# Rapport personnel

**Travail réalisé**

Au début de ce projet, j'ai dû m'approprier le sujet des expressions régulières car je n'avais pas une grande connaissance dans ce domaine. En effet, je ne connaissais pas le vocabulaire spécifique utilisé tel que les termes "match", "pattern", "group", "capture", etc. ainsi que les différentes syntaxes pour écrire les expressions régulières.

J'ai donc commencé par faire des recherches sur internet pour comprendre les bases de ce sujet. J'ai notamment utilisé des sites pour tester des expressions régulières en temps réel et pour voir les résultats correspondants.

Dans le cadre de ce projet, j'ai été chargé de travailler sur la partie UML de l'interface graphique de l'application. J'ai conçu les diagrammes de classes représentant les trois fonctionnalités principales de l'application, à savoir l'analyse, la construction et le test d'expressions régulières.

En ce qui concerne la partie UML de l'interface graphique, j'ai commencé par réaliser une étude des expressions régulières afin de bien comprendre les termes tels que match, capture, groupe de capture et quantificateur comme expliqué ci-dessus. J'ai ensuite travaillé à partir du travail d'Ewen, qui avait déjà réalisé le diagramme UML des expressions régulières, pour mettre en lien le diagramme UML de l'interface graphique et trouver les relations entre les différentes classes.

Dans le diagramme UML de l'interface graphique, la classe BlocksPanel représente la section des blocs dans l'interface utilisateur de l'application de construction d'expressions régulières. Cette classe hérite de la classe JPanel et contient les blocs sous forme d'imbrication que l'on peut réorganiser et supprimer grâce à des boutons de contrôle.

En outre, Ewen a choisi d'instaurer la classe Block pour représenter les différents types de blocs de construction pour les expressions régulières. Cette classe offre une interface commune pour manipuler les différents types de blocs, tels que les classes de caractères, les quantificateurs et les groupes de capture.

Enfin, il est important de noter que ces différentes classes ont des méthodes et des attributs communs, tels que des fonctions de validation de la syntaxe, pour éviter la duplication de code et faciliter la maintenance de l'application.

Ainsi, j'ai travaillé sur les fondements du projet en créant une structure de base pour les expressions régulières en élaborant un diagramme UML qui va servir de base pour la suite du développement de l'application.

**Mise en relation des diagrammes UML**

Pour établir les relations entre les différentes classes de l'interface graphique, j'ai dû m'appuyer sur le travail préalable réalisé par Ewen, qui avait déjà élaboré le diagramme UML des expressions régulières comme je l'ai déjà expliqué au-dessus. J'ai ainsi pu mettre en lien les deux diagrammes UML et trouver les relations entre les différentes classes, notamment la classe BlocksPanel qui représente la section des blocs dans l'interface utilisateur de l'application de construction d'expressions régulières. Cette classe contient des objets Block qui représentent les différents types de blocs de construction pour les expressions régulières et permettent de les réorganiser ou de les supprimer selon les besoins de l'utilisateur. Ainsi, la classe BlocksPanel est utilisée pour représenter les blocs dans l'interface graphique, alors que la classe Block est utilisée pour représenter les différents types de blocs de construction pour les expressions régulières dans le diagramme de classes pour l'analyseur d'expressions régulières.

La classe MainWindow utilise et crée des instances de la classe Expression.

La classe Expression a une relation d'agrégation avec la classe AbstractBlock (1 Expression peut avoir 0 ou plusieurs AbstractBlocks).

La classe AbstractBlock est une classe abstraite qui est étendue par la classe Block et la classe TerminalBlock. La classe Block a une relation de composition avec la classe AbstractBlock (1 AbstractBlock a au moins 1 Block).

Les classes concrètes telles que BlockCaptureGroup, BlockRepeat, BlockRepeatExactly, BlockRepeatAtMost, BlockRepeatAtLeast, BlockOneOrMore, BlockZeroOrMore, BlockOptional, BlockWord, BlockLine, BlockLookahead, BlockLookbehind, BlockLiteral, BlockCharacterRange, BlockAlphanumericOrUnderscore, BlockNewline, BlockDigit, BlockAnyCharacter, BlockUnicodeClass, BlockCaptureGroupReference, BlockRawRegex étendent soit la classe Block soit la classe TerminalBlock.

Les classes concrètes, celles qui ne peuvent pas être instanciées directement sont des feuilles c'est-à-dire les terminalblock dans l'arbre des classes et implémentent la méthode toRegex() pour renvoyer une expression régulière.

La classe BlockOr étend la classe AbstractBlock et a une relation de composition avec la classe AbstractBlock (1 BlockOr a au moins 2 AbstractBlocks).

**Travail en cours et à suivre**

Pour continuer mon travail sur l'application de construction d'expressions régulières reg7, je vais maintenant me concentrer sur l'implémentation de la barre de recherche dans le menu d'ajout de blocs. Cette fonctionnalité permettra à l'utilisateur de filtrer les différents types de blocs disponibles en fonction d'un mot-clé saisi dans la barre de recherche. Cette fonctionnalité est

essentielle pour faciliter la recherche de blocs spécifiques, en particulier lorsque la bibliothèque de blocs devient importante. Je vais m'assurer que la barre de recherche fonctionne correctement avec tous les types de blocs disponibles et que l'affichage des résultats filtrés est clair et facilement compréhensible pour l'utilisateur.

D’autre part, j’en profite pour préciser que le travail réalisé sur le diagramme UML de l’interface graphique peut être amené à évoluer si l’on décide d’ajouter des fonctionnalités par exemple ou si la conception comporte des erreurs.

**Remarque personnel**

Je tiens également à préciser que mon implication dans la junior entreprise en tant que Vice-Président notamment avec la demi-finale me prend beaucoup de temps sur mes études et mon temps libre. Ainsi, j'ai donné mon possible pour m'investir sur le projet reg7 mais je n'ai pas encore pu m'attarder autant que je l'espérais sur le code de l'interface graphique et j’ai préféré me concentrer sur la conception théorique de cette dernière. Il me paraissait important de le faire savoir dans ce rapport personnel pour éviter de mauvaises surprises par la suite.

**Alexandre Trotel**, étudiant en première année en SN à l’ENSEEIHT (reg7)