TP1: Application de gestion de notes - Rapport complet

Cours: MGL843 **Titre du projet**: TP1 - Modélisation et analyse d'un projet TypeScrip **Date de remise**: 1er février 2026 **Remis à**: Professeur Christopher Fuhrman **Équipe 1 Auteurs**: Marc-André Besner, Stacy Chan, Ilan Hoquidant, Stanislas Mabin

Table des matières

Introduction

Ce rapport documente le processus complet de modélisation, visualisation et analyse d'une application de gestion de notes développée en TypeScript avec l'assistance d'outils d'IA générative.

Objectifs du TP:

- Développer une application TypeScript avec assistance d'IA générative
- Générer et analyser un modèle Famix du projet
- Visualiser l'architecture avec Roassal
- Exporter et analyser les données en format CSV

Structure du dépôt GitHub

```
tp1-notes-app
├── tp1-export-pharo/ - la classe Pharo utilisée pour la génération des données en
.csv  (Partie 3)
├── tp1-notes-app/ - application TypeScript générée par l'IA (Partie 0)
│   └── tests/
├── images/ - contient toutes les captures d'écran (Partie 2 & 4)
├── classes_export.csv - le .csv généré par Pharo (Partie 3)
├── tp1-notes-app.json - le modèle Famix généré par ts2famix (Partie 1)
├── Visualisation.py - le code utilisé pour généré une visualisation des données
.csv (Partie 3)
├── README.md - notre rapport
```

Partie 0: Utilisation de l'IA générative

Contexte du développement

Le projet TypeScript a été entièrement généré en utilisant **GitHub Copilot** pour les fonctionnalités et les tests, selon les directives du laboratoire:

Questions - Utilisation de l'IA générative

Q1: Avez-vous déjà utilisé des outils d'IA générative pour programmer avant ce laboratoire ? Si oui, lesquels et dans quel contexte ?

Réponse:

Oui, j'ai déjà utilisé GitHub Copilot, DeepSeek, Perplexity et OpenAI pour écrire du code Python dans des fichiers Jupyter ou des fichiers .py.

Q2: Quelle a été votre expérience globale en utilisant l'IA générative pour créer ce projet TypeScript ? Quels aspects ont bien fonctionné et quels aspects ont été plus difficiles ?

Je n'avais jamais utilisé l'IA pour faire du TypeScript et nous n'avions pas beaucoup d'expérience dans ce langage. En général ça s'est bien fait. Nous avons débogué les quelques erreurs 1 à la fois principalement reliées à l'importation des paquets et à la configuration du fichier json.

Q3: Combien de temps avez-vous pris pour créer ce projet TypeScript avec l'IA générative ? Avez-vous respecté le temps suggéré ?

Temps suggéré: 45 à 60 minutes
Temps réel utilisé: Oui, nous avons pu le compléter en environs 30 à 45 mins.

Décomposition du temps:

```
Phase 1: Setup du projet (5-10 min)
  - npm init
  - Installation des dépendances (TypeScript, Jest, Express)
  - Configuration de tsconfig.json

Phase 2: Génération du code principal (10 min)
  - Classe Note (modèle)
  - Classe Person (modèle)
  - Classe NotesManager (logique)
  - Classe principale (index.ts)

Phase 3: Tests unitaires (5 min)
  - Configuration Jest + ts-jest
  - Écriture des tests pour Note
  - Écriture des tests pour Person
  - Écriture des tests pour NotesManager

Phase 4: Raffinements et validation (5 min)
  - Correction des erreurs de compilation
  - Vérification des tests
  - Documentation du code
```

```
Phase 5: Figuration du code (15 min)
  - Vérification finale
  - Gel du projet
```

Respect du temps: [OUI et justification]

Analyse:

- Le temps suggéré était **réaliste** pour un petit projet
- Il y a eu quelques erreurs avec les paquets, les classes et les configurations, mais Copilot à rapidement pu les corriger 1 par 1 avec des instructions supplémentaires.

