



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

BACHELOR PROJECT

Rapport de projet

Transformation de scénarios
multimédias modélisés avec
des réseaux de Pétri, en
application WEB exploitant
la combinaison HTML-CSS-JS

Dominique Roduit

dominique.roduit@epfl.ch

12.01.2018

TABLE DES MATIERES

1. Informations sur le projet.....	3
2. Cahier des charges	3
2.1. Description du projet	3
3. Technologies utilisés	3
4. Interface utilisateur	4
5. Utiliser le projet	6
5.1. Installations requises	6
5.2. Description des fichiers et dossiers du projet	6
5.2.1. Fichiers et dossiers générés.....	8
5.3. Fichier JSON : construction et traitement des données	8
5.3.1. Ordre et nommage des sprites.....	8
5.3.2. Sprites enfants, ordre	8
5.3.3. Traitement des erreurs	9
5.3.4. Vues	9
5.3.5. Scènes	9
5.3.6. Sprites.....	9
5.3.7. Transitions	10
5.3.8. Mises en pages avancées, compositions complexes	11
5.3.9. Dépendances, ajout de librairies.....	11
5.4. Rôle et fonctionnement du fichier STAMP	12
5.4.1. Pourquoi avoir un 2 ^e fichier détaché du fichier JSON ?	12
5.5. Mode debug	13
5.5.1. Mode normal vs. debug	13
5.5.2. Génération d'un nouveau STAMP	13
5.5.3. Génération et aperçu accélérés.....	13
5.6. Activer / Désactiver la demande de mot de passe	14
5.7. fichiers multimédias de l'utilisateur.....	14
5.8. Compatibilité smartphones / tablettes	14
6. Liste des mots clés JSON traités (API)	15

1. INFORMATIONS SUR LE PROJET

Etudiant

Nom	Roduit	Prénom	Dominique
Adresse	Rue de l'île 8	NPA, Localité	3979 Grône
Adresse E-mail	dominique.rodut@epfl.ch	N° Tél.	+41 79 861 01 60

Superviseur du projet

Nom	Eytan	Prénom	Zysman
Adresse E-mail	eytan.zysman@epfl.ch		

Dates de réalisation

✓ 02.10.2017 – 12.01.2018

2. CAHIER DES CHARGES

2.1. DESCRIPTION DU PROJET

Les réseaux de Pétri permettent de modéliser un nombre infini d'applications très diverses. De manière basique, ce sont des schémas composés de 4 éléments principaux : Token, Places, Transitions et arcs :

- Token ou « Sprites » : objets définis par des propriétés
- Places, activités ou « scenes » : contenant un ou plusieurs « sprite »
- Transitions : gère les « events » pour passer d'une activité à une autre
- Arcs : reliant les places aux transitions.

Un logiciel de modélisation de réseaux de Pétri pour générer des fichiers en représentant une description au format JSON est déjà existant. Nous voulons ici utiliser le JSON généré, correspondant à la description d'applications WEB, et ainsi concevoir une prochaine étape, qui transformera le réseau de Pétri en une application WEB fonctionnelle, exploitant la combinaison HTML-CSS-JS.

3. TECHNOLOGIES UTILISÉS

La navigation se fait en JavaScript et seul des pages HTML/CSS sont générées, PHP n'est pas utilisé dans le résultat généré. Le moteur de génération des pages et de « passage » des fichiers JSON cependant est écrit intégralement en PHP et un semblant de JQuery est utilisé pour produire une interface utilisateur conviviale.



4. INTERFACE UTILISATEUR

En mode utilisateur (non debug), l'application commence par demander le réseau de pétéri au format JSON tel que généré par l'application de schématisation du réseau.



Petri > Website

1. Sélectionnez votre réseau de Pétri au format JSON

Choisissez un fichier

Dominique Roduit - EPFL © 2018

Si le fichier fourni n'est pas un fichier JSON ou fait plus de 10 Mo (ce qui devrait être amplement suffisant même pour un fichier décrivant un énorme site web), il n'est pas accepté et un message d'erreur y est affiché. Au contraire, si le fichier fourni répond à ces exigences, l'interface passe à l'étape suivante en demandant le fichier STAMP contenant l'emprunte numérique et le mot de passe pour convertir ce réseau en une application web.



Petri > Website

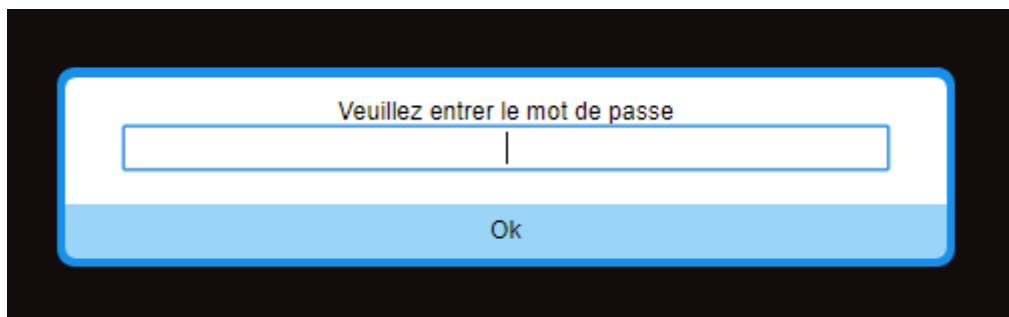
Réseau de Pétri importé avec succès !

