

# Manuel du projet (CAI-T00) 2023-2024

**Sujet : Application WEB pour la gestion  
des fichiers du type DMN**

---

Réalisé par :

MBARKI Mohammed  
ELOGRI Adnane

Année scolaire : 2023-2024

# Sommaire

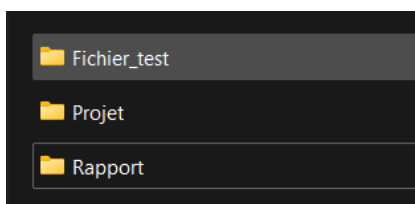
## Table of Contents

Introduction : .....	3
<b>CHAPITRE 1 : OUTILS ET CONCEPTS FONDAMENTAUX DU PROJET ....</b>	<b>4</b>
1. Introduction au Décision Model and Notation (DMN) : .....	5
2. Rôle et Utilité de la Librairie FEEL .....	5
<b>CHAPITRE 2 : MISE EN ŒUVRE ET MANIPULATION .....</b>	<b>6</b>
1. Explication d'utilisation de chaque composant dans le projet : .....	7
2. Les étapes qu'on a suivies pour créer et procéder le projet : .....	7
3. Rôles des Dépendances Spécifiques : .....	8
4. Les classes et les fichiers du projet : .....	11
<b>CHAPITRE 3 : TEST D'APPLICATION .....</b>	<b>12</b>
Conclusion : .....	16

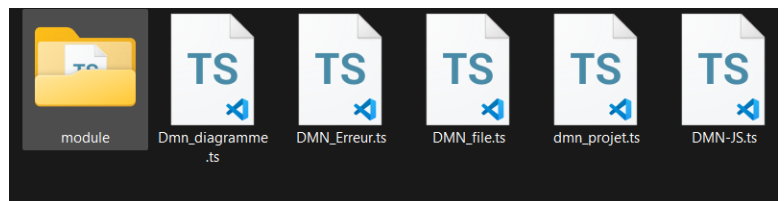
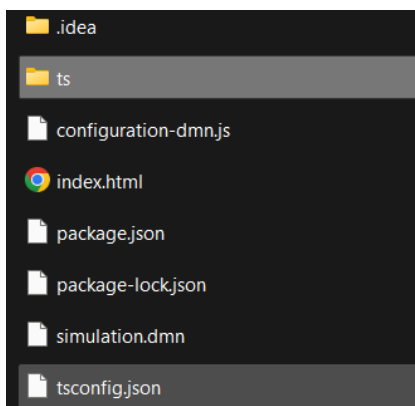
## Introduction :

Le projet actuel fonctionne et compile mais n'est pas déjà été compilé en JavaScript, ce qui signifie que vous ne pouvez exécuter le projet directement dans le navigateur sans compilation avant, et n'oubliez pas de exécuter la commande '**npm i**' afin d'installer tout les dépendances du projet , et de exécuter la commande suivante : '**npm run development**' pour compiler le projet.

Après avoir extrait le projet à partir de son fichier compressé, vous **trouverez 3 dossiers** le 1<sup>er</sup> '**Rapport**' contient le rapport( en format PDF et docx) et le 2eme il s'appelle **Projet** ,il contient le projet ,et le 3eme **Fichier\_test** ,il contient deux fichiers de tests , un de format JSON et l'autre de format DMN.



Dans le Dossier **Projet** vous trouverez le dossier ts qui contient tout les fichiers ts qu'on a écrit y compris le point d'entrée du projet : **dmn\_projet.ts**



# **CHAPITRE 1 :**

**OUTILS ET CONCEPTS  
FONDAMENTAUX DU PROJET**

## 1. Introduction au Décision Model and Notation (DMN) :

Le modèle de décision et la notation (DMN) se présente comme un standard de modélisation et de notation qui offre clarté et efficacité dans le domaine complexe de la prise de décision d'entreprise. Le DMN permet aux développeurs et aux spécialistes du domaine de travailler ensemble pour définir, modéliser et exécuter des décisions conformément aux normes du domaine. Cela facilite la représentation graphique des processus décisionnels, et donc à partir de cet outil Les décideurs peuvent visualiser la logique décisionnelle ce qui favorise une meilleure compréhension et une collaboration accrue entre les différents acteurs, tels que les spécialistes du domaine et les développeurs informatiques , Et donc on a décidé de choisir le langage Type Script pour gérer ce projet qui se base sur le DMN, et dans ce langage choisit y'a beaucoup des outils qui facilite la manipulation du DMN comme 'feelin'.

