

RAPPORT DE STAGE

Interface Web pour le logiciel libre Netmagis

Luigi Coniglio

du 17/05/2016 au 04/07/2016

Tuteur de stage : M. David Pierre

Enseignant Référent : M. Tajine Mohamed - UFR de Mathématique & Informatique

Établissement / Formation : Université de Strasbourg / Licence informatique

Entreprise d'accueil : Université de Strasbourg – Laboratoire ICube

Rapport de Stage

Table des matières

Remerciements.....	2
Introduction.....	3
Le laboratoire ICube et l'équipe Réseaux.....	3
Mon stage.....	4
Netmagis.....	4
Une nouvelle interface web pour Netmagis.....	6
Les outils utilisé.....	8
Les choix d'un framework.....	8
Toolchain et WebPack.....	9
Architecture de l'application.....	12
Bootstrap-lib: des composantes déjà prêtes pour chaque page.....	14
Prompters: l'interface à l'API REST.....	15
Gestion des authentifications.....	18
Internationalisation.....	19
Les pages réalisé.....	22
Conclusion.....	27
Bibliographie.....	28

Remerciements

Je tiens tout d'abord à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout particulièrement je tiens à adresser mes remerciements à mon tuteur de stage, M. Pierre David, pour son encadrement pendant le stage, le partage de son expertise au quotidien et surtout pour sa patience et sa compréhension.

D'une façon plus générale, je remercie l'équipe Réseaux du laboratoire ICube pour leur accueil.

Enfin, je tiens à remercier tout le reste des stagiaires et doctorants dans l'équipe pour leur appui et pour m'avoir laissé faire partie d'un climat d'échange d'idées et partage des connaissances.

Introduction

Dans le cadre de la deuxième année de ma licence en informatique à l'Université de Strasbourg, j'ai souhaité réaliser mon stage dans le domaine de la programmation web pour pouvoir approfondir mes connaissances à propos de nouvelles techniques dans ce secteur. En fait, aujourd'hui une connaissance des techniques de développement web est une des compétences les plus exploitables dans n'importe quel secteur de l'informatique et en conséquence aussi une des compétences les plus demandées dans le marché du travail.

Ce rapport représente le travail que j'ai effectué lors de mon stage au sein de l'équipe Réseaux du laboratoire de recherche ICube qui s'est déroulé entre le 17 mai et le 4 Juillet 2016.

Ce stage d'une durée de sept semaines a consisté à créer une maquette pour l'interface web du logiciel d'administration réseaux Netmagis. Le but principal de cette nouvelle interface était celui de contribuer à la transition de Netmagis vers une architecture du type REST. Cette migration représente aujourd'hui un des objectifs envisagés pour les futures versions de Netmagis.

À la fin de mon stage, les objectifs ont été atteints et ma contribution est aujourd'hui partie intégrante de la nouvelle version de Netmagis (en cours de développement)¹.

Mon travail en quelque chiffre:

- une centaine de commits
- plus de 3000 lignes de code
- une trentaine de composants créés avec le framework React.

Le laboratoire ICube et l'équipe Réseaux

Né de la fusion de cinq différents laboratoires, ICube regroupe 15 équipes et environ 450 personnes entre chercheurs et doctorants. Au sein du département Informatique recherche (D-IR), un des quatre départements selon lequel est divisé le laboratoire ICube, on retrouve l'équipe réseaux.

Les activités de cette équipe sont concentrées au tour de trois axes de recherche:

¹ <https://github.com/pdav/netmagis/tree/rest>

- analyse et modélisation des réseaux
- algorithmes et protocoles
- communications ubiquitaires pour la santé et l'environnement

Dans le cadre du laboratoire ICube on trouve aussi divers équipements de haute technologie gérés par l'équipe réseaux. En particulier, lors du stage j'ai eu l'occasion de voir en action, ensemble avec les autres stagiaires, la plate-forme SensLAB/FIT : un extraordinaire outil de recherche, qui permet d'effectuer des simulations relatives à des réseaux de capteurs sans fil pour en tester le comportement et en mesurer l'efficacité.

Pendant mes sept semaines de stage en tant qu'étudiantes hôtes de l'équipe réseaux, j'ai eu le plaisir d'être partie d'un environnement stimulante ou le travail, produit des expertises des différents membres de l'équipe et des compétences des stagiaires, était mélangé avec des occasions d'échange d'idées, partage des connaissances et agréables moments de détente.

Mon stage

Netmagis

Netmagis (pour Network Management Information System) est un outil d'administration réseau et comme la plupart des outils d'administration, il repose sur la nécessité d'automatiser certaines tâches, souvent pénibles.

Jusqu'en juin 2002, l'administration DNS du réseau métropolitain Osiris (qui regroupe les établissements d'enseignement supérieur et de recherche strasbourgeois pour un parc total d'environ 18 000 machines) était centralisée au Centre Réseaux Communication (CRC) de l'Université Louis Pasteur.

Depuis son déploiement au sein du réseau Osiris, l'application WebDNS dont Netmagis est le descendant direct, a permis aux environ 200 correspondants réseau d'administrer leurs portions de l'espace d'adressage et de générer automatiquement les zones DNS relatives en limitant la latence due précédemment à l'intervention du CRC dans toutes les actions administratives pour lesquelles le celui-ci n'avait pas de valeur ajoutée, en obtenant donc une plus forte réactivité. [1][2]

En 2011, avec la sortie de la version 2.0, le but initial de rédiger une application de gestion du DNS qui avait donné lieu à l'application WebDNS était déjà largement dépassée. WebDNS est renommé en Netmagis.[3]

Netmagis est donc né dans le contexte d'un grand réseau, comme le réseau universitaire strasbourgeois Osiris qui comprend plus de 400 sous-réseaux et des milieux d'utilisateurs, mais a été conçu pour fonctionner aussi sur de petits réseaux.

Pour garantir la séparation des rôles d'administration, Netmagis utilise un système de droits sur des réseaux associés à des groupes et utilisateurs.

Ce système fait en sorte que différents acteurs puissent par exemple ajouter ou supprimer des machines, alias ou adresse IP relativement à leur portion de réseaux, mais ne puissent pas modifier un réseau qui ne leur appartient pas.

