enlever l'enregistrement si le document est vide

il faudrait griser `Qaction->setEnabled(false)` les actions non réalisables (ex : enregistrer un fichier sans nom)

Cacher widgetLigne ne fonctionne pas

**Coloration :**

Lors de l'ouverture d'une balise, reste dans la même couleur même si la balise n'appartient plus aux mots clés

NE SURTOUT PAS AJOUTER UN MOT VIDE dans `modifierLangage()`

**ModifierLangage :**

Problème d'affichage des widgets qui est à remanier pour que ce soit plus présentable.

**Syntaxe :**

On la soupçonne de ralentir le projet

## 19.10 Beta 0.3 23/04/09

**Modifications :**

Réécriture globale de tous les noms de classes, attributs de classes, fonctions...

Début des commentaires à la Doxygène

Ajout d'un fichier `Entete.h` pour les constantes et struct utiles aux classes du dossier Editeur

Nouvelle classe : `editionwidgetliste` permettant d'ajouter/supprimer des signets

Ajout du bouton Rechercher/Remplacer dans le menu et barre d'outils

Dossier `RechercherRemplacer` contenant `FindDialog.h` et `.cpp`

Ajout d'une classe `changerCouleur` qui permet de rajouter un onglet dans les préférences, qui permettra de choisir la couleur de la coloration de chaque groupe de mot clé

Ajout du compléteur

Ajout de toutes les classes du `FTPBrowser`

Correction d'une erreur de segmentation lorsqu'aucun settings n'est présent sur la machine

Suppression d'un bug dans l'ouverture du `WebBrowser` pour la classe `editeur`

Amélioration de la classe `widgetLigne` : Les signets doivent exister dans un `QSettings` prédéfinis, changement de l'algorithme permettant de positionner les signets et les lignes

Les fichiers vides ne s'enregistrent plus à la fermeture.

Ajout du bouton Imprimer dans le menu et la barre d'outils

Gestion Impression

Ajout de ligne dans la fonction `fichierNouveau()` pour que quand un nouvel onglet est créé, il prenne en compte les preferences selectionnées

Ajout d'un connect avec le bouton général appliquer qui appelle le slot `recolorer()` et relance la coloration du document pour chaque onglet ouvert

Ajout d'un `getChoisirLangage()`

Ajout d'un slot `recolorer()`

Modification de l'affichage du widget `modifierLangage` il n'y a plus de problème de décalage

Modification des expressions régulières pour les tests qui autorisent l'ajout d'un mot

Ajout d'un slot `recolorer()`

Réglage d'un bug qui faisait qu'on ne pouvait pas colorer plusieurs lignes de html

Modification de la fonction `langage`. ex : si dans les préférences on choisit automatique et javascript, alors si on ouvre la balise `php` et comme else appartient a `php` et javascript ça renverra `php`.

Ajout de méthode qui récupère le type de bloc en fonction de la position de la souris pour savoir dans quel langage on est.

Correction du bug qui faisait que ça plantait si on recolorait un mot sur la même ligne

Ajout de modification pour que quand on clique sur appliquer, ça se recolor automatiquement

Ajout d'un booléen pour savoir si une case de préférences est cochée ou pas dans choisirLangage

Ajout d'un slot recolorer() dans choisirLangage

Ajout/suppression de signets dans editionwidgetliste

Mise à jour des signets dans l'éditeur de texte directement après modification dans les préférences

Modification de toutes les icônes par défaut de Qt (non identique sous windows et linux) en vrai icône

#### **Bugs :**

##### **WebBrowser :**

Double Clic sur onglet place l'image au 0, 0 du widgetParent

Si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.

Cookie

Javascript

Réimplémenter panneau\_url

##### **FTPBrowser :**

Clic sur fichier puis connexion ne met pas upload/download à setEnabled(true) dans CommandeTool-  
Bar les QProgressBar

corbeillevide.png à implémenter [corrigé]

Suppression de fichier/dossier [corrigé]

Téléchargement à revoir

##### **Mainwindow :**

Undo et redo ne fonctionnent plus

Enlever l'enregistrement si le document est vide[Corrigé]

il faudrait griser QAction->setEnabled(false) les actions non réalisables (ex : enregistrer un fichier sans nom)

Cacher wigdetLigne ne fonctionne pas

##### **Coloration :**

Lors de l'ouverture d'une balise, reste dans la même couleur même si la balise n'appartient plus aux mots clés

Coloration non complète à cause d'une modification de la fonction de coloration, seuls les langages php et javascript sont reconnus.

Problème de gestion des /\* \*/ et "" et ''

##### **ModifierLangage :**

Problème d'affichage des widgets qui est à remanier pour que ce soit plus présentable.

##### **Syntaxe :**

On la soupçonne de ralentir le projet

## 19.11 Beta 0.4 14/05/09

### Modifications :

- FTPBrowser : suppression de fichiers, de dossiers en local et sur serveur
- FTPBrowser : possibilité d'ajouter un serveur proxy
- FTPBrowser : modification de serveurs possible
- FTPBrowser : Fichier qrc renommer en FTPBrowser.qrc
- FTPBrowser : Dossier images renommer en FTPIImages/
- Fichier qrc du SQLBrowser renommer en SQLBrowser.qrc
- Dossier images du SQLBrowser renommer en SQLImages/
- FTPBrowser : La corbeille se vide (changement d'image) lors d'un clic dessus
- FTPBrowser : Clic droit -> supprimé fonctionne pour les fichiers comme pour les dossiers
- FTPBrowser : modifications de quelques icônes (upload, download, reload...)
- FTPBrowser : lineEdit distant opérationnel
- FTPBrowser : upload ok
- Compléteur en état de marche
- Remplacement des doubles whites par un for dans coloration

### Bugs :

#### WebBrowser :

- Double Clic sur onglet place l'image au 0, 0 du widgetParent
- Si on agrandit la fenêtre cela perturbe le QMenu des dossiers de marques-pages.
- Cookie
- Javascript
- Réimplémenter panneau\_url

#### FTPBrowser :

- Clic sur fichier puis connexion ne met pas upload/download à setEnabled(true) dans CommandeToolBar les QProgressBar
- Multi Download à finir
- lineEdit local on reload reste rouge ou boucle sans fin à voir WidgetLocal : :recharger()
- Pb sur distant, lineEdit d'emplacement bug à la main [corrigé]
- Téléchargement à revoir[corrigé]

#### Mainwindow :

- Undo et redo ne fonctionnent plus
- il faudrait griser QAction->setEnabled(false) les actions non réalisables (ex : enregistrer un fichier sans nom)
- Cacher wigdetLigne ne fonctionne pas

#### FrameAide :

- Le fait de déplacer le curseur avec le clavier ne fait pas apparaitre le mot (entre le mot et les parenthèses)

#### Coloration :

- Lors de l'ouverture d'une balise, reste dans la même couleur même si la balise n'appartient plus aux mots clés
- Coloration non complète à cause d'une modification de la fonction de coloration, seuls les langages php et javascript sont reconnus.
- Problème de gestion des /\* \*/ et "" et ''

#### ModifierLangage :

- Problème d'affichage des widgets qui est à remanier pour que ce soit plus présentable.

#### Syntaxe :

- On la soupçonne de ralentir le projet