## 2. Rôle et Utilité de la Librairie FEEL

Le langage FEEL « Friendly Enough Expression Language », est utilisé pour manipuler le DMN, permet d'exprimer la logique décisionnelle avec une précision rigoureuse tout en restant accessible aux non-programmeurs. Grâce à sa syntaxe intuitive, FEEL surpasse la division traditionnelle entre la conception métier et l'implémentation technique, permettant une expression naturelle des règles de gestion. Les bibliothèques comme 'feelin' en TypeScript et JavaScript jouent un rôle crucial dans l'interprétation de ces règles, transformant les expressions FEEL en décisions concrètes et exécutables au sein des applications métier installées via **npm** dans notre projet.



Et donc dans les sections suivantes, on va décrire comment tout composant dans ce projet est utilisé et les raisons derrière notre choix de certaines méthodes.

Puisque c'est notre première projet en **TypeScript**, on a inspiré la structure de projet à partir de la structuration de la plupart des projets qui utilisent certaines configurations ,comme l'utilisation du **node** pour la structuration et de mettre chaque type de fichiers dans un dossier séparé avec les fichiers de configurations principaux comme **package.json** et autres dans la racine .

# **CHAPITRE 2 :**

**MISE EN ŒUVRE ET MANIPULATION**

## 1. Explication d'utilisation de chaque composant dans le projet :

Afin qu'on puisse ré-utiliser ou améliorer le projet de la part d'un autre développeur, ce développeur doit savoir quelques notions et règles importantes, comme :

-**Package.json** : Ce fichier est essentiel dans ce type de projets. Il contient des informations sur le projet, comme son nom, sa version, ses scripts, ses dépendances et on peut ajouter également des informations supplémentaires comme le nom des auteurs et une description. Lorsque on exécute `npm install` dans le terminal, npm crée ou met à jour ce fichier. Cependant, on pourra également créer le fichier `package.json` manuellement. Il doit être un fichier JSON valide et doit contenir au moins les champs `name` et `version`.

- **Package-lock.json** : Après qu'on configure notre fichier `package.json`, ce fichier va être automatiquement généré par npm lorsqu'on installe les dépendances nécessaires comme le `dmn` dans notre cas. Il contient des informations précises sur les versions exactes des dépendances installées dans le projet. Cela garantit qu'on utilise les mêmes versions des dépendances chaque fois qu'on installe le projet, ce qui peut aider à prévenir aux différents problèmes causés par des différences de versions. Et après ça, on aura le fichier,

- **tsconfig.json** : Ce fichier est généré automatiquement également, et il est spécifique à TypeScript. Il contient des options de configuration pour le compilateur TypeScript. On peut spécifier des choses comme la version de JavaScript à laquelle compiler le code TypeScript, les dossiers à inclure dans la compilation, et d'autres options de compilation et aussi la partie `'include'` qui indique quels fichiers doivent être inclus dans le processus de compilation.

- **node\_modules** : est un dossier créé lors de l'installation des dépendances avec npm. Il contient le code de toutes les dépendances du projet. Comme `@bpmn-io` et autres.

-**Fichiers de configuration Webpack** : Dans le projet courant on a Webpack nommé `dmn-js.config.js`. Ce fichier contient des options de configuration pour Webpack, y compris les règles pour charger différents types de fichiers, les plugins à utiliser et l'emplacement du bundle de sortie et il est configuré de certaine façon et avec des options qu'il décide comment le web pack va réagir.