Partie 1: Modélisation du projet TypeScript

Étapes suivies :

1. Téléchargement de la plus nouvelle version de ts2famix (npm install -g ts2famix)
2. Génération du modèle Famix (ts2famix -i tsconfig.json -o tp1-notes-app.json
3. Importer le modèle Famix (tp1-notes-app.json) dans Moose
4. Naviguer dans le modèle pour voir les propriétés, etc.

Questions - Modélisation du projet TypeScript

Q1: Pourquoi appelle-t-on le modèle généré par ts2famix un « modèle de code source » ?

Le modèle généré par ts2famix représente le code source d'une façon structurelle pour des fins de visualisation et analyses. Il concentre sur les entités du programme et est indépendant du langage de programmation. Dans ce sens, il est très utile pour comprendre l'architecture du code source d'un programme.

Q2: Quelles sont les différences entre un modèle de code source Famix et un modèle de classes TypeScript en UML ? Peut-on tout modéliser de TypeScript dans les diagrammes de classes UML ? Soyez précis dans votre réponse.

Pour commencer, un modèle Famix et un modèle de classes en UML n'ont pas le même but. Famix a été conçu principalement pour les analyses des mesures ainsi que pour la réingénierie. Les modèles de classes en UML ont été développés pour la visualisation la structure d'un logiciel. Famix concentre sur les entités, tel que les classes, méthodes et variables. Par contre, les modèles de classes en UML représentent le code en objets et leur relation. En général, Famix est employé pour comprendre du code complexe déjà existant alors que les modèles en classes UML sont utilisés pour la conception ou pour la documentation d'un logiciel.

Est-ce qu'on peut modéliser de TypeScript dans les diagrammes de classes UML? Oui et non. Typescript supporte la programmation orientée objet et fonctionnelle. Tant que les parties et les usages orientés objet, on peut le modéliser dans les diagrammes de classes UML. Toutefois, pour les usages en tant que programmation fonctionnelle, on ne peut pas le modéliser dans le diagramme de classes UML, car il peut exister des entités hors des objets.

Partie 2: Visualisation du projet TypeScript

Processus de visualisation avec Roassal

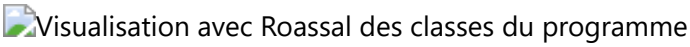
Le projet TypeScript a été visualisé en utilisant **Roassal** selon le tutoriel: https://fuhrmanator.github.io/tuto-Famix-ts/

Étapes suivies :

1. À partir de la partie précédente, le modèle Famix est importé dans Moose
2. Copier le code de visualisation Roassal fourni du tutoriel du professeur (https://fuhrmanator.github.io/tuto-Famix-ts/)
3. Lancer le code et naviguer dans les onglets et les pages pour en apprendre plus sur le modèle.

Captures d'écran de Roassal

Figure 1: Visualisation complète du projet TypeScript


Visualisation avec Roassal des classes du programme

```
Cette visualisation complète montre :
- Toutes les classes du projet (Note, Person, NotesManager)
- La centralité des classes (largeur et hauteur des boîtes proportionnelle à
l'importance)
-- La largeur représente le nombre des attributs dans la classe
-- La hauteur représente le nombre des méthodes dans la classe
```

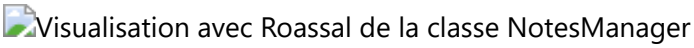
Figure 2: Navigation détaillée - Classe NotesManager

Visualisation avec Roassal de la classe NotesManager

Figure 3: Navigation détaillée - Classe Note 
Visualisation avec Roassal de la classe Note

Figure 4: Navigation détaillée - Classe Person 
Visualisation avec Roassal de la classe Note

```
La visualisation détaillée inclut :
- Une liste de toutes les méthodes de la classe choisie
- Des signatures de méthodes
- Une liste de tous les attributs de la classe choisie
```

Questions - Visualisation du projet TypeScript

Q1: Quelles sont les classes remarquables dans le projet ? Comment le voyez-vous dans la visualisation ?

La classe NotesManager a le plus des méthodes ainsi que la classe Note a le plus des attributs. On le voit clairement dans la figure 1 où la classe NotesManager est la boîte (rouge) la plus longue et la classe Note est la boîte (grise) la plus large. On peut facilement déduire que la NotesManager s'occupe surtout des fonctionnalités et que Note est une classe contenant les valeurs à stocker. Finalement, on a la classe Person, y

contient deux attributs et 5 méthodes. Elle est la boîte rouge foncé et elle est plus longue que Note et moins longue que NotesManager.