2. Sélectionnez l'emprunte numérique STAMP

Choisissez un fichier

Dominique Roduit - EPFL © 2018

Si l'interface est configurée de manière à ce que le mot de passe soit demandé pour convertir les fichiers JSON, une fenêtre apparaît où l'utilisateur peut y entrer son mot de passe. Si le mot de passe est erroné, l'utilisateur en est averti, sinon la fenêtre disparaît et la génération est lancée. Si le mot de passe n'est pas requis dans la configuration, cette étape est ignorée et l'application web est directement générée.



Veuillez entrer le mot de passe

Ok

Une fois la génération terminée, l'utilisateur peut voir le résultat de son site web décrit par le réseau de pétéri en direct. Il peut démarrer son site par la page d'index si celle-ci a été spécifiée, ou parcourir ses scènes/vues les unes après les autres pour vérifier que tout est bien comme il l'a décrit par son réseau de pétéri. Bien entendu, les fichiers multimédias locaux qu'il aurait inclus ne sont pas

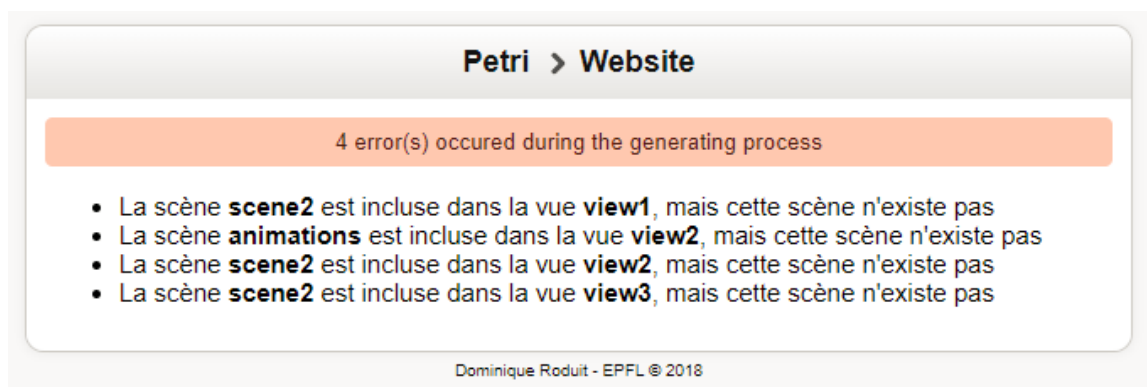
affichés ici. L'utilisateur peut choisir d'exporter son application web au format zip, ou de convertir un autre réseau de pétri, ce qui le ramènera à la toute première fenêtre décrite ci-dessus.



Lorsque l'utilisateur modifie le fichier JSON généré par l'application de schématisation, l'interface le détecte et informe l'utilisateur que ce fichier n'est pas l'original généré. Seul un utilisateur expérimenté connaissant la façon exacte de générer l'emprunte numérique de son fichier pourra éviter ce message en générant l'empreinte de son fichier modifié lui-même. Comme l'empreinte est le hash sha512 du fichier, haché à la façon HMAC en utilisant pour clé secrète le mot de passe haché, un utilisateur capable de modifier son empreinte peut donc facilement également modifier le mot de passe.



En cas d'erreurs lors du processus de traitement du fichier JSON, celles-ci seront affichées à l'utilisateur comme sur la capture ci-dessous :



5. UTILISER LE PROJET

5.1. INSTALLATIONS REQUISES

Pour le debug et développement : installer un petit serveur web tel que XAMPP server. Aucune base de données n'est utilisée, le serveur Apache suffit à faire tourner l'application.

Pour la production : poser les fichiers sur un serveur web via ftp.

***Note** : Sur linux et en production, il est possible que certains droits doivent être donnés sur les différents dossiers ou les fichiers utilisateurs et les fichiers générés sont stockés. Si les dossiers n'ont pas les droits nécessaires à recevoir les fichiers, l'application ne fonctionnera pas.*

5.2. DESCRIPTION DES FICHIERS ET DOSSIERS DU PROJET

Tous les fichiers du projet sont largement commentés à la manière d'une Javadoc pour permettre aux futurs développeurs de se retrouver parmi des classes composées d'un nombre assez conséquent de méthodes et d'attributs.

La hiérarchie des répertoires du projet ainsi que les rôles de chacun de leurs fichiers sont décrits ci-dessous :