## 2. Les étapes qu'on a suivies pour créer et procéder le projet :

Et comme étapes à suivre pour compléter la configuration du projet, on peut faire le suivant :

Au début, le projet sera vide, Et donc on aura besoin de créer le fichier **json** `'package.json'` à l'aide de la commande **`npm init`**, une fois qu'on a un fichier `package.json` dans notre projet, on pourra ajouter des dépendances à nos projet en utilisant la commande **`npm install`**. Par exemple, si on exécute **`npm install webpack --save-dev`**, npm installera Webpack dans le projet et ajoutera une entrée pour Webpack dans la section `'devDependencies'` du fichier `package.json` et en exécutant la commande **`npm install webpack-cli --save-dev`** le CLI de Webpack (Command Line Interface) va être s'installer.

Les scripts définis dans la section `'scripts'` de fichier `package.json` peuvent être exécutés avec la commande **`npm run`**. À titre d'exemple dans le projet courant, on a

un script nommé "development" qui peut être exécuté avec la commande ``npm run development``. Ce script exécute d'abord le compilateur TypeScript pour transpiler nos fichiers TypeScript en JavaScript, puis exécute Webpack pour regrouper les fichiers JavaScript.

Et Après on a créé les fichiers qu'on est besoin de, pour écrire et afficher le contenu de notre projet, les fichiers de type **TypeScript**, on l'ai mis dans le dossier **ts**, et ses copies compilées en **JavaScript** on l'ai mis dans le dossier **js** (cette structure de dossiers est définie dans le fichier **tsconfig.json**), et les fichiers de test de type DMN sont placées dans le dossier **test files**, Après on initialise le fichier html qui inclure le fichier JavaScript regroupé généré par Webpack, et après on peut tester notre projet et voir que tout va bien.

Et pour information, le **Webpack** est un outil essentiel dans ce projet qui sert de module bundler pour JavaScript. Il est utilisé pour compiler le code TypeScript en JavaScript, regrouper tous les fichiers JavaScript et leurs dépendances en un seul fichier, et gérer les fichiers CSS. Ces opérations sont essentielles pour préparer l'application à être exécutée dans le navigateur. La configuration de Webpack est définie dans le fichier **dmn-js.config.js**, qui spécifie comment Webpack doit traiter les différents types de fichiers et où il doit écrire le bundle de sortie.

**Mais avant, ce fichier package.json, sert à quoi ? pourquoi on doit l'ajouter pour une meilleurs experience ?**

Le package.json est le cœur d'un projet Node.js, jouant un rôle multi-facettes dans la gestion des dépendances, l'automatisation des tâches, le partage d'informations sur le projet, et la configuration des outils de développement. En centralisant toutes ces informations, il rend le processus de développement plus transparent et normalisé, ce qui est essentiel pour une collaboration efficace et une maintenance facile.

### 3. Rôles des Dépendances Spécifiques :

Dans le package.json, la section dependencies liste les librairies nécessaires pour que l'application fonctionne correctement. Voici un aperçu des rôles clés joués par certaines dépendances spécifiques à notre projet :

**-@bpmn-io/dmn-migrate** : Permet la migration des diagrammes DMN entre différentes versions de la spécification DMN.

**-dmn-js** : Utilisé pour la visualisation et l'interaction avec les diagrammes DMN directement dans le navigateur web.

**-dmn-moddle** : Fournit la capacité de manipuler les diagrammes DMN comme des structures de données JavaScript.

**imicros-feel-interpreter** : Interprète le langage FEEL, utilisé pour définir la logique des décisions dans les diagrammes DMN.

**-process** : Module offrant des fonctionnalités pour interagir avec le processus Node.js courant.

**-util** : Module contenant des outils divers pour aider avec des tâches telles que l'inspection d'objets et l'héritage.

Ces modules sont intégrés au projet via **npm install**, qui est initié après la création



de package.json avec npm init.