Aujourd'hui Netmagis est arrivé à sa version 2.3.2 et englobe, dans une architecture modulaire, un grand nombre de fonctions comme par exemple la gestion des serveurs DHCP et la possibilité de visualiser la topologie d'un réseau. Développé par Pierre David et Jean Benoit, compte aujourd'hui diverses contributions, plus de 1.500 commits depuis la migration sur github effectuée en 2010 et est en ce moment la solution adoptée par plusieurs instituts¹ :

- Université Paris VI (Université Pierre et Marie Curie)
- Université de Versailles / St-Quentin-en-Yvelines
- Université de Caen Normandie
- Observatoire de Paris
- etc..

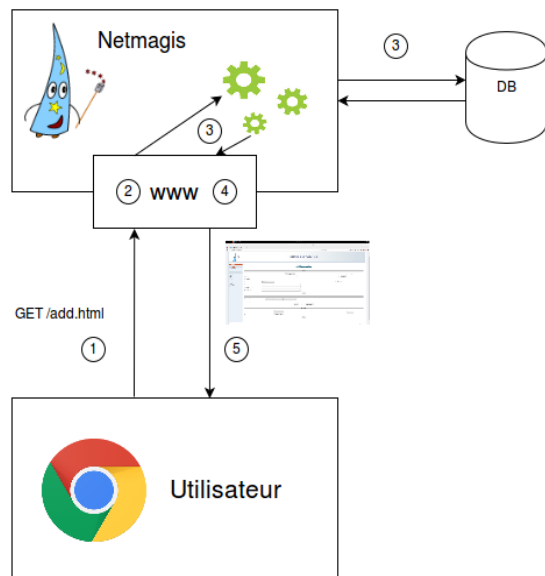
À l'heure actuelle la version majeure 3.0 de Netmagis est en phase de développement et, parallèlement à l'ajout de nouvelles fonctionnalités, poursuit comme objectif un grand changement dans l'architecture de Netmagis. En fait, comme déjà indique dans l'introduction, Netmagis est en train de migrer vers une architecture base sur un API REST dont l'utilise est subordonnée à des simples requêtes HTML. Ce passage particulièrement important ouvre la porte à une exploitation plus facile par des applications tierces des fonctionnalités de Netmagis et caractérise le contexte du stage réalisé.

Une nouvelle interface web pour Netmagis

Jusqu'à présent le principal moyen d'interagir avec les fonctionnalités de Netmagis est

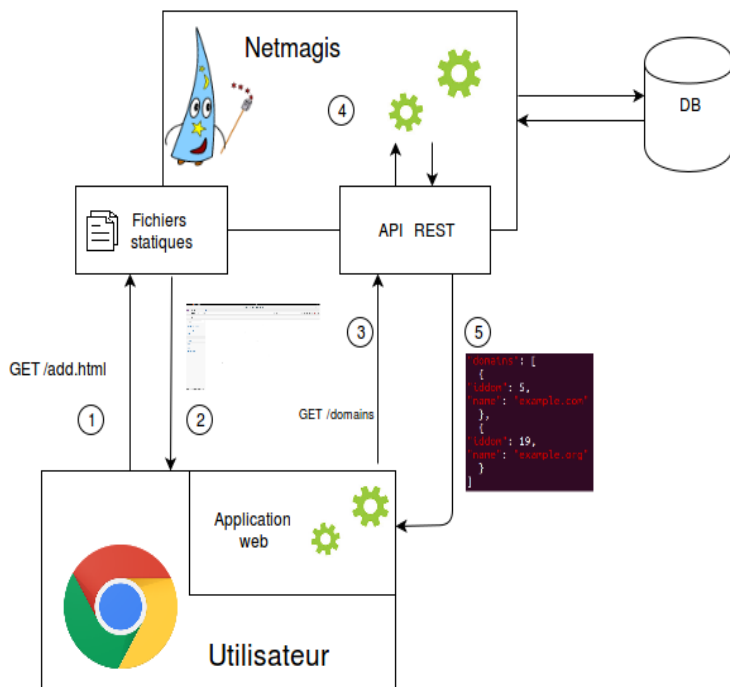
¹ Étant Netmagis une application open source il est difficile de individuer tous les utilisateurs, néanmoins, grâce à un sondage lancé par mon tutor de stage à travers la mail-list des utilisateur, j'ai pu en déterminer quelques un

à travers son interface web faisant partie du module www. Ainsi, grâce à ce module, ensuite à chaque requête de l'utilisateur l'application se charge entièrement de la génération du code de la page web demandé et, après avoir effectué les opérations nécessaires (contrôle des permissions, interrogation base de données, etc.), renvoie à l'utilisateur le résultat sous forme de page statique. Ce mécanisme est décrit, sans entrer trop dans les détails, dans l'image suivante:



1. L'utilisateur demande une page web
2. Traitement de la requête
3. Plusieurs requêtes vers la base de données et autres parties de l'application afin de récupérer les informations nécessaires
4. Traitement des données et génération de la page web
5. Réponse à l'utilisateur

Dans une optique visée à simplifier les actions du serveur Netmagis, le passage vers l'architecture REST permettra de déléguer une grande partie du travail de génération du contenu des pages web à du code exécuté côté client là où aujourd'hui est le serveur Netmagis avec le module www à se charger entièrement de la génération des pages web:



1. L'utilisateur demande une page web
2. Le serveur répond avec un fichier statiques lequel contient l'application web. Le navigateur peut déjà afficher la page.
3. L'application web demande les données nécessaires à l'API REST.
4. Netmagis s'occupe de gérer les requêtes reçu par l'API.
5. Les informations sont renvoyé à l'application sous forme de documents JSON. L'application s'occupe de générer le contenu de la page selon les informations reçu.

Cette nouvelle approche, à l'apparence plus complexe, laquelle comporte une nette séparation entre l'application web et le serveur Netmagis, offre à ce titre plusieurs avantages.

