Rapport Domain Specific Language

WebGen

Générateur d'IHM pour réseaux sociaux

Équipe H
Eric BOUDIN, Clément MONESTIER, Florian NAUD,
Lucas RAKOTOMALALA, Loïc RIZZO

1 mars 2022

Enseignant
Julien DEANTONI

POLYTECH
NICE-SOPHIA

Université
Nice
Sophia Antipolis

Table des matières

1 Informations

Le code de notre DSL est disponible sur ce repository Github.

2 Langages

2.1 Choix technologiques

Nous avons opté dans un premier temps pour le langage MPS pour le DSL, et ensuite pour du JavaScript pour la partie du langage généré. Nous exploitons les librairies React pour la structure et l'affichage du frontend et Grommet pour utiliser des composants déjà construits et facilement configurables.

Il est important de noter que notre DSL est voué à être utilisé par des UI designer. Ainsi, l'obligation de télécharger un nouvel *IDE* pour pouvoir utiliser le DSL externe (*MPS* ne devrait pas être un obstacle, bien que son utilisation n'est pas triviale au début. De plus, nous ne connaissons que la création d'un DSL *textuel*, alors que dans le cadre du projet, il aurait été plus adapté de réaliser un DSL *graphique*.

2.1.1 MPS

Nous avons choisi MPS puisqu'il s'agit d'un outil que nous avons utilisé lors du précédent projet. Il nous a permis d'être encadré lors du développement du DSL.

MPS possède aussi des méthodes de vérification des variables données par l'utilisateur dans le langage DSL créé.

2.1.2 React

React est un framework de frontend rapide à mettre en place et populaire dans le développement Web. Il possède une grande communauté et donc de nombreuses documentations. *React* nous permet donc de facilement mettre en place des composants représentant les éléments de notre domaine modèle.

2.1.3 Grommet

Grommet est un framework développé par dessus React permettant de facilement réaliser des applications *responsive* grâce aux nombreux composants qu'il possède. Nous avons choisi ce framework pour sa facilité de prise en main, d'autant plus que plusieurs personnes du groupe avait peu ou pas d'expérience en *React*. De plus, grâce à son système de thème, il nous permet de gagner du temps sur la mise en place du design.

2.2 Modèle sémantique

Notre modèle sémantique prends en compte à la fois, le positionnement des éléments sur la page et leur style. Dans l'idée, nous utilisons le terme *template* pour parler du positionnement