Q2: Expliquez le rôle de ces classes dans le projet. Pourquoi sont-elles importantes ?

NotesManager est la classe centrale du projet comme service de gestion centralisé des notes. Elle gère les opérations CRUD sur les notes, est la pointe d'entrée unique pour la manipulation de données et orchestre l'interaction entre les composants.

La classe Note contient tous les attributs d'un objet Note et son constructeur. Alors, on a besoin cette classe pour initialiser une note qu'on peut attribué à une personne.

La class Person contient les attributs d'une personne et les accesseurs et les mutateurs de ces attributs. Il est important, car chaque note (classe Note) est liée à une personne (classe Person).

Q3: Commentez sur la qualité de la conception du projet. Y a-t-il des classes qui semblent mal conçues ? Pourquoi ?

Il n'y a pas de couplage non nécessaire entre les objets. Il y a une cohésion claire dans les classes Person et Note. Par contre, la classe NotesManager a beaucoup de responsabilités qui ne suit pas le principe 'Single Responsability' (SOLID). Étant donné que le programme est très petit, sa structure actuelle est correcte. Toutefois, le principe 'Single Responsibility' devient plus important avec le temps pendant que le programme agrandit. NotesManager a quatre responsabilités distinctes : la persistance, la logique métier, l'exportation des données et la recherche des données. Une amélioration possible sera de diviser NotesManager en quatre classes : NotesRespository, NotesManager, NotesSearch et NotesExporter. NotesRepository s'occupera de la persistance, tel que 'load' et 'sauvegarde' des notes. La classe NotesManager restera et elle s'occupera de la logique métier (CRUD - Create, Read, Update and Delete), telles que la création et la lecture des notes. NotesSearch fera la recherche des notes avec des queries. Et finalement, NotesExporter fera l'exportation des données. Alors, si on veut ajouter du formatting à l'exportation ou la logique aux queries pour faire les recherches customisés, il y a aura les classes dédiées évitant une grande classe qui fait tout.

Partie 3: Exportation des données

Création de la classe Pharo pour exportation CSV

Une classe Pharo a été créée pour exporter les données du modèle Famix en format CSV, utilisant la bibliothèque **NeoCSV**. La classe Pharo se trouve dans le dossier tp1-export-pharo du dépôt GitHub. (https://github.com/mab001/tp1-notes-app/tree/57721a9cd34421cb2b80db413ded75e1687730a3/tp1-export-pharo)

La première étape de la création de la classe a été de créer un package **TP1-Export** dans pharo pour ranger la nouvelle classe.

Classe Pharo: TypeScriptToCSVExporter

```
Object subclass: #TypeScriptToCSVExporter
    instanceVariableNames: 'famixModel outputDirectory'
```

```
    classVariableNames: ''
    package: 'TypeScript-Exporters'
```

Ensuite, nous avons créer trois protocoles **initialization**, **accessing** et **exporting** dans cette classe pour ranger les méthodes de la classe.

Méthode 1 : initialize

Pour initialiser une variable "model" quand on crée une instance

```
initialize
    super initialize.
    model := nil
```

Méthode 2 : model:

Pour pouvoir assigner un modèle à l'exporteur.

```
model: aFamixTSModel
    model := aFamixTSModel
```

Méthode 3 : exportToCSV:

Méthode qui gère la logique principale de la classe : écrire les données dans un CSV avec NeoCSV.

```
exportToCSV: filename
    | classes writer file |
    classes := model allModelClasses.

    "Créer une référence au fichier et le supprimer s'il existe"
    file := filename asFileReference.
    file exists ifTrue: [ file delete ].

    file writeStreamDo: [ :stream |
        writer := NeoCSVWriter on: stream.

        "Écrire l'en-tête"
        writer nextPut: #('ClassName' 'NumberOfMethods' 'NumberOfAttributes'
'LinesOfCode').

        "Écrire chaque classe ligne par ligne"
        classes do: [ :class |
            writer nextPut: {
                class name.
                class numberOfMethods.
                class numberOfAttributes.
                class numberOfLinesOfCode
```

```
            }
        ]
    ]
```

