Projet CAI et TOO

2023-2024

Licence 3 Informatique - Semestre 5

Iban LEGINYORA

Table des matières :

1 - Objectif du projet :

Le projet consiste à charger (par Drag & Drop par exemple) des fichiers .dmn à visualiser puis charger des données d'entrée (par Drag & Drop par exemple) en format JSON pour l'évaluation FEEL. Le navigateur Web donne le résultat de l'évaluation voire les erreurs le cas échéant.

2 - Présentation :

Le projet est développé en *TypeScript* et utilise l'outil WebPack pour optimiser le chargement des pages dans le navigateur.
Le projet est prêt à l'emploi, l'ensemble des fichiers ont été compilés correctement. Vous pouvez l'essayer directement en ouvrant *index.html* et en utilisant les fichiers (*.dmn* et *.json*) mis à disposition dans le dossier *./test_files* [voir partie 8].

3 - Organisation du répertoire :

Le projet est composé de plusieurs éléments :

- **répertoire courant** :
  - index.html
  - tsconfig.json (configure l'exportation des fichiers .ts vers .js) [voir partie 5]
  - webpack.config.json (configure l'exportation du fichier main.js vers bundle.js) [voir partie 6]

- **./node_modules** :
  répertoire généré par npm, il contient les packages (bibliothèques ou modules) nécessaires pour le fonctionnement de l'application.

- **./ts** :
  - fichiers de type *.ts* servant au développement (c'est à cet endroit qu'on travaille sur le projet).
  - **./types** :
    - fichiers de type *.d.ts* (fichiers de déclaration TypeScript, il permettent à TypeScript de comprendre l'API d'un module)
    - DMN-JS.ts (déclare les types des données se trouvant dans la structure d'objets JavaScript)

- **./js** :
  dossier généré par npm (grâce à tsconfig.json), c'est le résultat de l'exportation des fichiers *.ts* (et *.d.ts*) du dossier **./ts**.

- **./dist** :
  - dossier généré automatiquement par webpack (grâce à webpack.config.json).
  - bundle.js (fichier principal généré par webpack et appelé dans le fichier index.html) [voir partie 6]

- **./styles** :
  dossier répertoriant les feuilles de style *CSS*.

- **./pictures** :
  dossier répertoriant les images.

- **./test_files** :
  jeu de tests permettant de tester différents fichiers dans le projet.

4 - Librairies, bibliothèques et framework utilisés :

Pour simplifier le processus de développement, certaines parties de l'affichage ont été gérées par la bibliothèque SweetAlert (pour l'affichage dynamique des messages) et par le framework Bootstrap (pour l'affichage des tableaux dans le cadre de ce projet).

> *SweetAlert* a été importée comme dépendance dans le projet, et utilise la version 9.17.4.
> *Bootstrap* a été chargé via un lien directement dans le fichier index.html.

Pour la **visualisation**, la librairie dmn-js a été choisie. Elle a été importée comme dépendance dans le projet, et utilise la version 14.7.1. Un fichier de déclaration TypeScript (*dmn-js.d.ts*) a dû être créé afin de permettre à TypeScript de comprendre l'API du module. De plus, il a fallu déclarer dans le fichier *tsconfig.json* le chemin vers le module :

```
"paths": {
  "dmn-js": ["node_modules/dmn-js"],
  "dmn-moddle": ["node_modules/dmn-moddle"]
},
```

capture d'extrait de code provenant du fichier tsconfig.json

Pour **transformer** le fichier *.dmn* en structure JavaScript, la librairie dmn-moddle a été utilisée. Le même problème que pour la librairie dmn-js a été rencontré, donc la même solution a été employée (voir la capture ci-dessus). Quant au fichier de déclaration il a été nommé *dmn-moddle.d.ts*.

Pour **l'évaluation** feel, la librairie feelin a été importée.