En effet, avoir une application web capable de générer du code html en raison des actions de l'utilisateur et des éventuelles réponses de l'API, va nous permettre la création d'une interface web dynamique, qui change rapidement selon les besoins de l'utilisateur et améliore en conséquence l'ergonomie de l'application. De plus, le fait de déléguer une grande partie du travail à du code exécute côté client implique un considérable allègement de la charge du travail effectuée par le serveur Netmagis.

Ce n'est pas tout: supposons que l'utilisateur veuille interagir avec l'application par exemple en ajoutant un nouvel élément dans une table. Dans l'actuelle interface de Netmagis cette action déclencherait une coûteuse mise à jour intégrale de la page (et donc la répétition de tous les passages de la première image) alors qu'avec la nouvelle architecture on pourra garder tous les éléments inchangés de la page et, après une requête à l'API, mettre à jour seulement les parties de la page concernées.

Comme on a pu voir dans l'illustration précédente, lorsqu'un utilisateur demande le contenu d'une page web, Netmagis répond d'abord en renvoyant une page statique laquelle réside en mémoire déjà prête pour être envoyé. Cette page contient seulement les éléments immuables et qui font toujours partie de la page (menu, titre,

pied de page, ...) y compris le code de l'application web.

Le résultat est donc une séparation entre le contenu statique et le contenu dynamique d'une page, le premier renvoyé tout au début, le deuxième généré dans une seconde phase par l'application web.

Or, le contenu immuable peut être maintenant mémorisé séparément par le browser donc cette séparation favorise l'utilise de la cache du navigateur web, encore un avantage sur les performances.

On peut voir comme l'unique désavantage apparente, celui de l'augment du nombre des requêtes, est largement dépassé par les avantages décrits ci-dessus. En outre, il ne faut pas oublier que la taille des données reçues lors des réponses de l'API est bien inférieure à celle des pages renvoyé dans la version actuelle de Netmagis.

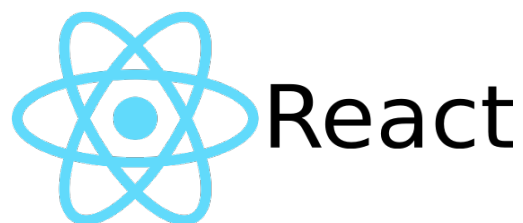
Dans le cadre de ce stage, ma mission était celle de créer une maquette pour cette nouvelle interface web qui puisse en particulier satisfaire les deux objectifs suivants:

1. garantir une certaine fluidité des actions de l'utilisateur et donner l'impression que, même si celles-ci se décomposent en plusieurs requêtes vers l'API, le résultat soit perçu à chaque fois comme une actionne unique.
2. Créer une interface plus ergonomique qui puisse guider l'utilisateur et, quand possible, fournir des raccourcis et suggestions.

Les outils utilisé

Les choix d'un framework

Pour développer une telle application web l'utilise d'un framework est presque imposée, cela surtout pour des raisons d'efficience dans le développement et dans le fonctionnement de l'application, mais aussi pour des raisons de soutenabilité du code ou même de sécurité (voir par exemple les protections contre les XSS de React). Ma mission de stage prévoyant le choix entre la version 2.0 du framework Angular, développé par Google, et la bibliothèque React, développé par Facebook.



N'ayant aucune expérience préalable dans l'utilise d'aucune de ces deux frameworks

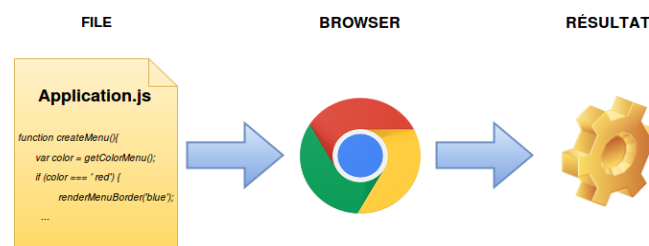
pour le développement d'une application web, la période avant le début du stage a consisté à comparer ces deux possibilités et choisir la plus appropriée pour le développement de la nouvelle interface web de Netmagis. À ce titre j'ai lu plusieurs articles à propos de ces deux possibilités et étudié leur documentation. J'ai également mis en place quelque petite application pour comprendre mieux le fonctionnement de chacune de ces deux solutions. Après ce travail d'analyse j'ai enfin choisi d'utiliser React principalement pour les raisons suivantes:

- La taille de React est environ cinq fois plus petite de celle de Angular 2
- React offre sans doute une courbe d'apprentissage beaucoup plus rapide par rapport à Angular: c'est un avantage considérable que m'a permis d'obtenir de bons résultats même dans un stage de courte durée et facilitera sans doute le travail pour ceux qui devront revenir sur mon projet après la fin de mon stage.
- Grâce à JSX, un langage qui permet de gérer les composants des applications React (et donc programme en Javascript) comme si c'était de l'XML, React est simple et agréable à la lecture.

Toolchain et WebPack

En principe une application en Javascript n'a pas besoin en soi d'aucun outil particulier pour être développée: un browser et un éditeur de texte sont normalement suffisants.

Le code, écrit par le développeur, peut être directement exécuté par le browser sans aucun passage supplémentaire.



Si ce système peut toujours être utilisé pour de petites applications, malheureusement il n'est pas la meilleure solution pour des applications plus complexes. En effet, lorsqu'il s'agit d'une application plus élaborée, on aimerait par exemple pouvoir gérer

efficacement le de code partagé entre différents fichiers de l'application ou même entre différentes applications (c'est le cas d'une bibliothèque par exemple). La plateforme NodeJS (interpréteur Javascript) offre déjà, de la même façon que les autres interpréteurs (Python, Ruby, etc.), la possibilité d'organiser un programme en modules et en importer certaines quand il est nécessaire, mais ce mécanisme n'est pas encore offert par les navigateurs web. D'ici donc la nécessité d'un outil intermédiaire pour contourner ce problème.