- **index.php** : Point de départ de l'application. L'index ne contient que l'enveloppe de l'application. Des inclusions, asynchrones en JQuery ou directes en php sont faites pour afficher les fragments voulu dans l'enveloppe. L'index détermine quand afficher quel fragment. Il permet aussi d'activer/désactiver le mode debug et la demande de mot de passe (voir les sections « mode debug » et « demande de mot de passe »).
- **header.inc.php** : Ce fichier a pour unique but d'inclure toutes les dépendances nécessaires au fonctionnement de l'interface utilisateur et du « moteur » de génération des pages. Il inclue en outre tous les objets et fonctions utiles de l'application. Lorsqu'un appel est fait de manière asynchrone en JavaScript, ce fichier doit être inclus ou des dépendances seront manquantes et des erreurs seront générées.
- **parser.php** : C'est le parseur de l'application. Il est isolé du reste et son travail est de travailler sur les objets reçus du fichier JSON pour les interpréter et les transformer en objets PHP. Le résultat final du parseur est ensuite traité dans le fichier « processing.php ».
- **processing.php** : Lis le fichier JSON décrivant le réseau de pétri, invoque le parseur « parser.php » en lui fournissant le contenu du fichier sous forme de tableau (on ne travaille pas véritablement sur des objets JSON mais sur un tableau key-value multidimensionnel), puis utilise les résultats du parseur (les objets PHP générés) pour les transformer en fichiers HTML concrets. Affiche graphiquement le résultat du passage : soit les erreurs générées par le parseur, ou les fichiers générés, que l'utilisateur peut consulter immédiatement. Offre la possibilité d'exporter le résultat compressé au format zip ou de convertir un réseau de pétri différent (uniquement l'affichage des boutons. Les actions sur les boutons sont attachées en JQuery dans le fichier common.js).
- **upload.php** : Affiche l'interface d'upload des fichiers JSON et STAMP.
- **uploadStamp.php** : dépose sur le serveur le fichier STAMP reçu dans la variable \$_FILE.
- **create-zip.php** : crée un fichier zip contenant les fichiers générés du projet courant.

- **class** : contient tous les objets (les classes) du projet. On y trouve dans ce dossier les objets :
 - o **Event** : Objet décrivant un évènement d'une transition
 - o **Scene** : Objet décrivant une scène du réseau de pétri
 - o **Sprite** : Objet décrivant un élément HTML attaché à une scène
 - o **Transition** : Objet contenant tous les évènements attachés à une scène
 - o **View** : Objet regroupant plusieurs scènes

Chacun des fichiers PHP de ce dossier contient donc une classe représentant un objet reflétant les différents objets du fichier JSON, et contiennent des méthodes supplémentaires pour traiter et interpréter les données afin d'en générer du code HTML-JS-CSS.

- **css** : Contient les feuilles de style de l'interface utilisateur
 - o **debug.css** : Fichier inclus dans chaque document HTML généré lorsque l'application tourne en mode debug. Il définit le style des éléments ajoutés dans la page par le script « debug.js ».
 - o **style.css** : Contient le style de tous les éléments de l'interface utilisateur.
- **demo** : Contient des exemples de réseaux de pétri décrit en JSON à des fins de démo utilisateur lorsque l'application sera publiée ou de debug pour les développeurs qui devraient corriger ou améliorer/étendre l'application.
 - o **Dossier « debug »** : lorsque l'application est en mode debug (cf. section concernant le mode debug), les fichiers contenus dans ce dossier seront pris en considération par l'application et seront utilisés pour générer le site web du réseau de pétri décrit en JSON.
On peut alors utiliser les fichiers de ce dossier pour le développement de l'application ou pour debugger un problème éventuel. Il suffit de passer en mode debug, et de changer le contenu des fichiers JSON et STAMP.
Dans ce dossier, le fichier STAMP n'a pas forcément besoin d'être cohérent puisqu'il n'est jamais utilisé en mode debug (aucun mot de passe n'est demandé). On peut par contre vouloir générer un fichier STAMP à partir du mode debug pour un fichier JSON particulier. C'est dans ce cas précis que le fichier STAMP de ce dossier se révélera intéressant.
- **img** : contient les images utilisées dans l'interface utilisateur
- **js** : contient les fichiers JavaScript utilisés dans l'interface utilisateur
 - o **common.js** : Toutes les fonctions, animations et évènements attachés à l'interface utilisateur.
 - o **debug.js** : Fichier attaché à chaque document HTML généré lorsque l'application tourne en mode debug. Il définit les éléments à ajouter dans la page et leur comportement afin d'offrir un outil de débogage qui simplifie la vie du développeur.
 - o **jquery-3.2.1.min.js** : Librairie JQuery version 3.2.1. Utilisé pour l'interface utilisateur et non comme dépendance des fichiers HTML générés !
- **utils** : contient les fichiers regroupant les méthodes, variables et constantes globale à l'application
 - o **common.php** : Regroupe toutes les fonctions utiles à travers l'application.
 - o **const.php** : Regroupe toutes les constantes de l'application

5.2.1. Fichiers et dossiers générés

Lors de la mise en production de cette application web, les dossiers ci-dessous doivent posséder les droits nécessaires en écriture afin de recevoir les fichiers utilisateurs ou les fichiers générés à partir des fichiers utilisateurs. Si les droits en écriture ne sont pas accordés à ces dossiers, l'application ne fonctionnera pas.

- **generated** : Contient les dépendances de base, communes à tous les fichiers des projets générés. C'est dans ce répertoire que les fichiers HTML générés à partir du JSON sont placés. Chaque projet est placé dans un dossier spécifique. Les dépendances de bases y sont également copiées.
- **uploads** : contient les fichiers JSON et les fichiers STAMP des réseaux de pétri uploadés par les utilisateurs. Il ne fait en fait que de stocker temporairement les fichiers afin d'extraire leur contenu pour générer les fichiers HTML correspondants.
- **zips** : contient les archives zip de chaque projet. En effet, lorsque la conversion du fichier JSON en site web est terminée, l'utilisateur peut choisir d'exporter les fichiers afin de les uploader vers un serveur web. Chaque zip de ce dossier contient les mêmes fichiers que le dossier associé au projet dans le répertoire « generated ».

5.3. FICHIER JSON : CONSTRUCTION ET TRAITEMENT DES DONNÉES

Le fichier JSON contient la description du réseau de pétri au format JSON, tel que généré par l'application de schématisation.