5 - Explication fichier tsconfig.json :

Ce fichier *tsconfig.json* configure les options du compilateur TypeScript pour un projet. Voici une explication des principales propriétés :

1. **"baseUrl": "." :** Définit le dossier de base pour la résolution des chemins relatifs. Dans ce cas, le dossier de base est le répertoire racine du projet. [voir partie 3]
2. **"module":** "ESNext" **:** Spécifie le format des modules utilisés dans le code source. Ici, il est configuré pour générer des modules compatibles avec ESNext (la version ECMAScript la plus récente).
3. **"moduleResolution":** "node" **:** Indique au compilateur d'utiliser la résolution de module Node.js pour trouver les modules.
4. **"noImplicitAny":** true **:** Active le mode strict pour détecter et signaler les cas où le type any est utilisé implicitement.
5. **"outDir":** "js" **:** Indique le dossier de sortie pour les fichiers JavaScript générés. [voir partie 3]
6. **"sourceMap":** true **:** Active la génération de fichiers source map, facilitant le débogage en liant les fichiers JavaScript générés aux fichiers TypeScript d'origine.
7. **"strict":** true **:** Active la vérification stricte.
8. **"target":** "ES6" **:** Spécifie la version ECMAScript cible pour la sortie JavaScript. Dans ce cas, il est configuré pour générer du code compatible avec ECMAScript 2015 (ES6).
9. **"esModuleInterop":** true **:** Facilite l'interopérabilité entre les modules CommonJS et les modules ES6.

10. **"typeRoots": ["./ts/types", "./node_modules/@types"] :** Indique les dossiers où TypeScript recherche les fichiers de définition de type (*.d.ts). [voir partie 3]
11. **"skipLibCheck": false :** Active la vérification des fichiers de définition de type dans les bibliothèques externes.
12. **"paths": { ... } :** Associe des alias de chemin à des chemins spécifiques, utiles pour simplifier les importations de modules. [voir partie 4].
13. **"include": ["ts/"] :** Spécifie les fichiers à inclure lors de la compilation. Dans ce cas, les fichiers dans les dossiers *./ts* seront inclus. [voir partie 3]

6 - Explication du fichier webpack.config.json :

Ce fichier de configuration Webpack (*webpack.config.js*) définit les paramètres de construction pour le projet. Voici une explication des principales parties :

1. **entry : './js/main.js' :** Spécifie le point d'entrée du projet. Le fichier principal est *main.js* situé dans le dossier *./js*. [voir partie 3]
2. **output :** Configure les paramètres de sortie, tels que le nom du fichier de sortie (*bundle.js*) et le chemin de sortie (*./dist*). Le fichier généré par cette configuration se trouvera dans le dossier *./dist*. [voir partie 3]
3. **resolve :** Définit les extensions de fichiers que Webpack doit prendre en compte lors de la résolution des modules. Cela inclut les fichiers avec les extensions *.ts* et *.js*.
4. **mode : 'development' :** Spécifie le mode de construction, soit 'development'. En mode développement, le code généré est plus lisible et inclut des informations de débogage.
5. **module et rules :** Configure les modules et les règles pour traiter différents types de fichiers.

7 - Présentation de la structure du code :

Le code a été divisé en 2 grandes parties :

Partie interface utilisateur : *main.ts* On déclare les différents événements du navigateur (*window.addEventListener*) permettant d'écouter les actions de l'utilisateur de l'application et d'afficher les résultats ou les erreurs.	
window.addEventListener("DOMContentLoaded", function () { … })	Événement lorsque la page est entièrement chargée.
fileInputDMN.addEventListener('change', async (event) => { … })	Événement lorsque la zone de dépôt du fichier *.dmn* a changée (permet de faire le drag and drop).
fileInputJSON.addEventListener('change', async (event) => { … })	Événement lorsque la zone de dépôt du fichier *.json* a changée (permet de faire le drag and drop).
btnEvaluation.addEventListener('click', async () => { … })	Événement lorsque le bouton Évaluer est cliqué.
btnClear.addEventListener('click', () => { … })	Événement lorsque le bouton Effacer est cliqué (Il est déclaré dans l'événement *fileInputDMN.addEventListener('change', …)*)

Partie traitement arrière plan: *fileImport.ts, fileDmn.ts, fileJson.ts*

Les classes *FileDmn* et *FileJson* héritent des attributs et méthodes de la classe *FileImport*.

FileImport._verificationFormat() a été surchargée dans les classes héritières afin de correspondre au mieux à l'extension souhaitée.

Result est une énumération et elle est utilisée afin de connaître l'état du fichier.
De plus, les messages informatifs (positifs ou non) sont retournés dans l'attribut *_message* et peut être récupéré dans la partie interface (*main.ts*) grâce au getter *get_message()*.