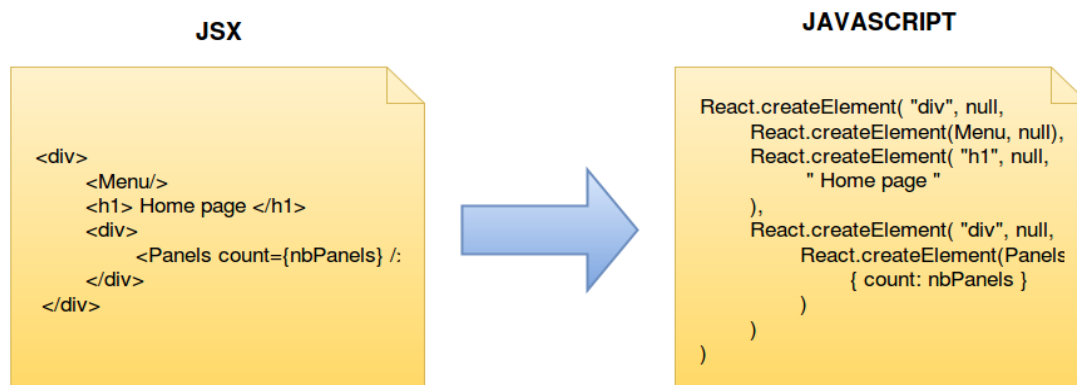
Parmi les programmes plus utilise dans ce but il y a notamment : CommonJS, Browserify et SystemJS. Lors de mon stage j'ai choisi d'utiliser WebPack pour son excellente compatibilité avec React et les nombreuses possibilités fournies sous forme de plugins.

Décrire le fonctionnement et les innombrables possibilités de WebPack c'est bien au-delà des objectifs de ces rapports, donc je me concentrerai plutôt sur la façon dont j'ai utilisé WebPack lors du développement de l'application web de Netmagis.

Dans Netmagis WebPack n'est pas utilise seulement pour résoudre les dépendances entre les différents fichiers mais il effectue aussi des actions additionnel pour améliorer l'application web dans son résultat final et en faciliter le développement:

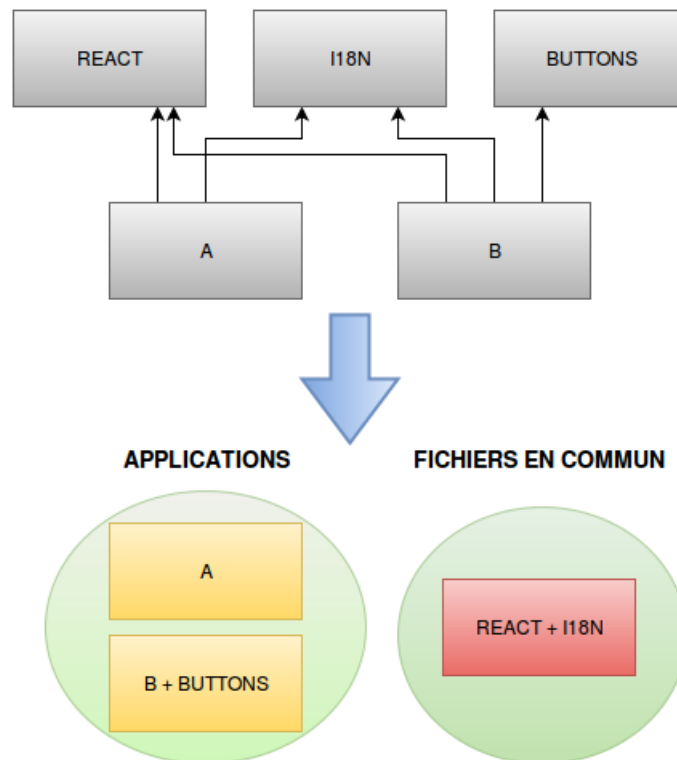
Comme j'ai mentionné précédemment, React offre la possibilité d'utiliser un nouveau langage appelle JSX, lequel permet d'écrire sous forme de XML les éléments (composants) crée avec React. Le résultat est un code beaucoup plus lisible.

Malheureusement JSX n'est pas compris pour aucun navigateur, il faut donc une étape supplémentaire pour traduire le JSX en normal Javascript habituellement lu par n'importe quel navigateur web, cette étape est aussi prise en charge par WebPack. Dans ce passage chaque tag XML est traduit en un appel à une méthode de la bibliothèque React:

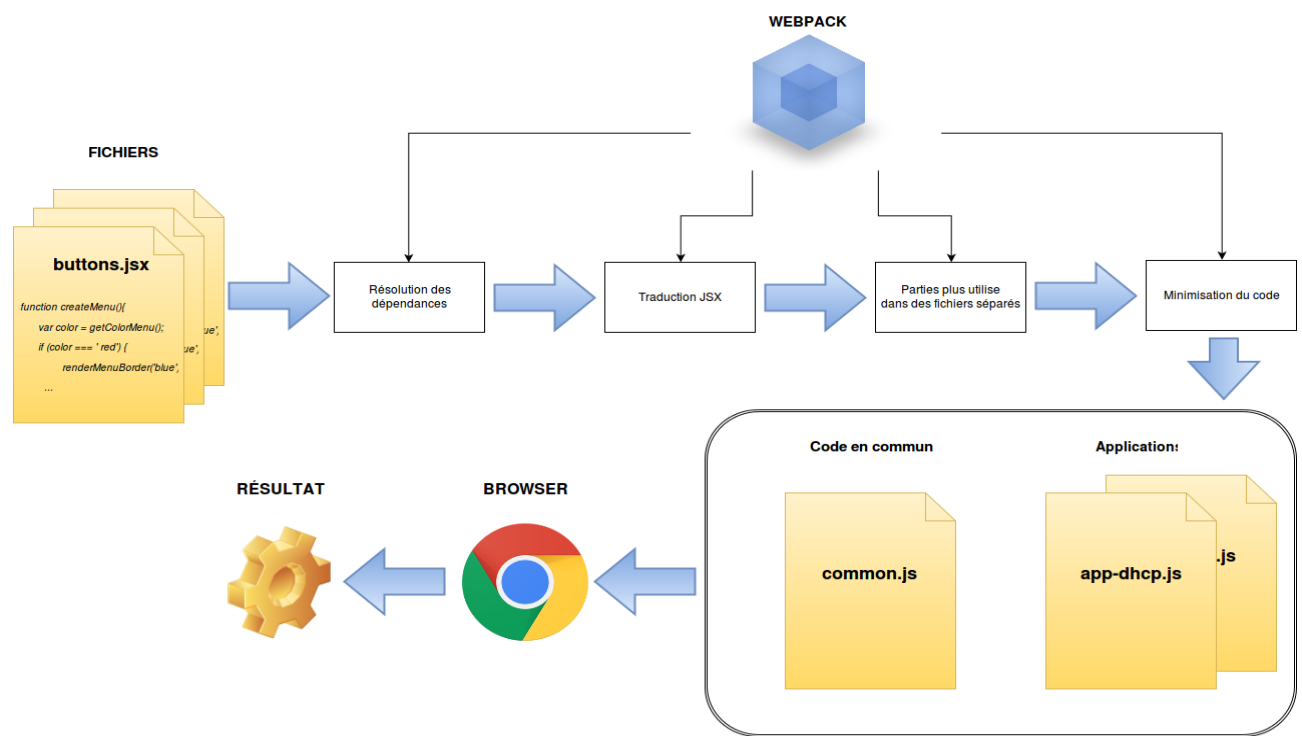


En plus, dans ce passage WebPack traduit toutes les syntaxes issues de la plus récente version de Javascript (Ecmascript 6) en sa version plus ancienne (Ecmascript 5) mais compatible avec tous les navigateurs, autrement dit on a la possibilité d'écrire du code plus moderne.

Dans une autre étape WebPack est utilisée pour mettre dans des fichiers communs toutes les dépendances demandées par plusieurs parties de l'application. Dans l'exemple suivant les fléchés représentant les dépendances. Les applications A et B dépendent de la bibliothèque React et du module I18N, seulement B a besoin de la bibliothèque BUTTONS. Un fichier contenant que les parties en commun est crée au lieu d'insérer le code en commun dans chacune des deux applications. Par contre, le code du module BUTTONS est inséré directement dans l'application B étant celle-ci l'unique qui en dépend:



Ce passage réduit la taille des fichiers et aide l'exploitation du mécanisme de cache des navigateurs. Au final, pour réduire au maximum la taille des fichiers, le code est «miniaturisé»: le nom des variables et des fonctions sont changées pour occuper le minimum d'espace, les commentaires sont éliminés, etc... .En conclusion, la chaîne de production de l'application web de Netmagis peut être résumé dans les grandes lignes par le schéma suivant:



Architecture de l'application

Pour mieux comprendre l'architecture de l'application web que j'ai développée pendant le stage, il est pratique repartir le code en trois couches: modules externes, bibliothèques internes, pages de l'application.

Modules tiers

ici on retrouve tous bibliothèques externes utilise dans le développement de l'application, notamment:

ici on retrouve toutes les bibliothèques externes utilisées dans le développement de l'application, notamment:

- React, la bibliothèque cœur de l'application web dont j'ai parlé déjà avant, utilise pour créer tous les composantes
- JQuery, une bibliothèque que j'ai utilisé surtout pour son exceptionnel API pour les requêtes Ajax (requêtes http en Javascript).
- React-autosuggest, une bibliothèque basée sur React utile pour créer des

champs d'inputs qui proposent des suggestions à l'utilisateur

Des bibliothèques internes

Dans cette partie on retrouve le code partagé par différentes pages de l'application web. Cette partie est composée par:

- Bootstrap-lib: une petite bibliothèque pour faciliter la création des éléments graphiques de l'interface web.
- Prompters: une interface qui permet à l'application web de communiquer avec l'API REST du serveur Netmagis
- Lang: pour l'internationalisation de l'application web

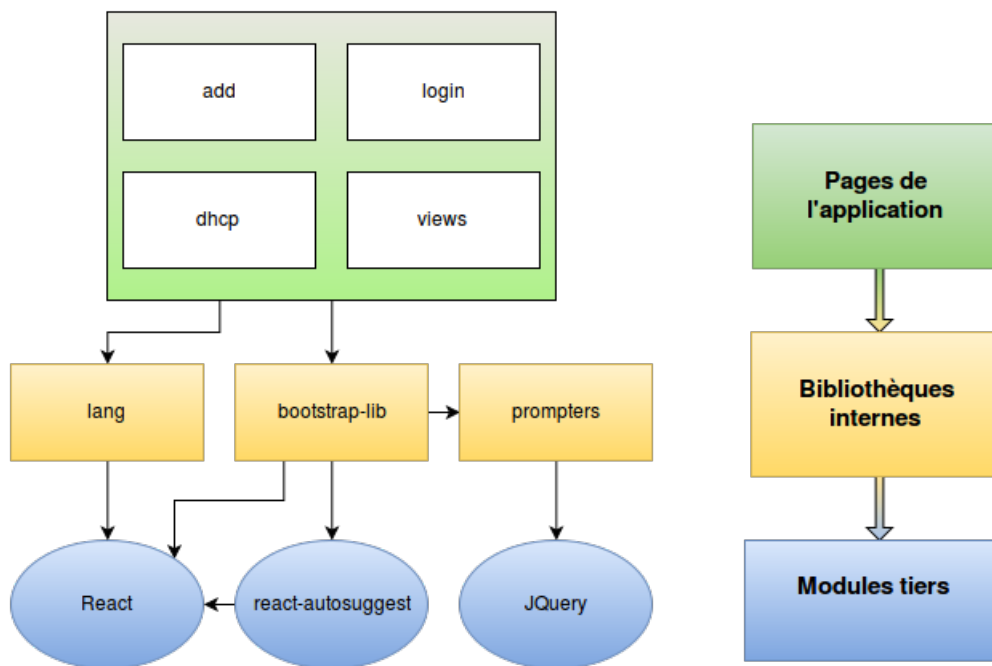
Les pages de l'application

Ici on trouve les composantes terminées de l'application web de Netmagis. Chaque page peut être considéré comme une petite application web en elle-même, elle est indépendante des autres pages et représente une interface vers une particulière fonctionnalité de Netmagis.

Si pour le moment cette structure reste plutôt opaque ne vous inquiétez pas, je parlerai des deux dernières parties plus en détail dans les paragraphes suivants.

Bien sur, tout le code sauf ceux des modules tiers, a été écrit par moi.