### Configuration du Compilateur TypeScript avec tsconfig.json

Le tsconfig.json est un fichier de configuration pour le compilateur TypeScript, qui détaille comment le code TypeScript doit être transpilé en JavaScript. Ce fichier peut être généré avec la commande **tsc --init**, créant une configuration standard, ou il peut être personnalisé pour répondre aux besoins spécifiques.

```
1  {
2    "compilerOptions": {
3      "baseUrl": ".",
4      // "esModuleInterop": true,
5      "module": "ES6",
6      "moduleResolution": "node",
7      "outDir": "js",
8      "skipLibCheck": true,
9      "sourceMap": true,
10     "strict": true,
11     "target": "ES6",
12   },
13   "include": [
14     "ts/**/*"
15   ]
16 }
```

Enfin, le fichier **dmn-js.config.js** est crucial pour la construction et le bundling de l'application. Webpack, en se référant à ce fichier, comprend comment traiter les fichiers TypeScript et CSS, détermine le point d'entrée de l'application comme dans notre cas c'était le **'./ts/dmn\_projet.ts'**, et où placer le fichier JavaScript compilé. La suppression de ce fichier entraînerait une absence de directives pour Webpack, résultant en des erreurs de construction. Si jamais ce fichier est manquant, il est impératif de le restaurer ou de le recréer avec les configurations nécessaires pour maintenir le fonctionnement fluide du processus de développement.

#### **Note :**

On mentionne qu'on a utilisé le sweetAlert2 comme bibliothèque javascript (typescript également), vue en cours pour donner une belle vue à l'utilisateur. Et donc après on peut exécuter notre application et d'explorer ce qu'il fait.

Voici des exemples de l'implementation qu'on a fait avec une petite explication:

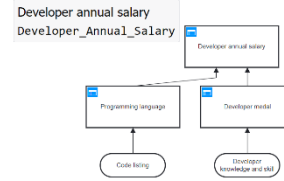
### Gestion du DMN

Glissez un fichier DMN ou JSON n'importe ou

Rechercher le fichier

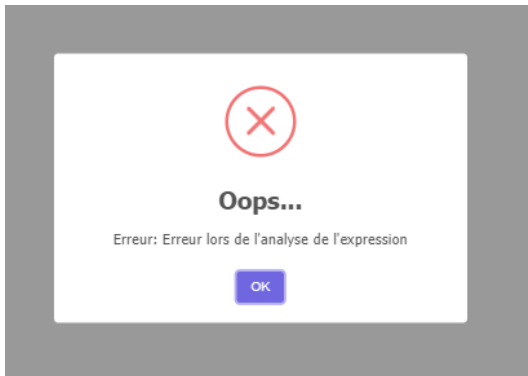
### Gestion du DMN

Glissez un fichier DMN ou JSON n'importe ou



Rechercher le fichier

BPMM 10



### Gestion du DMN

Glissez un fichier DMN ou JSON n'importe ou

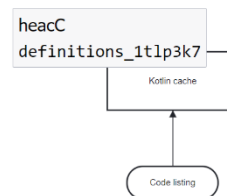


Figure : si le fichier inserer n'est pas dans la bonne format

### Autres informations importantes:

Comment le fichier dmn est afficher normalement dans le navigateur avec une belle interface graphique et un peu de code ?

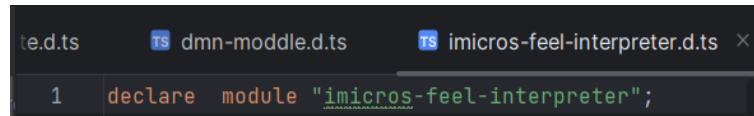
- ⇒ Au plus des étapes d'installation qu'on a fait précédement , on a fait la bibliothèque dmn-js et ses classes et fonctions qu'ils nous a permettent de faire cette operation , on a importer **Set\_current\_diagram** depuis **DMN-JS** qui est un module TypeScript défini dans notre projet dans le fichier DMN-JS.ts , et à partie de cette instanciation d'objet DmnJS de cette façon , on a pus la faire :

```
DMiNer.Viewer = new DmnJS({
  container: document.getElementById("canvas")
});
```

Maintenant on décrie une autre chose très importante , qui est : c'est quoi l'interet de la partie module présente dans le dossier ts ? et est-ce que le programme va marcher après si on a l'enlever ?