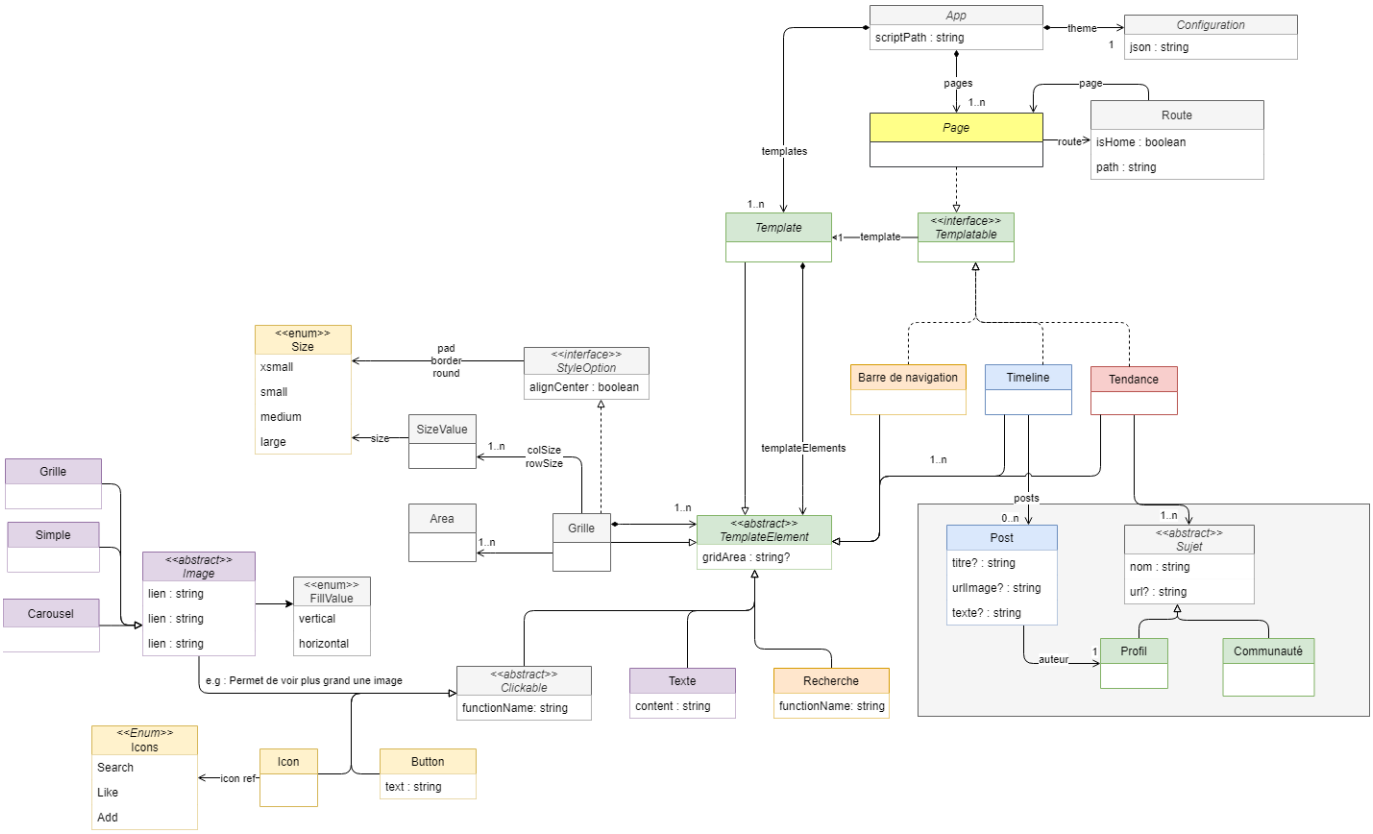

FIGURE 1 – Modèle sémantique général

des éléments sur la page. On utilise une structure qui s'assimile à celle utilisée pour le HTML, de haut en bas, sauf dans le cas du concept de grille où nous pouvons positionner les éléments avec plus de liberté.

Le style est ajouté sur certains éléments, en priorité par souci de temps, l'image et la grille.

La plupart des concepts se voient implémenter la classe abstraite *TemplateElement* qui signifie que le concept peut faire parti d'un template.

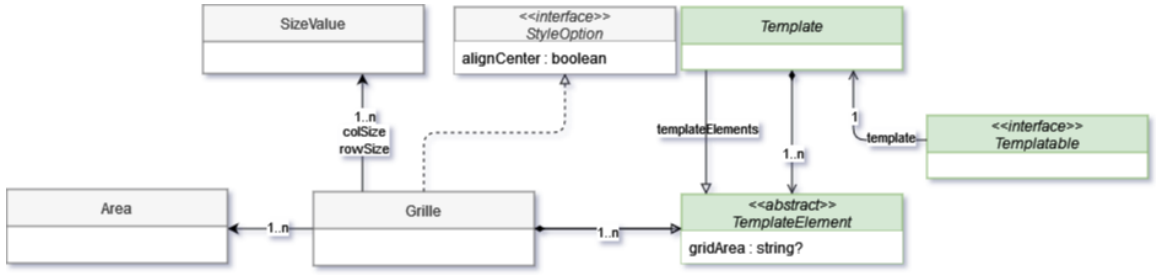

FIGURE 2 – Modèle sémantique du templating

Nous avons voulu fournir des composants indispensables à la construction de réseaux sociaux, comme des images, des boutons, des icônes. Tous sont des **TemplateElement** et peuvent être utlisés dans des **Template**. Au final nous voulons laisser le choix au designer de positionner les éléments à l'endroit où il le souhaite.