Pour terminer cette introduction à la structure de l'application je vous propose de jeter un œil sur le schéma suivant, lequel synthétise ce qui a déjà été dit dans ce paragraphe. Les flèches représentant les dépendances entre les différentes parties de l'application web.



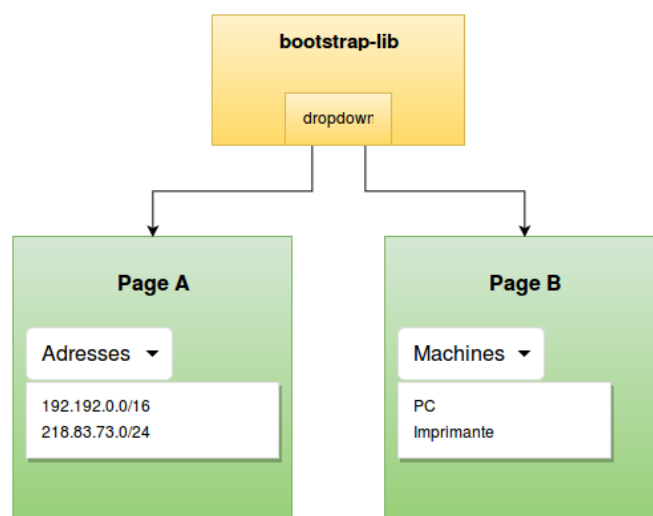
Bootstrap-lib: des composantes déjà prêtes pour chaque page

Une grande partie du travail effectué lors du stage a consisté en étudier l'ancienne interface web de Netmagis pour en comprendre le fonctionnement. La majorité des pages de Netmagis se présente sous forme d'un simple formulaire ou un ensemble de formulaires, le but de la nouvelle interface web était celui de recréer ces formulaires dans une version interactive et dynamique à l'aide de Javascript/React utilisant l'API REST pour récupérer/envoyer les données.

Chaque formulaire est composé par un ensemble d'éléments bien précis: des champs pour le texte, des boutons, des menus déroulantes, des cases à cocher, etc. ... Comme ces éléments réapparaissent plus ou moins dans tous les formulaires (même si avec un contenu différent et dans un autre ordre) j'ai voulu écrire une petite bibliothèque pour en faciliter la création.

Cette bibliothèque contient des composantes génériques code avec React. Chacune de ses composantes est codé pour être facilement adaptable à n'importe quelle page de Netmagis: si par exemple une certaine page de l'application web utilise un menu déroulant qui contient des adresses réseaux et une autre page utilise le même type de menu déroulant, mais avec des noms de machines, le code pour la création du menu déroulant n'est pas répété et grâce à l'appel à une méthode de cette bibliothèque (chaque fois avec des paramétrés différents), les deux menus sont créés sans aucun

effort.



Cette approche a permis d'utiliser les fonctionnalités des éléments plus petits pour créer des éléments plus grands et complexes.

À la fin de mon stage, cette bibliothèque contenait environ 16 éléments différents: champ d'input, menu déroulant, table, etc... . Pour ajouter un contrôle sur le style des éléments, le code HTML de chaque élément reprend les réglés du framework Bootstrap (d'ici le nom de cette bibliothèque) très utilisée dans le domaine du web design.

Prompters: l'interface à l'API REST

La majorité des éléments des pages de l'application web de Netmagis a besoin de communiquer avec l'API pour obtenir les informations à afficher et/ou envoyer des données. Cette section de l'application représente la couche de liaison entre les éléments d'une page et l'API REST, en fait celle-ci se charge de toute sorte de requête vers l'API et en communique le résultat directement aux éléments qui ont créé la requête.

Le principe de fonctionnement de cette couche est très simple: chaque élément reçoit un nom qui identifie un entre dans un dictionnaire. Cette entre est un objet qui fournit des méthodes pour communiquer avec l'API. Si par exemple une page utilise un menu déroulant que doit contenir les adresses de tous les réseaux dans la base de données, la page créera ce menu en lui donnant en paramètre le nom de l'objet

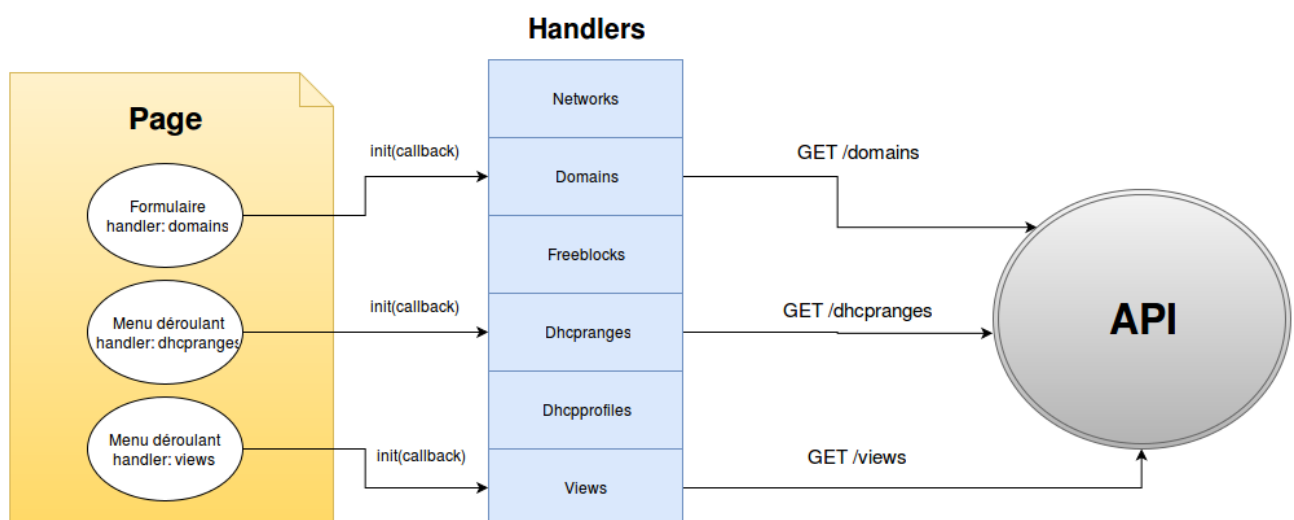
(handler) qui s'occupe de récupérer les adresses réseaux depuis la base de données.