Un logiciel est déjà existant pour la schématisation des réseaux de pétri de base, mais celui-ci n'est pas complètement adapté à la schématisation de réseaux de pétri décrivant des applications web. Il faudra donc l'adapter pour qu'il propose par exemple de donner différents attributs aux tokens des places (aux sprites des scènes).

Quelques explications sont donc données ici afin d'informer un futur développeur de la façon dont le générateur traite les données et de définir alors la partie qui convient d'être implémentée sur le logiciel de schématisation.

5.3.1. Ordre et nommage des sprites

Les sprites, tout comme les scènes, les vues et les transitions, doivent être nommés de façon unique. L'identifiant peut être un entier ou une chaîne de caractère.

L'ordre d'affichage est séquentiel et l'ordre a donc une importance. Ce qui signifie que si le sprite avec l'id 15 est placé en premier et le sprite id 1 à la fin de la liste des sprites, le sprite avec l'id 15 sera affiché en premier et le sprite id 1 en dernier. L'identifiant, qu'il soit une chaîne de caractère ou un entier n'influence en rien l'ordre d'affichage des sprites en tant qu'éléments HTML.

5.3.2. Sprites enfants, ordre

Chaque sprite peut contenir d'autres sprites, ce qui correspond à imbriquer des éléments HTML lors de la génération. Le traitement des enfants utilise une structure de donnée en arbre et un algorithme de tree traversal inversé pour les afficher dans l'ordre à l'intérieur de leur parent. L'arbre est rarement équilibré lorsque sa profondeur dépasse 2, puisque certains sprites contiendront des enfants qui eux même contiendront d'autres enfants, et certains n'en contiendront pas, ce qui créera des profondeurs différentes sur chaque branche. Cette partie du traitement des données est la moins évidente de l'application au niveau code et algorithmes.

L'ordre n'a pas d'importance en ce qui concerne le placement des sprites qui contiennent des enfants dans le fichier JSON. Autrement dit, le sprite parent peut être placé avant, après, ou au milieu des enfants, et les enfants n'ont pas besoin d'être placés dans l'ordre dans le fichier. **Mais l'ordre dans lequel les enfants sont spécifiés dans le tableau est important** ! Si on donne pour enfants les identifiants des sprites [12,10,11], on placera d'abord l'élément avec l'id 12 en premier, ensuite 10, puis 11. Si l'identifiant d'un sprite qui n'existe pas est spécifié dans la liste des enfants, la génération va s'interrompre et un message affichera l'erreur

« Le sprite avec l'id xxx est utilisé comme enfant du sprite xx mais n'existe pas »

Si un sprite est utilisé plusieurs fois comme enfant dans plusieurs sprites parents différents, le comportement n'est pas défini, il se peut que tout fonctionne correctement, mais il se pourrait aussi qu'une erreur soit déclenchée. En revanche, un sprite peut être utilisé plusieurs fois comme enfant du même parent. Autrement dit, la liste d'enfants [3,3,3,4,4,5] est valide, mais si un autre sprite contient lui aussi les sprites 3,4 et 5, il se pourrait qu'il déclenche un message d'erreur :

Le sprite 1 est utilisé comme enfant par plusieurs sprites conteneurs différents

5.3.3. Traitement des erreurs

Lors du traitement du fichier JSON, si des erreurs se produisent, celles-ci seront reportées à l'utilisateur au lieu de faire crasher l'application. Le but est d'éviter autant que possible d'afficher à l'utilisateur les erreurs générés par PHP avec leur numéro de lignes, afin d'avoir une application professionnelle et non juste un simple projet d'étudiant qui crash lorsqu'une situation inattendu par le parser se produit.

Bien entendu, toutes les erreurs possibles et imaginables n'ont pas été traitées, seul lorsque des erreurs se sont produites ou lorsque la possibilité d'une erreur était évidente, les cas ont été traités, et il est possible que des erreurs encore non rencontrées puissent faire crasher le parser. Il suffira alors de rajouter le traitement de cette erreur, en l'ajoutant au tableau \$err qui contient toutes les erreurs affichées à l'utilisateur.

5.3.4. Vues

Au niveau HTML, une vue ne contient que des iframes avec les attributs "src", "id" et "style". Les iframes contiennent chacune une scène. La vue offre donc une flexibilité assez grande. On peut réutiliser plusieurs fois une même scène dans différentes vues, disposées de manière différentes. La vue est l'un des deux éléments disponibles dans la navigation du site, le deuxième étant la scène. Des pages sont créés pour les vues et pour les scènes séparément. Donc si une scène n'est jamais utilisée dans une vue, elle est tout de même générée dans une page qui lui est dédiée. A l'intérieur d'une vue, les frames peuvent charger de nouveaux contenus dans d'autres frames ou à l'intérieur de leur propre frame, ou encore charger une nouvelle vue ou une simple scène.

5.3.5. Scènes

La scène peut contenir tous types d'éléments HTML. Elle pourrait au même titre que la vue contenir des iframes également si l'utilisateur le souhaite, par exemple pour afficher le contenu d'une page externe sur internet ou encore une vidéo youtube (qui peut aussi plus simplement être intégrée par un élément "video" en spécifiant l'attribut "src"). Les éléments HTML intégrés dans une scène sont des Sprites.