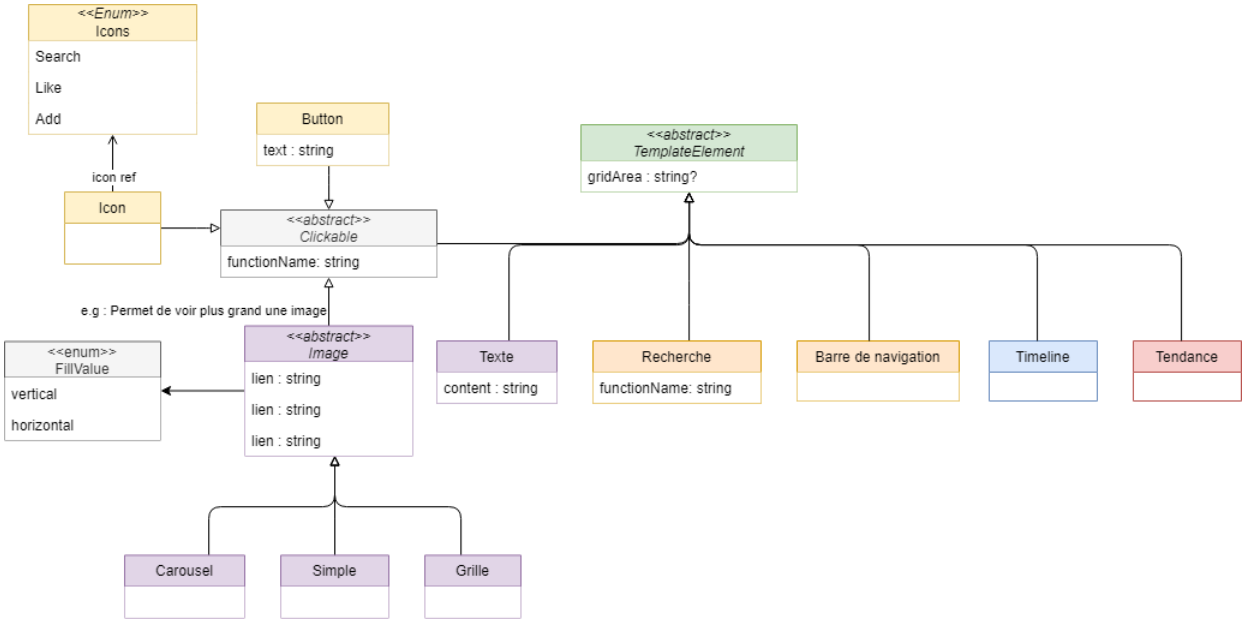

FIGURE 3 – Modèle sémantique des composants

Tout au long du développement de notre DSL, nous avons mis à jour le modèle sémantique. Il est important de noter que les mises à jour du modèle sémantique étaient faites en amont du développement, afin de nous assurer que nos DSL couvraient les mêmes cas d'utilisation pour une certaine cohérence. Pour cela, nous avons utilisé l'outil en ligne *Draw.io*.

Pour versionner nos différents changements, nous avons usé de la possibilité de créer plusieurs pages sur l'outil en ligne que nous avons utilisé.

2.3 Syntaxe

Voici une syntaxe "BNF like" de notre DSL :

Listing 1 – App BNF

```
<App> ::=
    "WebGen : " <string>
    "Script path : " <string>

    "Theme : " <Configuration>
```

```
    "Templates : " <Template>+

    "Pages : " <Page>+
```

Listing 2 – Area BNF

```
<Area> ::= "grid area named " <string> " will start at (row,column)
(" <int> ", " <int> ") and end at (" <int> ", " <int> ")"
```

Listing 3 – Button BNF

```
<Button> ::= "Text button " <string> " contains "
<string> " on " <Clickable> <GridArea>
```

Listing 4 – Clickable BNF

```
<Clickable> ::= "click event " <string>
```

Listing 5 – Configuration BNF

```
<Configuration> ::= "Theme " <string> " insert json theme : " <string>
```

Listing 6 – DisplayPost BNF

```
<DisplayPost> ::= "Displaying a post with <TemplateElement>
```

Listing 7 – Grid BNF

```
<Grid> ::=
    "Grid component :"
        <GridArea>
        "Style :"
            <Border>
            <Pad>
            <Round>
            <AlignCenter>
        "colmuns size :"
            <SizeValue>+
        "rows size :"
            <SizeValue>+
    "with following areas :"
        <Area>+
    "for templates elements :"
        <TemplateElement>+
```

Listing 8 – GridArea BNF

```
<GridArea> ::= "Grid area: " <string>
```

Listing 9 – Icon BNF

```
<Icon> ::= "Icon: " <string> <Icons> " on " <Clickable> <GridArea>
```