- ⇒ Dans le projet courant , on a utilisé des bibliotheques qu'ils ne sont pas

directement intégrées dans le TypeScript mais plutôt en JavaScript ,pour régler ce problème on aurait besoin de créer nos propres fichiers de déclaration de module (fichiers '.d.ts') pour fournir des informations de type , le nom de ce fichier en fonction du nom du module JavaScript , comme le module qui a le nom **imicros-feel-interpreter** on nomme le fichier : **imicros-feel-interpreter.d.ts** et à l'intérieure de ce dernier on declare le module à l'aide de cette ligne :



```
1 declare module "imicros-feel-interpreter";
```

Et on a fait la même chose pour les autres fichiers de type '.d.ts' .

Si on a enlevé ces fichiers le TypeScript ne sera plus en mesure de comprendre les types utilisés par ces modules externes ce qui va entraîner des erreurs de compilation.

Pour les fichiers Ts qu'ils contiennent les fonctions utilisées et le fonctionnement générale, on va vous fournir des explications concises mais rigoureuses pour les parties clés.

#### 4. Les classes et les fichiers du projet :

##### -Pour le fichier **Dmn\_diagramme.ts** :

Cette classe représente un diagramme DMN. Elle contient des méthodes pour récupérer le diagramme DMN, afficher le diagramme DMN et évaluer le diagramme DMN. Elle contient également des propriétés pour stocker l'interprète, le diagramme, le XML du diagramme et le JSON de l'évaluation

##### -Pour le fichier **Dmn\_projet.ts** :

le fichier **dmn\_projet.ts** gère l'interaction de l'utilisateur avec l'application, notamment le chargement des fichiers DMN et l'affichage des diagrammes **DMN**.

##### -Pour le fichier **Dmn\_erreur.ts** :

Cette classe est utilisée pour gérer les erreurs dans l'application. Elle étend la classe Error intégrée de JavaScript et ajoute une méthode pour afficher l'erreur à l'aide de **sweetalert2**.

##### -Pour le fichier **Dmn\_file.ts** :

Cette classe gère la lecture des fichiers DMN. Elle contient des méthodes pour initialiser le lecteur de fichiers, gérer le chargement des fichiers, vérifier le type de fichier et déclencher des événements personnalisés.

##### -Pour les fichiers du dossier « Module » :

Pour utiliser certaines bibliothèques JavaScript qui n'ont pas de définitions de types TypeScript, il faut déclarer un module pour chacune de ces bibliothèques dans un fichier .d.ts. Cela permet à TypeScript de comprendre comment interagir avec ces bibliothèques. Dans notre cas, nous avons déclaré des modules pour 'dmn-js' et '@bpmn-io/dmn-migrate', entre autres, dans des fichiers .d.ts. Cela signifie que nous pouvons maintenant utiliser ces bibliothèques dans notre code TypeScript sans erreurs de compilation