5.3.6. Sprites

Les sprites représentent des éléments HTML natifs. Vous avez donc à loisir de leur attribuer tous les attributs disponibles en HTML ainsi qu'une valeur affichée, par exemple le texte d'un paragraphe, ou la valeur par défaut d'un champ de formulaire.

5.3.7. Transitions

Une transition représente une action déclenchée par un élément HTML. Soit auto déclenché de lui-même, par exemple à la fin d'une vidéo, ou déclenché par une interaction de l'utilisateur avec l'élément.

Chaque transition contient un identifiant unique, une association In, et une association Out. Toutes les transitions d'une scène peuvent être groupées à l'intérieur d'un objet dans le tableau des transitions, sous forme de plusieurs règles.

Les associations In sont les prérequis; les conditions à remplir sur un sprite pour qu'une transition soit déclenchée. Elles décrivent la source et l'association out la destination.

Les associations Out sont liées aux associations In et décrivent l'aboutissement lors du déclenchement d'une transition décrite par l'association In. Concrètement, l'association out décrit la ou les scènes de destination lorsque la condition décrite par l'association In associée est remplie.

Chaque association In / Out contient un identifiant, et une ou plusieurs règles

Ci-dessous, vous trouverez de plus amples informations concernant la signification des champs de l'association In et de l'association Out.

Associations In

- **"id"** : Identifiant unique de l'association In.
- **"regles"** : une règle s'applique sur un ou plusieurs sprites d'une ou plusieurs scènes.
 - o **"id"** : Doit être le même pour l'associationOut lié à l'associationIn.
 - o **"sprites"** : liste contenant les sprites déclencheurs de la transition et l'action qui va appeler l'association out.
Exemple : "sprites":["1.endDuration", "2.click"]
Ici, la fin de video du sprite avec l'identifiant 1 va déclencher la transition out, et la même transition out sera également déclenchée par le clique sur le sprite avec l'identifiant 2.
 - o **"scenes"** : décrit la ou les scènes sources contenant les éléments déclencheurs de la transition out.

Associations Out

```
"id":"identifiant unique de l'association out",
"regles":[
  {
    "id": "idenfiant unique de la règle, non utilisé par le parser",
    "dest": [
      {"type":"scene ou view", "id":"sceneID ou viewID", "targets":["frame1"]},
      {"type":"scene ou view", "id":"sceneID", "targets":["frame2", "frame3"]}
    ]
  }
]
```

Dans l'exemple ci-dessus, l'association out charge la scène (la view) donnée par sceneID (viewID) dans les iframes spécifiées de la vue dans laquelle on se trouve. Si on se trouve dans une scène (qui n'a donc pas de « targets » à spécifier), ou si on cherche simplement à changer de page, on ne donne simplement qu'un tableau vide ou pas de propriété targets du tout.

Il est également possible d'utiliser un autre type de règles pour les associations out. Ci-dessous, voici un exemple utilisant deux règles contenant des destinations de type js (JavaScript). Les destinations de type js acceptent 2 types d'actions : load, et animation. En utilisant le type load, on peut charger une scène ou une vue (spécifié par « source ») dans

un sprite spécifié par « target ». En utilisant l'action animation, on peut lancer une animation sur un sprite au déclenchement d'un évènement particulier.

```
"associationOut": {
  "id": "...",
  "regles": [
    {
      "id": "bt2",
      "dest": [
        {
          "type": "js",
          "js": {
            "action": "load",
            "target": "container2",
            "source": { "scene": "scene7" }
          }
        }
      ]
    },
    {
      "id": "bt3",
      "dest": [
        {
          "type": "js",
          "js": {
            "action": "animation",
            "target": "container3",
            "animations": [ { "type": "toggle" } ]
          }
        }
      ]
    }
  ]
}
```

5.3.8. Mises en pages avancées, compositions complexes

Les possibilités sont infinies, le parseur peut interpréter tout ce qui peut être fait par la main d'un programmeur. La seule restriction est l'imagination, et **les possibilités qu'offre l'interface de création visuelle** du réseau de pétri décrivant le site web. L'interpréteur est maintenant fonctionnel, il faudra à présent que le logiciel actuel de schématisation des réseaux de pétri soit adapté et étendu pour donner la possibilité de créer des sites web.

Si on veut par exemple créer des grilles, le générateur peut très facilement l'interpréter, mais la grille doit être créée dans le logiciel de schématisation. Si l'utilisateur est un peu plus connaisseur, il peut se simplifier la vie en utilisant la librairie bootstrap qui peut être incluse de manière très simple dans les projets et qui propose une très large gamme de styles et fonctions déjà prêtes, pour se simplifier la vie tout en profitant de la compatibilité multi-navigateurs pour construire des interfaces « responsive » en un clin d'œil.

5.3.9. Dépendances, ajout de bibliothèques

Les bibliothèques JQuery, JQuery UI, et Bootstrap sont disponibles nativement dans toutes les applications générées. JQuery et JQuery UI sont intégrés automatiquement car de nombreuses fonctionnalités générées utilisent les bibliothèques. Cependant, Bootstrap n'est pas intégré si l'utilisateur ne l'inclue pas lui-même dans son application, mais une seule ligne suffit pour le faire :

```
"include": {
  "libraries": ["bootstrap"]
}
```

Il est également très simple d'ajouter d'autres librairies JavaScript et/ou CSS si l'utilisateur est déjà familier avec l'une ou l'autre des nombreuses librairies disponibles sur le web. Il peut spécifier le nom des fichiers à inclure de cette façon :

```
"include":{
  "js":["file1.js", "file2.js", "file3.js"],
  "css":["file1.css", "file2.css"]
}
```

Les références aux fichiers seront ajoutés lors de la génération, mais les fichiers ne seront pas sur le serveur, donc l'aperçu de l'application en direct lors de la génération pourrait ne pas fonctionner complètement si l'utilisation des fichiers ajoutés est fortement impliquée.