Listing 10 – Image BNF

```
<Image> ::=
    "Image: link to " <string> ", on " <Clickable>
    <GridArea>
    "Filling :" <FillValue>
```

Listing 11 – Navbar BNF

```
<Navbar> ::=
    "Navbar contains " <string>
    <GridArea>
```

Listing 12 – Page BNF

```
<Page> ::=
    "Page " <string> ":"
        <Route>
        "using template " <string>
```

Listing 13 – Route BNF

```
<Route> ::=
    "Route " <string> " is page index " <boolean>
    " for page <Page::string>
```

Listing 14 – Search BNF

```
<Search> ::=
    "Search with " <string> " sending to " <string>
    <GridArea>
```

Listing 15 – SizeValue BNF

```
<SizeValue> ::= <Size>
```

Listing 16 – Template BNF

```
<Template> ::=
    "Template " <string> " :"
        <TemplateElement>+
```

Listing 17 – Tendance BNF

```
<Tendance> ::=
    "Tendency displaying like " <string> <GridArea>
```

Listing 18 – Texte BNF

```
<Texte> ::=
    "Text: " <string> <GridArea>
```

Listing 19 – Timeline BNF

```
<Timeline> ::=
    "Tendency displaying like " <string> <GridArea>
```

Compréhensivité

La syntaxe montre notre choix de réaliser un templating de la structure de la page avant tout. Elle se veut rigoureuse et force l'utilisateur à définir strictement chaque composant qu'il souhaite et à les positionner/styliser. Tous les choix de styles sont visibles par défaut, et seul le positionnement dans une grille est visible à la seule condition, que le composant soit dans une grille. Notre choix du thème a été impactant sur la difficulté de réalisation de notre DSL, et on peut le ressentir sur la grammaire. Celle-ci se voudrait moins verbeuse et plus simple en abstractant les concepts plus loin et plus proche de l'universe d'un UI designer.

Listing 20 – Exemple du code du scénario **Site à page unique**

```
WebGen : HomeUI
ScriptPath :
  ./script.js

Theme myTheme , insert json theme :
  {}

Template :
  Template Tendance :
    Text: Here is your tendencies !

Template Post :
  Grid component :
    Style :
      Border : true
      Padding : small
      Rounding : medium
      Align center : false
    columns size :
      -> auto
    rows size :
      -> xsmall
      -> auto
  with following areas :
    grid area named title will start at (row,column) ( 0 , 0 ) and end at ( 0 , 0 )
    grid area named img will start at (row,column) ( 1 , 0 ) and end at ( 1 , 0 )
  for templates elements :
    Text: Titre
      Grid area : title
    Image: link to https://vu.fr/vEgA , on click event clckImg
      Grid area : img
      Filling : horizontal

  Template navbar :
    Search with searchField sending to searchFcnt
```

```
Template Home :
  Grid component :
    Style :
      Border : false
      Padding : medium
      Rounding : none
      Align center : true
  columns size :
    -> small
    -> large
    -> small
  rows size :
    -> auto
    -> auto
  with following areas :
    grid area named nav will start at (row,column) ( 0 ,  0 ) and end at ( 0 ,  2 )
    grid area named timeline will start at (row,column) ( 1 ,  1 ) and end at ( 1 ,  1 )
    grid area named tend will start at (row,column) ( 1 ,  2 ) and end at ( 1 ,  2 )
  for templates elements :
    Navbar contains navbar
      Grid area : nav
    Timeline displaying post like Post
      Grid area : timeline
    Tendency displaying like Tendance
      Grid area : tend

Page :
  Page Home :
    Route / is page index true for page Home
    using template Home
```

Utilisation

Afin d'afficher le rendu de MPS sur son navigateur, il suffit de démarrer le projet React fourni. Pour cela, *NodeJS* est nécessaire, pour télécharger les modules npm avec la commande `npm install` et lancer le serveur local avec la commande `npm run start`.

Il faut également exécuter le script shell `watch.sh` à la racine du projet. Ce dernier permet, lors de la modification d'un script du DSL, de copier le fichier *JavaScript généré* pour le placer dans le dossier `src` du **client**. La page sera alors automatiquement rafraichi puisque le fichier a été modifié.

