Université de Pau et des pays de l'Adour

Rapport projet DMN

Author:
Corentin Calmels

Décembre 2023

Contents

1 Introduction

L'objectif de ce projet était de pouvoir afficher et évaluer un fichier DMN. Ce rapport explique l'architecture du projet ainsi que les choix de conception, les fonctionnalités principales et les librairies utilisées pour atteindre ces objectifs. Le code source est également disponible ici C21111222/DMN_project.

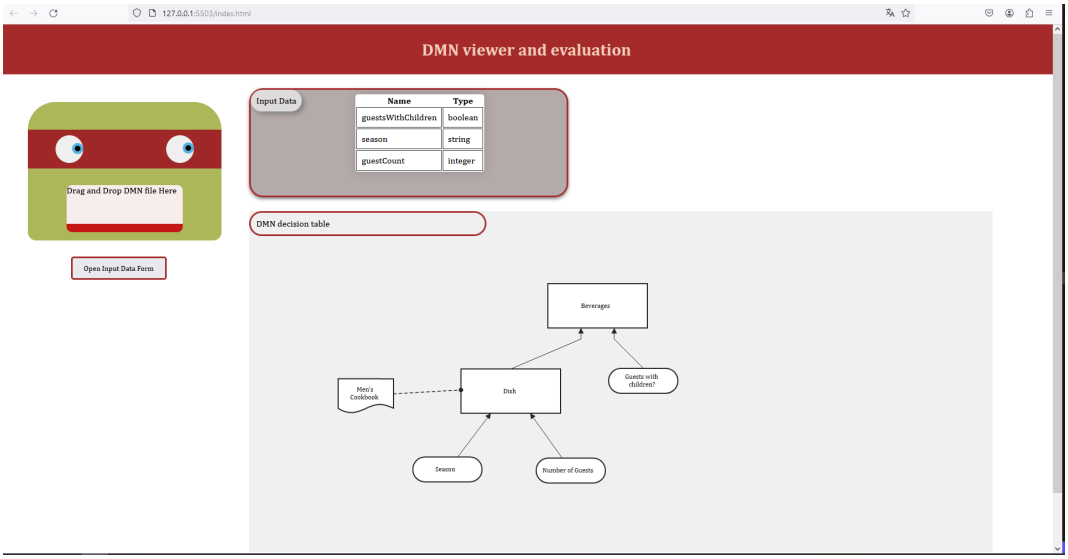

Figure 1: Interface de l'application

2 Organisation du code

2.1 Structure des dossiers

Le projet est constitué de plusieurs dossiers et sous-dossiers, suivant la structure suivante :

- `ts/`: Dossier contenant tous les fichiers TypeScript.
  - `animation/`: Fonctions d'animation de la page (par exemple, `eyes_movements`)
  - `models/`: Classes de modèles de données (par exemple, `DMNModel`)
  - `utils/`: Fonctions et classes utilitaires (par exemple, `migrate_DMN`)
  - `view/`: Classes et fonctions pour l'affichage des données (par exemple, `DataDisplay`)
  - `main.ts`: Fichier principal du projet, importation des différentes fonctions et données, prise en charge des fichiers par glisser-déposer.
- `css/`: Dossier pour les fichiers CSS
- `@types/`: Dossier pour la déclaration de types TypeScript nécessaire au build
- `DMN_files`: Dossier contenant les fichiers DMN d'exemple

2.2 Convention de nommage

- **Fichier**: snake_case (par exemple, `decision_table.ts`).
- **Classes & Interfaces**: PascalCase (par exemple, `DMNModel`).
- **Méthodes & Variables**: camelCase (par exemple, `defineInputData`).
- **Constantes**: UPPER_SNAKE_CASE (par exemple, `MAX_TIMEOUT`).

2.3 Détail des classes et fonctions

Le détail des classes, leurs méthodes ainsi que les fonctions du projet sont consultables dans la documentation. Pour cela, se référer à la section associée dans "Installation et utilisation".