Chaque handler fourni des méthodes préétablies, voila quelque exemple:

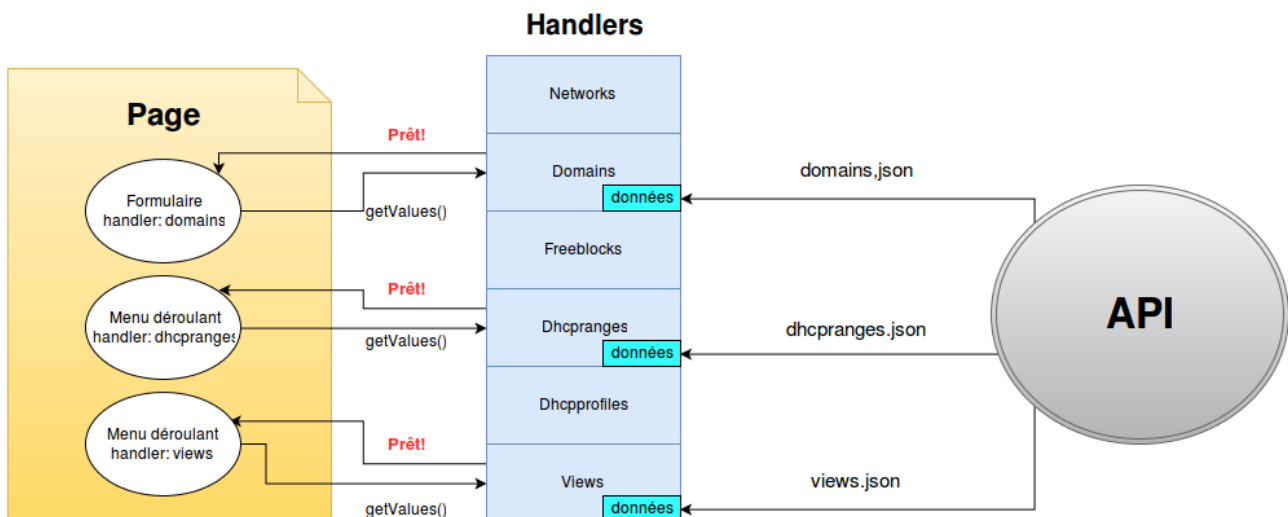
- *init(callback)* - récupère les informations depuis l'API
- *getValues()* - renvoie une liste contenant les données récupérées
- *delete(key)* - élimine l'entité identifiée par «key» dans la base de données
- etc.

Certaines méthodes sont très spécifiques à certaines composantes, comme par exemple la méthode *getEmptyRow()* qui permet à une table d'obtenir les informations nécessaires à la création d'une ligne vide.

Chaque handler doit être initialisé avant d'être utilisé pour récupérer les informations. L'initialisation se fait toujours par la méthode «init» laquelle demande les données à l'API et les mémorise à l'intérieur de l'handler. Cette méthode n'est pas bloquante, en effet (pour des raisons de performances) les requêtes sont effectuées de façon asynchrone, cela dit il faut que l'élément qui attend une réponse de l'API sache quand la réponse est arrivé, c'est pour ça que la méthode *init* accepte en paramétré une fonction qui sera appelée lorsque l'handler a été initialisé. L'approche asynchrone permet de mettre à jour parallèlement tous les éléments d'une page.



Dans la majorité des cas la fonction passée en paramètre à *init* fera elle-même un appel à la méthode *getValues* pour obtenir les données nécessaires et lancera après une mise à jour graphique de l'élément.



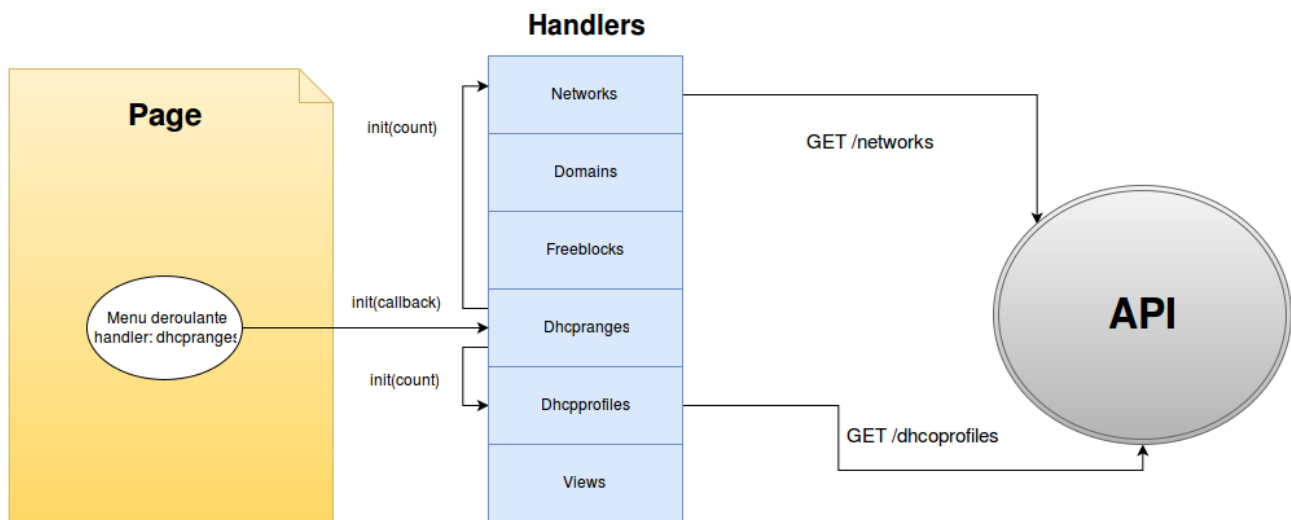
Cette approche comporte deux avantages importantes:

1 - Possibilité de traiter les données à l'intérieur d'un handler

Dans la majorité des cas les réponses fournies par l'API ne sont pas utilisables directement par les éléments de l'interface graphique. En particulier cette condition se vérifie dans les deux cas suivants:

- La réponse de l'API contient plus d'informations que celle nécessaire à l'élément. Exemple: un menu déroulant a besoin d'un tableau de chaînes de caractères mais l'API répond avec un tableau d'objets chacun contenant une chaîne de caractères et des informations supplémentaires
- L'élément a besoin de données qui doivent être dérivées de la combinaison de plusieurs requêtes de l'API.