# **CHAPITRE 3 :**

## **TEST D'APPLICATION**

## Page principale :

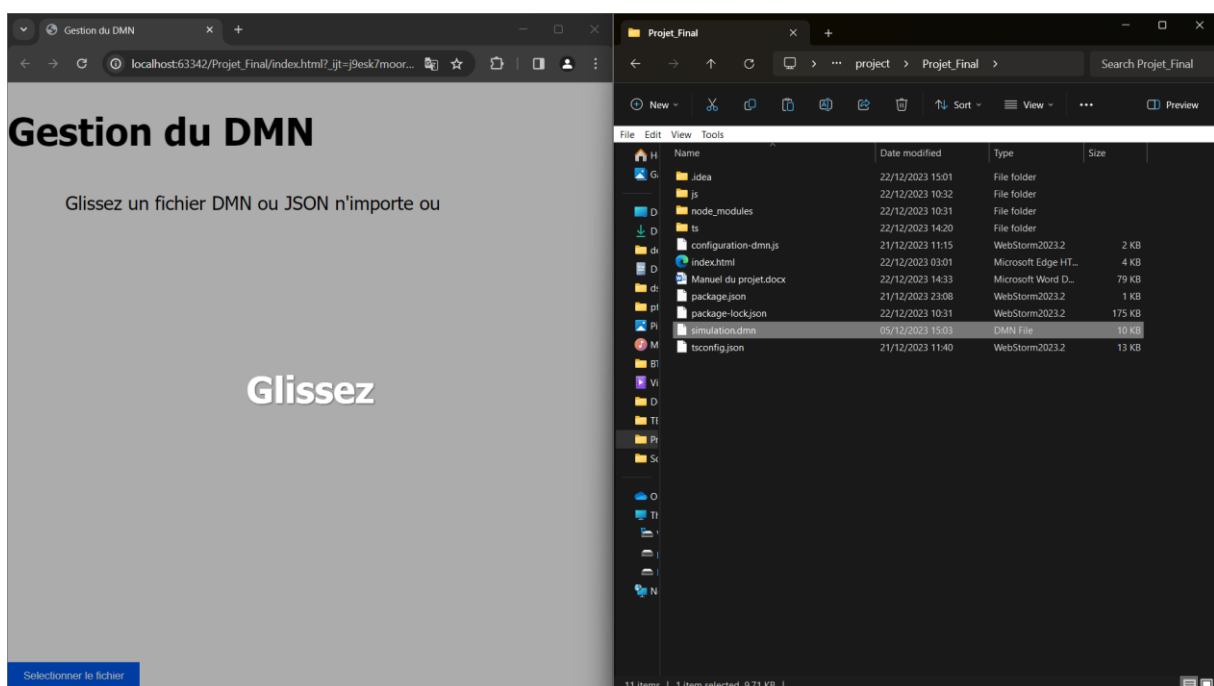
La page d'accueil du projet offre une interface utilisateur intuitive pour importer des fichiers DMN ou JSON. L'utilisateur peut importer un fichier en utilisant le bouton 'Sélectionner le fichier', qui ouvre une boîte de dialogue permettant de naviguer dans le système de fichiers local. Alternativement, l'utilisateur peut utiliser la fonctionnalité de glisser-déposer pour importer un fichier en le glissant simplement dans la zone délimitée, comme indiqué dans l'image ci-dessous.