Partie 4: Visualisation externe

Outil sélectionné: Python

Figure 1: Comparaison des métriques par classe

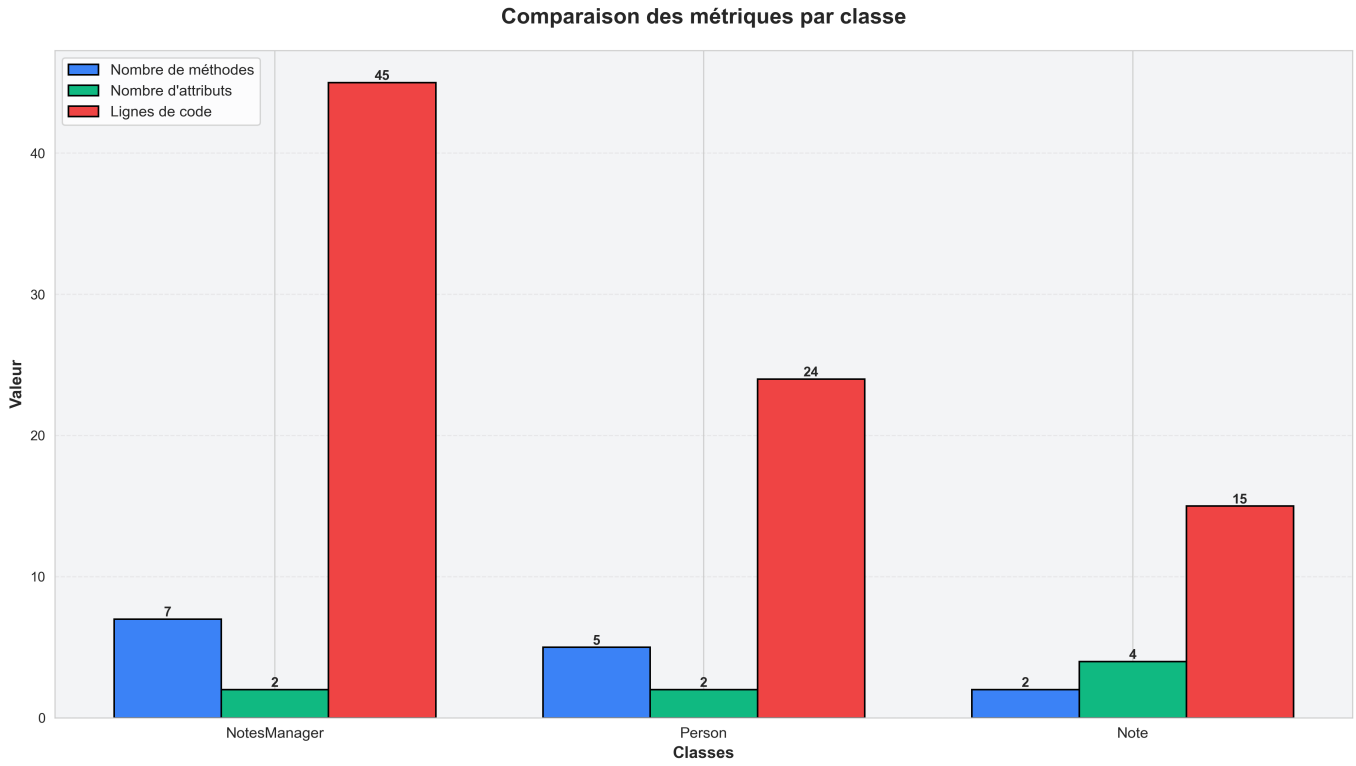


Figure 2: Heatmap

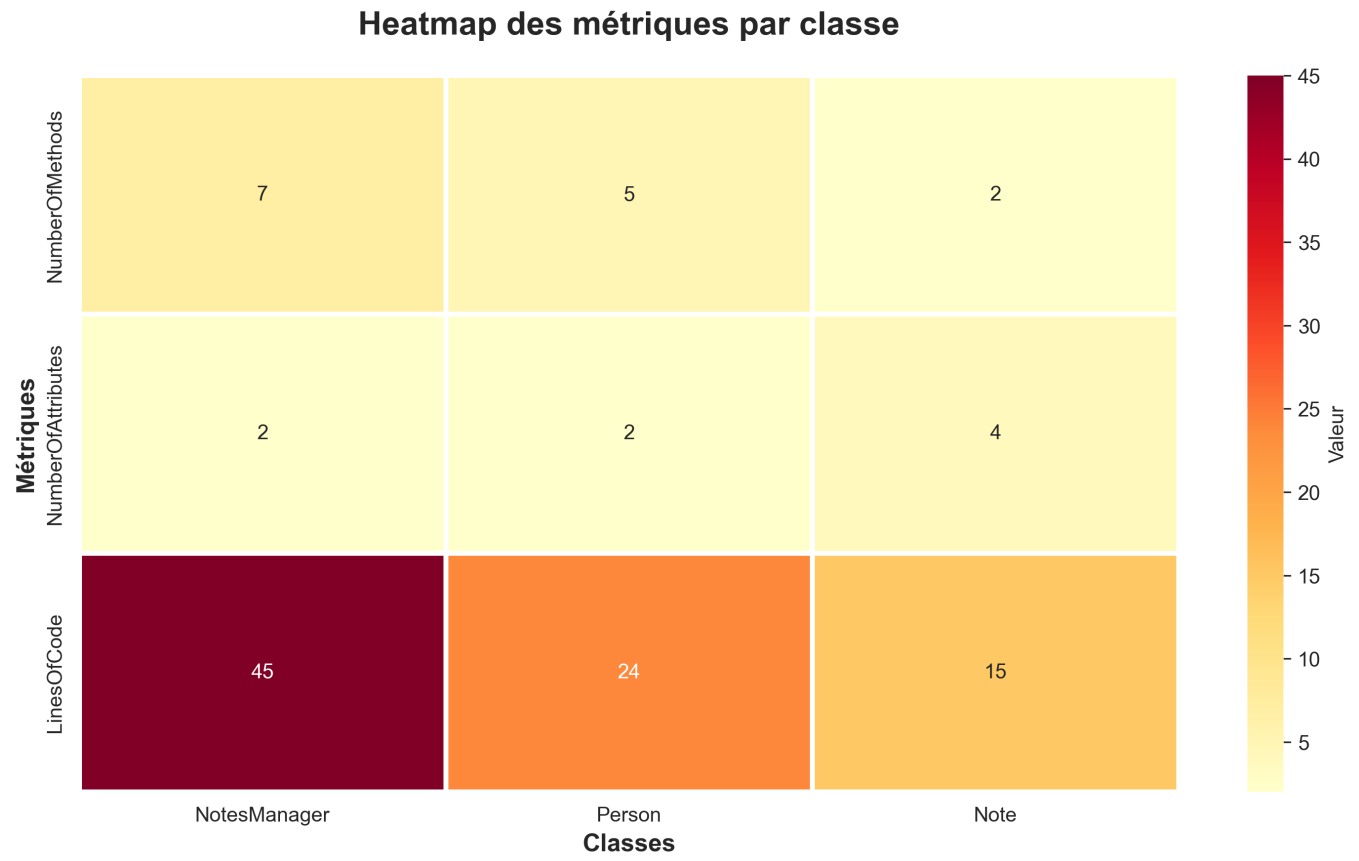
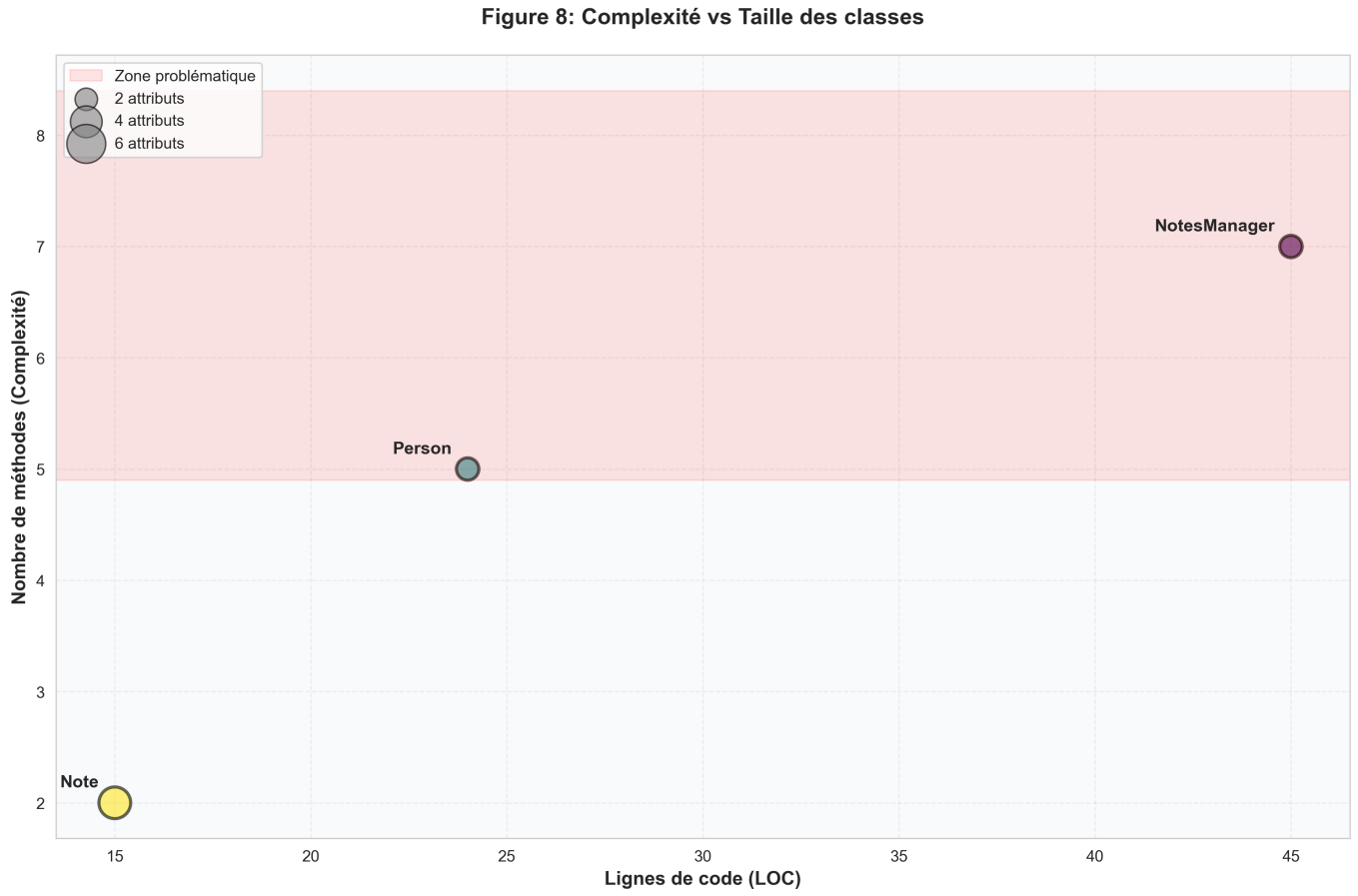


Figure 3: Complexité vs Taille des classes (Scatter plot)



Fichiers à remettre

1. **RAPPORT_TP1.pdf** - Ce rapport en format PDF

2. **dist/model.json** - Modèle Famix du projet TypeScript
3. **Dépôts GitHub** - https://github.com/mab001/tp1-notes-app

Références et ressources

- **Tutoriel ts2famix**: https://fuhrmanator.github.io/tuto-Famix-ts/
- **Documentation Famix**: http://Famix.org/
- **Roassal**: https://roassal.github.io/
- **NeoCSV Pharo**: https://github.com/svenvc/NeoCSV
- **Iceberg Pharo**: https://iceberg.githubusercontentload.com/
- **GitHub Copilot**: https://github.com/features/copilot
- **TypeScript**: https://www.typescriptlang.org/