L'application doit être exportée, et les fichiers .js / .css ajoutés respectivement dans les répertoires « dependencies/js » et « dependencies/css ». Alors, l'application sera complète et fonctionnera parfaitement. Cette étape n'est pas nécessaire avec Bootstrap car la librairie est incluse dans les dépendances par défaut si l'utilisateur choisi de l'activer.

La raison pour laquelle il n'y a pas plus de dépendances proposées par défaut comme Bootstrap, est que les librairies pour le web changent très rapidement, et parfois, le code des anciennes versions ne fonctionne plus avec les nouvelles versions (jQuery en est un bon exemple).

5.4. RÔLE ET FONCTIONNEMENT DU FICHIER STAMP

Le fichier STAMP (nom donné en référence à l'emprunte numérique qu'il contient), a été introduit dans le but de sécuriser le fichier JSON décrivant le site web. Il contient deux informations :

- L'emprunte du fichier JSON en sha512 hachée à la façon HMAC en utilisant comme clé secrète le mot de passe de l'utilisateur haché en sha512
- Le mot de passe de l'utilisateur haché en sha512.

Le mot de passe permet au concepteur du réseau de pétri d'être le seul à pouvoir générer un site web à partir de son réseau de pétri, même si quelqu'un possède son fichier JSON décrivant le réseau.

L'emprunte permet de détecter si l'utilisateur a tenté de modifier le fichier JSON ou si c'est un fichier authentique généré par l'application.

Note : Actuellement, le mot de passe ne protège que partiellement les fichiers, car un informaticien qui veut s'approprier le fichier JSON et pouvoir en générer l'application web associée pourrait facilement forger le contenu du fichier STAMP avec une fausse empreinte aléatoire et un vrai mot de passe haché en SHA512. Son mot de passe lui sera demandé – il le connaît puisqu'il l'a choisi lui-même – l'application web sera générée, et seul un message avertissant que « le fichier STAMP ou le fichier JSON a été modifié » sera affiché. On pourrait à ce moment-là décider de pousser la sécurité plus loin : si l'emprunte ne correspond plus à celle qu'on devrait avoir, on ne laisse pas générer le site web. L'inconvénient (ou l'avantage selon le point de vue), est que l'utilisateur ne peut plus du tout modifier à la main son fichier JSON même s'il sait le faire de manière sûre. Comme à ce stade le mot de passe ne donne pas une sécurité très poussée, la possibilité de désactiver la demande de mot de passe lors de la génération a été implémentée.

5.4.1. Pourquoi avoir un 2^e fichier détaché du fichier JSON ?

Dans le but de pouvoir stocker l'emprunte complète du fichier JSON. Si l'emprunte elle-même est stockée dans le fichier JSON, le fichier est donc modifié par l'ajout de l'emprunte après le calcul de l'emprunte, ce qui posera problème au moment de recalculer l'emprunte pour la comparer avec celle stockée dans le fichier, puisque l'emprunte recalculée sera différente.

5.5. MODE DEBUG

Dans un but de débogage et de développement de l'application de manière générale, un mode debug a été implémenté.

Il suffit de définir la variable `$debug_mode = true;` dans la page `index.php` et les outils de débogage seront disponibles sur toutes les pages générées.

5.5.1. Mode normal vs. debug

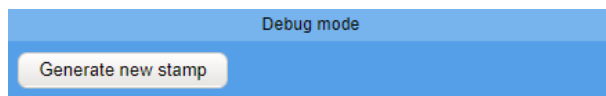
La génération en mode normal (non debug) appelle les fichiers PHP de manière asynchrone de manière à offrir une interface user-friendly munie d'un loader si le fichier JSON est extrêmement lourd et un peu long à convertir.

Le mode debug ne génère pas d'appel asynchrone en javascript, Les pages PHP sont incluses directement. De cette façon, la plupart des messages d'erreurs éventuels peuvent être affichés immédiatement.

Le mode debug peut être conservé (mais désactivé) même lorsque le logiciel est prêt à être publié. Pour cela, il convient de définir la variable `$debug_mode = false` dans la page `index.php` et aucune trace du mode debug ne sera visible.

5.5.2. Génération d'un nouveau STAMP

Pour générer un fichier STAMP associé à un fichier JSON, un bouton est disponible en mode debug. Le mot de passe y est demandé, et l'emprunte du fichier suivi du mot de passe haché sont créés dans un fichier STAMP, ou le fichier STAMP existant est mis à jour.

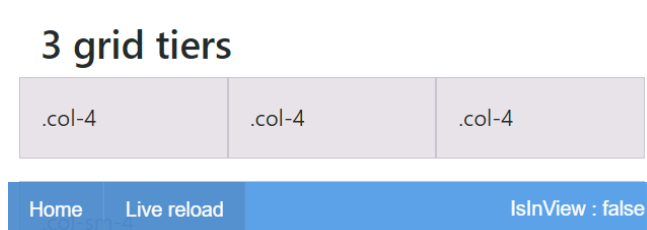


5.5.3. Génération et aperçu accélérés

L'aperçu de l'application générée, en mode debug, fournit une barre d'outil dans chaque scène et dans chaque vue.