3 Synergie du code

La synergie entre les différents composants du code est essentielle pour assurer une expérience utilisateur satisfaisante et une maintenance aisée du projet. Voici comment les différentes parties du code interagissent :

- **Interaction entre les modèles et la vue** : Les classes dans `ts/models/` définissent la structure des données DMN et leur logique métier. Lorsqu'un fichier DMN est chargé, ces modèles sont instanciés et peuplés avec les données du fichier. Ensuite, `ts/view/` prend ces modèles et les rend visuellement dans le navigateur, permettant aux utilisateurs d'interagir avec les données de manière intuitive.
- **Utilitaires et modèles** : Les fonctions dans `ts/utils/` offrent plusieurs services communs comme la migration des fichiers DMN (`migrate_DMN`) qui est utilisée par les classes de modèles pour s'assurer que les fichiers DMN sont dans le bon format avant d'être traités.
- **Animations et interactions utilisateur** : Le dossier `ts/animation/` contient des scripts qui améliorent l'interface utilisateur avec des animations.
- **Gestion des événements** : `main.ts` agit comme le chef d'orchestre, en écoutant les événements de glisser-déposer et en coordonnant les actions entre les modèles, les vues et les utilitaires. Il s'assure que les fichiers DMN sont correctement chargés et que les données sont affichées sans délai.

Cette synergie est conçue pour être modulaire, chaque partie du code ayant une responsabilité claire, ce qui facilite les mises à jour et l'ajout de nouvelles fonctionnalités.

4 Librairies et Frameworks

Le projet se base sur plusieurs librairies pour fonctionner :

- **DMN-JS**: Regroupe des objets JavaScript pour "lire" les fichiers DMN.
- **SweetAlert2**: Permet d'afficher simplement des notifications.
- **Xmldom**: Permet de parser des fichiers XML.
- **DMN-Migrate**: J'ai utilisé cette librairie (lien GitHub) en l'adaptant en TypeScript, permettant la migration de fichiers DMN en version 1.3 à partir de versions antérieures.
- **Feelin**: Permet d'évaluer les expressions du langage FEEL, qui correspondent aux règles dans un fichier DMN.
- **TypeDoc**: Permet de générer une documentation.

5 Fonctionnement général

Le principe de l'application est simple : on fait glisser et déposer un fichier DMN. On peut ainsi le visualiser et l'évaluer. Ensuite, on peut observer les données en sortie.

5.1 Fonctionnalités annexes

- **Migration**: Grâce à la librairie DMN-Migrate, l'application permet de prendre en charge les fichiers DMN en versions 1.1 et 1.2.
- **Visualisation des sous-tables**: Lorsque le fichier DMN contient plusieurs décisions, l'affichage correspond à l'ensemble de celles-ci. Pour visualiser en détail les sous-tables, on peut cliquer sur l'objet les représentant dans le diagramme.
- **Formulaire**: Pour évaluer un fichier DMN, on peut au choix déposer un fichier JSON contenant les données requises ou utiliser le formulaire généré automatiquement en cliquant sur "Open Input Data Form".
- **Documentation**: La génération de la documentation de chaque fichier est disponible dans la section "Documentation".

6 Installation et utilisation

6.1 Prérequis

- **npm (Node Package Manager)**: Le gestionnaire de paquets de Node.js est nécessaire pour le téléchargement des dépendances et la construction de l'application.
- **Live Server**: Un serveur local est nécessaire pour ouvrir le fichier HTML du projet.

6.2 Build

1. (depuis GitHub uniquement) Clonez le dépôt.
2. Accédez au répertoire du projet.
3. Installez les dépendances nécessaires en exécutant `npm install`.
4. Construisez le projet en exécutant `npm run build`.

6.3 Utilisation

1. Ouvrez index.html avec un serveur local dans un navigateur web moderne.
2. Glissez et déposez un fichier DMN dans la zone prévue.
3. Consultez la table de décision et les données d'entrée.
4. Évaluez la logique de décision en fournissant un fichier JSON ou en utilisant le formulaire fourni.
5. Observez la sortie.

6.4 Documentation

1. Accédez au répertoire du projet.
2. Exécutez `npm run doc`.
3. Ouvrez `docs/index.html` dans un navigateur web.

7 Limites

Plusieurs points sont à améliorer dans ce projet :

- Gestion des erreurs incomplète.
- Prise en charge des fichiers DMN aberrants, par exemple dans un fichier comportant plusieurs décisions non reliées (comme le rebooking de vol notamment).

- Robustesse du code.