Le code généré ne possède aucune méthode fonctionnelle, c'est au développeur de placer le fichier des scripts à l'endroit approprié pour que l'import fonctionne. Lors de la génération du code en JS depuis *MPS*, on aura un fichier qui se structure de la façon suivante :

1. Import des librairies
2. Déclaration du thème
3. Déclarations des composants/templates
4. Déclarations des pages et du routing

À savoir

Le style n'est actuellement personnalisable que sûr les éléments *Grid* et *Image*. La stylisation des autres éléments fait partie des implémentations futures.

3 Scénarios

L'objectif de notre DSL est de permettre à un UI designer de créer un site de type réseaux sociaux comme *Reddit*, *Instagram* ou *Twitter*. Nous voulons lui offrir un maximum de point de personnalisation qu'il peut configurer à travers notre DSL.

3.1 Scénarios basiques

1. **Positionnement** : L'utilisateur à la possibilité de placer les composants primaires (*Timeine*, *Tendance*, *Navbar*) où il le souhaite sur la grille qu'il a défini. L'UI designer peut alors pour chaque composant contenant d'autres composants les agencer comme il le désire à l'intérieur de celui-ci.
2. **Style** : Afin de toujours laisser une plus grande liberté aux UI designer, nous voulons lui donner la possibilité de styliser chaque élément, par exemple arrondir les bords, transparence, etc... L'objectif à termes est de lui proposer le plus de choix possible.

3.2 Implémentation future

- **Responsiveness** : En extension aux deux points précédents, permettre de définir des modes alternatifs pour d'autres supports (smartphone ou tablette) afin d'adapter la position et le style voir la présence d'un composant dans le but d'offrir la possibilité à UI designer de personnaliser le site selon le support de l'utilisateur final.

4 Analyse critique

Notre modèle sémantique a nécessité de nombreuses retouches, car trop technique et éloigné du domaine. Nous avons donc repris notre modèle de zéro pour y inclure les bonnes fonctionnalités et abstraction en rapport avec le domaine. La syntaxe nous a également posé problème. Nous avons eu dû mal à en trouver une qui permet à l'utilisateur de faire ce qu'il souhaite de manière compréhensible. Les premières syntaxes que nous avions étaient très proche du JSON voire quasiment identique.

Même si ce DSL est voué à être utilisé par des développeurs web qui souhaitent gagner du temps, une telle syntaxe rend difficile la lecture dès que le site devient conséquent. Il aurait été probablement plus adapté de réaliser un DSL graphique de type *drag & drop*, mais la définition de notre modèle sémantique nous a pris trop de temps pour réellement explorer cette piste.

Ainsi, l'utilité de notre outil s'avère amoindri, d'autant plus que des outils tels que *WebFlow*, *Figma* ou encore *Sketch2Code* permettent de créer des sites internets ou des maquettes génériques et de manière graphique, ou par le biais d'un dessin dans le cas de *Sketch2Code*.

5 Prise de recul

Notre langage ne permet actuellement pas de réaliser des bons design de sites webs tel qu'il était prévu. Il est possible de réaliser des applications basiques, sans design aussi libre que prévu. Nous avons probablement passé trop de temps sur la réalisation du domaine modèle ce qui s'est ressenti lorsqu'il a fallu passer au développement du DSL. L'inconvénient d'avoir choisi MPS est également la répartition du travail. MPS rend difficile le travail par plusieurs personnes en même temps. Ainsi, pendant qu'un membre du groupe travaillait sur une fonctionnalité, les autres ne pouvaient que regarder et l'aider sans toucher au code.

6 Répartition du travail

Voici la répartition au sein du groupe :

- Eric
  - ✓ Réalisation du *domain model*
  - ✓ Rédaction du rapport
- Florian
  - ✓ Réalisation du *domain model*
  - ✓ Rédaction du rapport
- Clément
  - ✓ Développement du DSL en *MPS*
  - ✓ Rédaction du rapport
- Lucas
  - ✓ Réalisation du *BNF*
  - ✓ Participation au développement du DSL en *MPS*
  - ✓ Outillage
  - ✓ Rédaction du rapport
- Loïc
  - ✓ Réalisation du *BNF*
  - ✓ Outillage
  - ✓ Rédaction du rapport