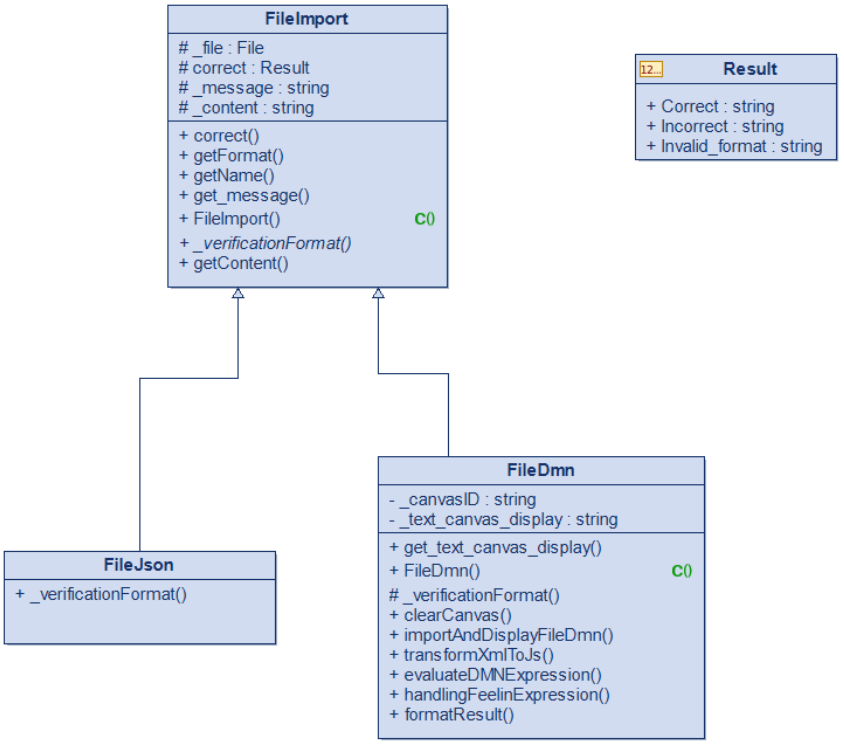


Diagramme de classe du projet

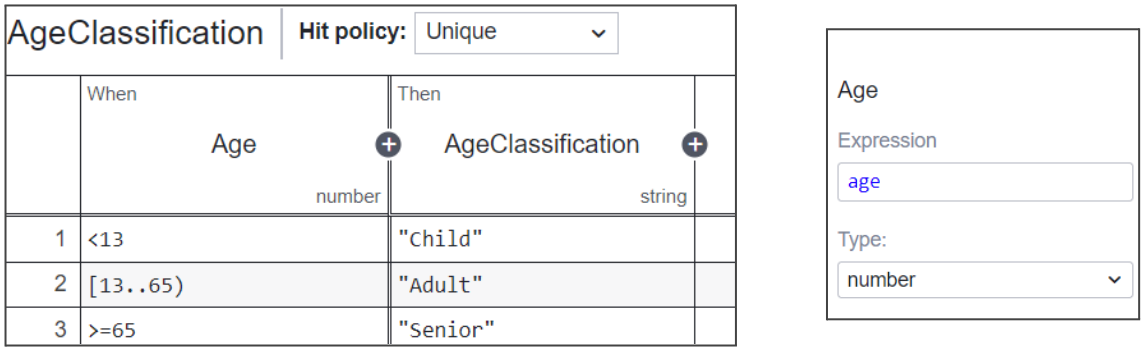
8 - Utilisation de l'application :

Différents fichiers avec leurs données d'entrée sont disponibles dans le dossier ./test_files. Des fichiers permettant de générer des erreurs et de vérifier si elles sont bien détectées sont aussi présents.

L'utilisateur doit charger (par drag and drop ou par l'explorateur de fichiers) le fichier dmn dans la zone "Fichier (dmn) visualisé" et le fichier json dans la zone "Fichier (json)".

Le fichier json doit fournir les données d'entrée correspondantes aux données d'entrée du diagram dmn. Le nom d'une variable d'entrée doit être le même que le nom de l'expression de l'input dans la colonne de la table traitant les données d'entrée.