Celle-ci permet de recharger très rapidement les pages lors du développement en cliquant sur le bouton « Live reload », sans passer par la page de génération. La génération est faite dans une tâche asynchrone par un thread en arrière-plan, puis la page est rechargée automatiquement.

Le bouton « Home » permet de revenir à la page de génération instantanément lorsque l'on a parcouru un grand nombre de page, pour éviter de devoir cliquer 20x sur le bouton retour.



5.6. ACTIVER / DÉSACTIVER LA DEMANDE DE MOT DE PASSE

Comme pour le mode debug, il est possible de définir si les utilisateurs doivent entrer un mot de passe ou pas afin de générer l'application web décrite par le fichier JSON. Ceci se fait en définissant la variable `$mdpRequired` à `true` ou `false` dans la page « `index.php` ».

5.7. FICHIERS MULTIMÉDIAS DE L'UTILISATEUR

L'utilisateur qui inclut dans son projet des fichiers multimédias qu'il possède sur son disque dur local ne les verra pas lors de l'aperçu des fichiers générés.

Pour voir la version finale utilisant ses fichiers multimédias, il doit les inclure tous à la racine de son site web, dans le même dossier que les fichiers HTML exportés en zip.

Cette contrainte est obligatoire en partant du principe que l'application est un service hébergé sur un serveur web (toute l'application étant écrite dans un langage web).

Ce comportement pourra être modifié par la suite, si le propriétaire de l'application décide d'intégrer cet outil comme une étape finale au logiciel visuel de création du réseau de pétri.

5.8. COMPATIBILITÉ SMARTPHONES / TABLETTES

L'utilisateur peut tout à fait créer des applications responsive sur appareils mobiles et PC. Pour ce faire, il peut activer la librairie Bootstrap qui lui fournira des classes toutes prêtes, ou il peut ajouter une librairie comme JQuery mobile qui lui facilitera la tâche.

Une autre possibilité pour les utilisateurs plus expérimentés est de créer leurs propres CSS media queries :

```
@media screen and (max-width: 1280px) {  
  /* Propriétés CSS */  
}
```

Voici un exemple simple de grille générée en activant Bootstrap. Les affichages sur mobile en mode portrait et en mode paysage adaptent la grille pour que l'aspect visuel reste agréable et adapté au mode d'affichage de l'utilisateur.

3 grid tiers

.col-4	.col-4	.col-4
.col-sm-4	.col-sm-4	.col-sm-4
.col-md-3		
.col-md-6		
.col-md-3		

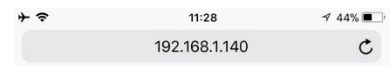
Home Live reload IsInView : false

iPhone : Grilles en mode paysage

3 grid tiers

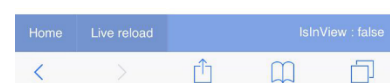
.col-4	.col-4	.col-4
.col-sm-4	.col-sm-4	.col-sm-4
.col-md-3	.col-md-6	.col-md-3

PC : Affichage étendu



3 grid tiers

.col-4	.col-4	.col-4
.col-sm-4		
.col-sm-4		
.col-sm-4		
.col-md-3		
.col-md-6		
.col-md-3		



iPhone : Grilles en mode portrait

6. LISTE DES MOTS CLÉS JSON TRAITÉS (API)

attributeName* : attribut requis. Ceux qui n'ont pas d'astérisque sont optionnels et ne compromettent pas le bon fonctionnement de la génération s'ils ne sont pas renseignés.

- **index** : fichier de départ de l'application, le fichier index.html (une scène ou view existante)
 - **type*** : "view" ou "scene"
 - **id*** : identifiant de la vue ou de la scène à afficher comme page de départ

- **include** : Fichiers supplémentaires à attacher au projet ou librairies à activer
 - **js** : tableau contenant les noms des fichiers js personnalisés à ajouter au projet
 - **css** : tableau contenant les noms des fichiers css personnalisés à ajouter au projet
 - **libraries** : ["bootstrap"] seul librairie disponible par défaut pour le moment