### Gestion du DMN

Glissez un fichier DMN ou JSON n'importe où

Sélectionner le fichier

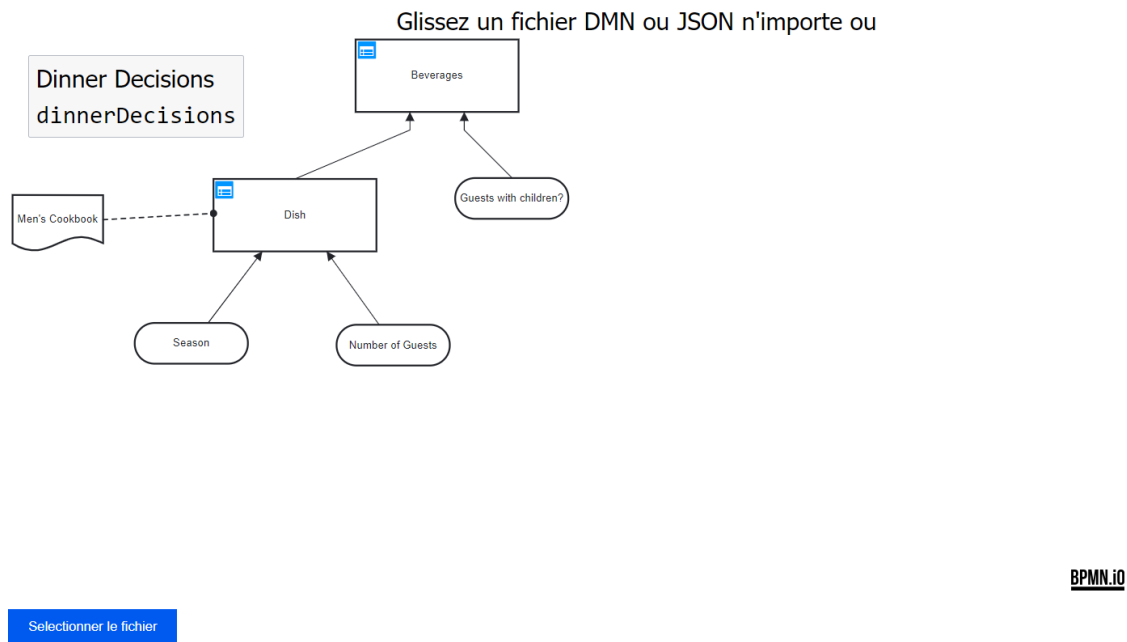
Voici l'image qui décrit l'opération drag&drop.



## Affichage du DMN :

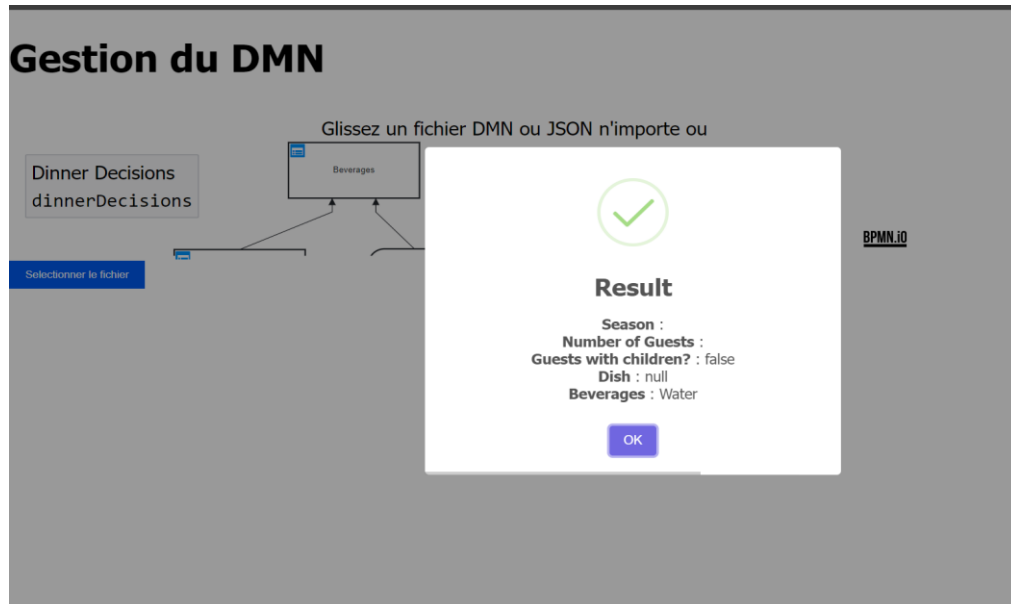
Après l'importation du fichier DMN, le diagramme DMN est affiché dans le navigateur grâce à la bibliothèque **dmn-js**. Cette bibliothèque fournit une interface utilisateur riche et interactive pour visualiser et interagir avec les diagrammes DMN. L'utilisateur peut explorer le diagramme en détail, y compris les différentes règles de décision, les entrées et les sorties. voici l'image suivante :