Exemple :



Dans la table *AgeClassification*, la donnée input *Age* porte l'expression *age*.

Donc le fichier json doit être semblable à celui-là:

```
{
  "age": 84
}
```

AgeClassification_test.json

Une fois les fichiers fournis, l'utilisateur peut visualiser le diagramme, et l'évaluer (grâce au bouton).
Au moment de l'évaluation une expression feel est générée à partir du diagramme dans la méthode *fileDmn.handlingFeelinExpression()*, elle a pour syntaxe:

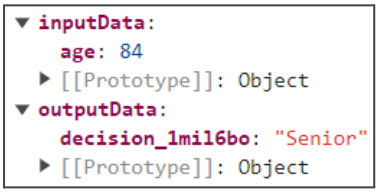
$$\text{if((var1 in conditions1) and (var2 in conditions2)) then output1 else "Erreur"}$$

par exemple, pour la table *AgeClassification*, l'expression est:

$$\text{if ((age in <13)) then "Child" else if ((age in [13..65))) then "Adult" else if ((age in >=65)) then "Senior" else "Erreur"}$$

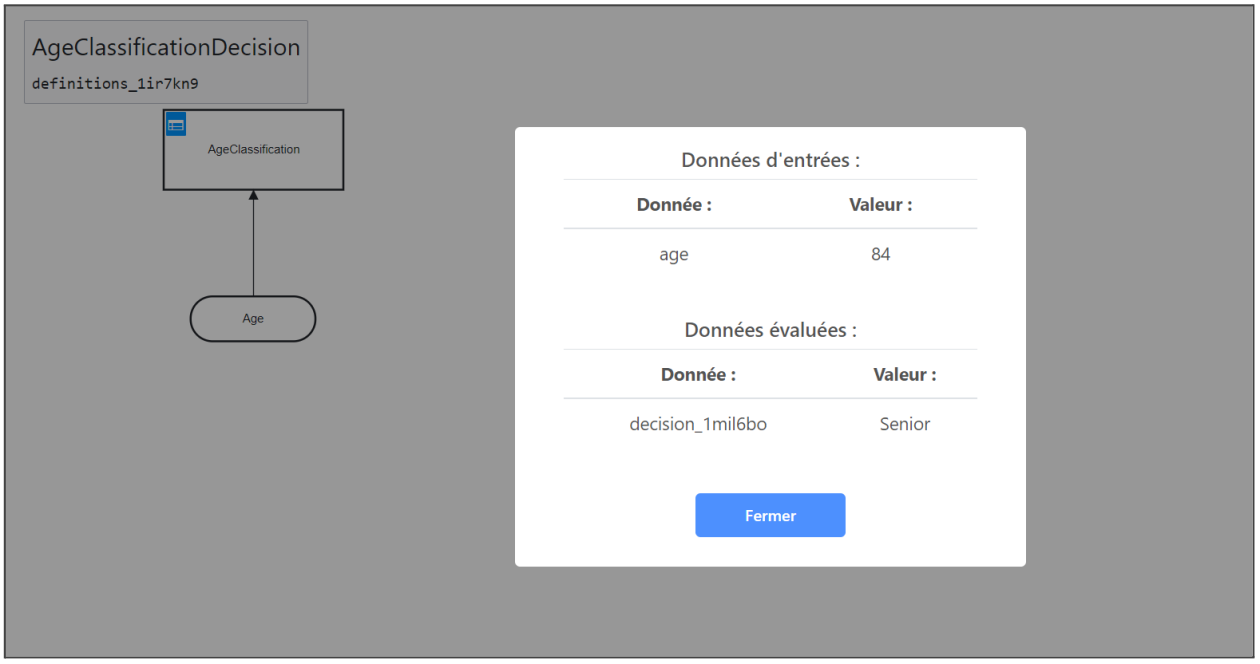
Le résultat est affiché sous forme de tableau dans un SweetAlert, il provient du résultat de l'expression qui renvoie deux objets: *inputData* et *outputData*.

reprenons l'exemple, voici le les objets retournés:



console du navigateur

et voici l'affichage de l'évaluation finale sur l'interface grâce à la fonction *fileDmn.formatResult()*:



Capture de l'interface avec la réponse finale