- **scenes*** : Scenes du projet (transformées en pages HTML à la génération)
 - **id*** : Identifiant unique de la scène, entier ou chaîne de caractère
 - **title** : Titre de la page HTML affiché dans la balise <title></title>. Par défaut : l'identifiant de la scène
 - **mobile** : true si l'application est utilisée pour mobile aussi, des balises meta sont ajoutées
 - **lang** : Langue de la page. Informe le navigateur de la langue utilisée dans la page. ex. : fr
 - **style** : Style CSS appliqués au body de la page comme ci-dessous :

```
"style": {
  "background" : "#f8f8f8",
  "color"      : "brown",
  "font-size"  : "15px"
}
```
 - **js** : JavaScript libre ajouté à la page (tout le code JS, JQuery et JQuery UI est permis)

```
"js": [
  "$('.contact-box').css('display', 'none')",
  "setTimeout(function() { $('.contact-box').slideDown(3000) }, 1000)"
]
```
 - **css** : Règles CSS personnalisées à ajouter à la page. Tout le CSS existant est autorisé

```
"css":{
  ".contact-box"      : { "color":"#020015", "padding":"6px", "margin":"20px auto" },
  ".contact-box:hover" : { "cursor":"pointer", "background":"blue" },
}
```
 - **sprites** : Liste des éléments HTML contenus dans la page
 - **id*** : Identifiant unique du sprite, entier ou chaîne de caractère
 - **nature*** : Le type d'élément HTML que représente ce sprite (p, div, h1, video, img, ...)
 - **value** : Texte affichée à l'intérieur de l'élément HTML. Peut aussi contenir des balises HTML.
 - **attr** : Vous pouvez utiliser tous les attributs existant sur l'élément HTML (par ex. : class, width, src, ...)

```
"attr":{
  "src"      : "https://www.youtube.com/embed/Vpg9yizPP_g",
  "width"    : 560,
  "height"   : 315
}
```
 - **style** : style CSS personnalisé pour cet élément. On peut affiner un élément qui a la même class css que d'autres

```
"style":{
  "border" : "3px solid black",
  "cursor" : "pointer"
}
```
 - **hover** : Style CSS appliqué à l'élément lors du passage de la souris sur ce dernier

```
"hover" : {
  "opacity" : 0.8,
  "border"  : "4px solid #000"
}
```
 - **childs** : Liste des sprites enfants (les sprites imbriqués dans cet élément), ex : [7,8]
 - **animations** : Animations appliquées sur l'élément HTML. On peut en ajouter autant qu'on veut.
 - **type** : **in** : animation entrante, **out** : sortante, **toggle** : entrante / sortante, **custom** : personnalisé


```

--- delay : Temps en millisecondes avant de démarrer l'animation. Par défaut : 0
--- effect : blind, bounce, clip, drop, explode, fade, fold, highlight, puff, pulsate, scale, shake, size, slide, transfer
    Par défaut : fade. Aperçu des différents effets : https://jqueryui.com/effect/
--- duration : Durée de l'animation en ms. Par défaut : 500
--- interval : Interval de répétition de l'animation, en millisecondes
--- ... (avec type="custom") : Vous pouvez spécifier toutes les propriétés CSS que vous voulez animer
    "animations": [
      {
        "type" : "custom",
        "opacity" : "0.25",
        "padding" : "20px",
        "delay" : 4000
      }
    ]
--- clone : Identifiant du Sprite à cloner (le sprite cloné doit être dans la même scène)
--- count : Combien de fois répéter le Sprite courant

```

```

- views : Les vues sont faites d'iframes qui contiennent des scènes que l'on peut disposer dans la page.
  -- id* : Identifiant unique de la vue, un entier ou une chaîne de caractère
  -- title : Titre de la page HTML affiché dans la balise <title></title>. Par défaut : l'identifiant de la vue
  -- scenes* : Tableau d'objet des scènes à inclure dans la vue
    --- frame-id* : Identifiant unique de l'iframe dans la page. Indispensable pour gérer la navigation interframes.
    --- scene-id* : Identifiant de la scène à afficher dans l'iframe
    --- style* : Style (pseudo)-CSS pour le placement de l'iframe (propriétés supplémentaires interprétées)
      Ex 1 : "style":{"x":0, "y":0, "width":"50%", "height":0}
      Ex 2 : "style":{"right":"25%", "top":"210px", "width":0, "height":"200px"}

```

```

- transitions* : Représente les événements attachés aux éléments HTML (sprites) des vues et scènes
  -- id* : Identifiant unique de la transition (uniquement utile au logiciel de schématisation, non utilisé)
  -- associationIn* : Conditions à remplir pour déclencher l'événement
    --- id* : Identifiant unique (uniquement utile au logiciel de schématisation, non utilisé)
    --- regles* : Règles (conditions) qui déclenchent les règles de l'association Out
    --- id* : Identifiant unique de la règle, doit être le même pour la règle de l'association out correspondante
    --- sprites* : Identifiants des sprites et le type d'événement qui déclenche l'action
      "sprites":["1.endDuration", "2.click", "3.click"]
    --- scenes* : Scène (une seule !) contenant les sprites qui écoutent l'événement
      "scenes":["scene4"]
  -- associationOut* : Actions à effectuer lorsque l'association In est déclenchée
    --- id* : Identifiant unique (uniquement utile au logiciel de schématisation, non utilisé)
    --- regles* : Règles (conditions) déclenchées par les règles de l'association In
    --- id* : Identifiant unique de la règle, doit être le même pour la règle de l'association In correspondante
    --- dest* : Page de destination ou action JavaScript à appeler au déclenchement de l'événement
    --- type* : Valeurs possibles : scene, view ou js
    --- id* (avec type scene ou view) : Identifiant de la scène ou de la vue à charger
    --- targets : dans quel(s) frame(s) de la vue charger la scène ou vue spécifiée par son id. Pas nécessaire si on désire simplement changer de page (pas charger une page dans une frame).
      "targets":["frame1", "frame2", "frame3"]
    --- js : Si le type de la destination est "type":"js", on peut spécifier les paramètres suivants
      --- action* : load ou animation
      --- target* : Identifiant du sprite dans lequel on va charger le contenu spécifié par "source"
      --- source : La scene ou view à charger dans la target spécifiée.
        "source":{"scene":"scene7"} ou "source":{"view":"view2"}
      --- animations : Toutes les animations disponibles (décrites ci-dessus pour les sprites)

```