## Gestion du DMN



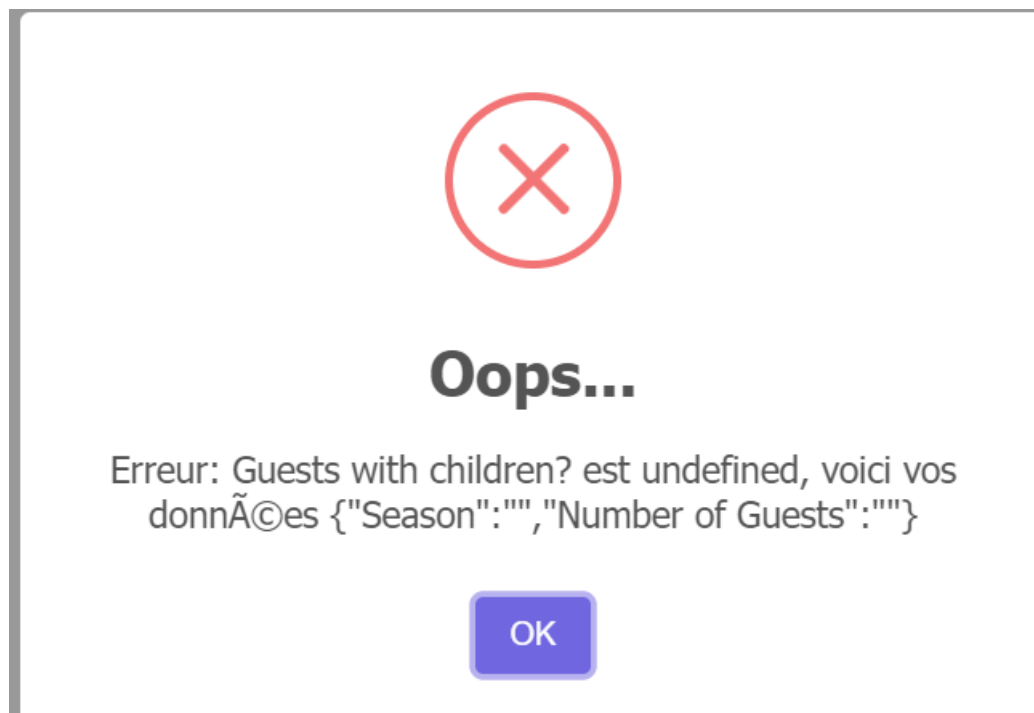
### Cas de succès :

Ce message est affiché lorsqu'on insère des données correctes, et on peut insérer même avec beaucoup de données et même le json et en forme array.



### Cas d'erreur :

Ce message est affiché lorsqu'on insère un fichier de format invalide.



## Conclusion :

Ce projet illustre efficacement l'utilisation de TypeScript dans le développement d'applications modernes. TypeScript, en tant que surcouche typée de JavaScript, apporte une robustesse supplémentaire au code, facilitant la détection précoce des erreurs, l'amélioration de l'autocomplétion et la lisibilité du code.

Dans ce projet, TypeScript est utilisé en conjonction avec Webpack, un outil de regroupement de modules, pour gérer et optimiser le code de l'application. Le fichier `package.json` du projet démontre une gestion efficace des dépendances et des scripts, ce qui facilite le processus de développement et de déploiement.

De plus, l'utilisation de `ts-loader` dans le fichier de configuration Webpack permet de transpiler le code TypeScript en JavaScript, ce qui rend le code compatible avec les navigateurs qui ne supportent pas directement TypeScript. Cela démontre une bonne pratique pour le développement d'applications TypeScript destinées à être déployées sur le web.

En conclusion, ce projet est un bon exemple de la manière dont surtout TypeScript et javascript peut être utilisé pour améliorer la qualité du code et l'efficacité du développement. Il démontre les avantages de l'utilisation de TypeScript, notamment la sécurité des types, l'amélioration de l'outillage et la meilleure lisibilité du code.

Je vous remercie mon professeur pour votre soutien et vos conseils tout au long de ce projet. Votre expertise et votre dévouement ont été inestimables pour la réussite de ce projet. Merci pour votre temps.