Le schéma des requêtes sera donc, dans certains cas, plus proche à ce que montre l'image suivante:



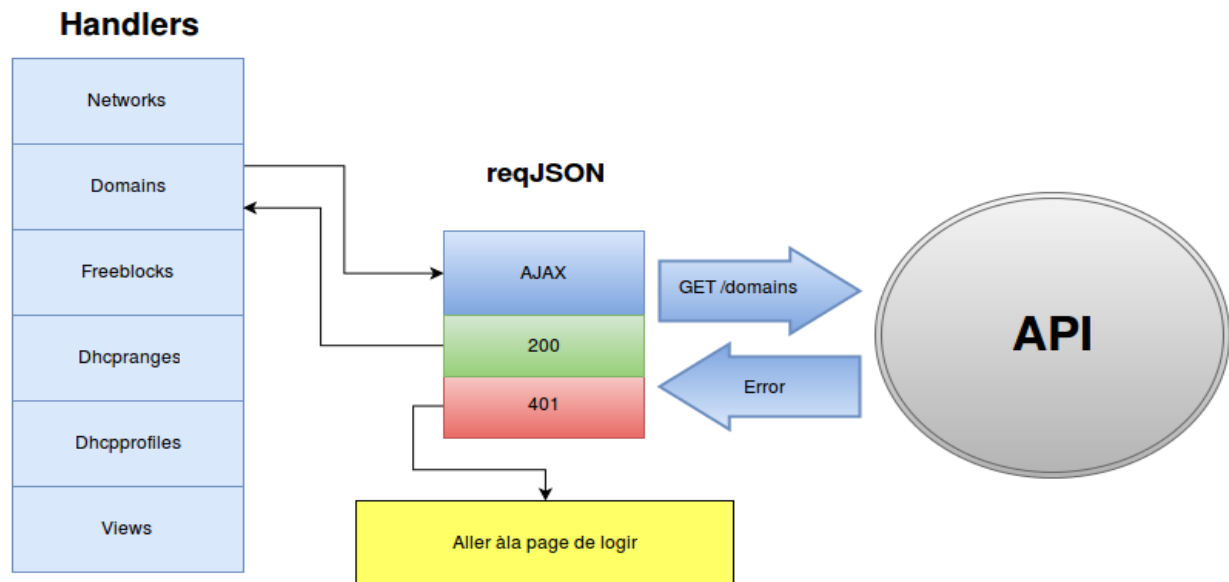
Dans ce cas, sera le premier handler (Dhcp ranges dans l'image) à devoir attendre les réponses de tous les autres. À ce titre il doit forger une fonction (*count* dans l'image) à utiliser comme callback qui à son tour devra effectuer le traitement des données et appeler la callback originale une fois que l'API a répondu à toutes les autres requêtes.

2 - Possibilité de garder les données dans le handler si celle-ci sont utilisées par plusieurs éléments

Si plusieurs éléments ont besoin directement ou indirectement des données obtenues à travers un certain handler, l'handler en question n'a pas besoin de faire plusieurs requêtes à l'API, mais une seule requête sera suffisante.

Gestion des authentifications

Chaque requête passe par une fonction (reqJSON) qui, en utilisant elle-même l'API Ajax fournie par la bibliothèque JQuery, effectue la demande d'un document du type JSON vers l'API REST. Ce passage supplémentaire permet d'ajouter des mécanismes qui seront communs à toutes les requêtes, comme par exemple la gestion des réponses. Cette technique est utilisée aujourd'hui par exemple pour rediriger l'utilisateur vers la page de login lorsque l'API renvoie le code d'erreur d'authentification 401 (c'est le cas d'un login manqué ou d'une session expiré).



Internationalisation

Le support pour l'internationalisation est une des parties les plus intéressantes de l'application. L'idée était celle de trouver un moyen simple qu'aurait permis de gérer l'internationalisation directement à l'intérieur de l'application sans devoir effectuer de grands changements dans le code interne à l'application.

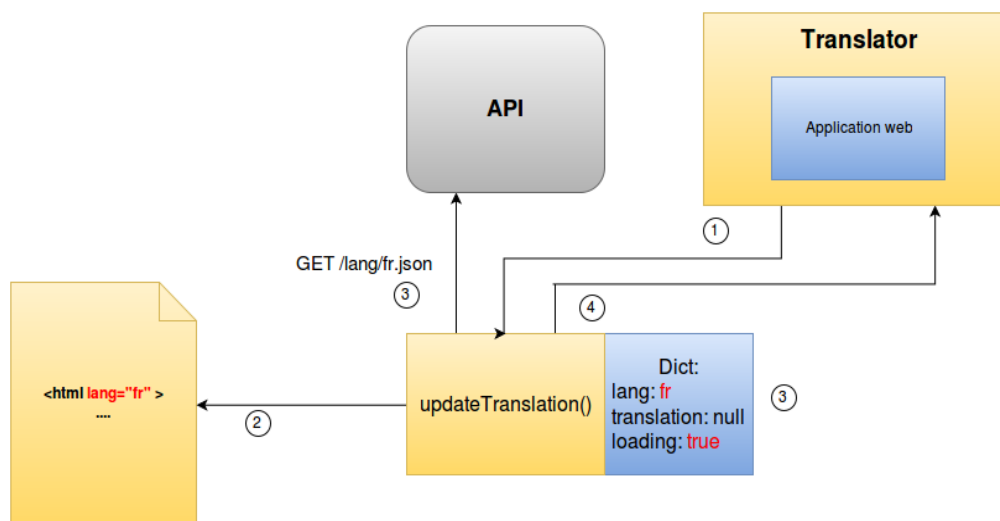
L'objectif était celui de pouvoir:

- Demander n'importe quelle page dans une des langues disponibles.
- Changer la langue d'une page en tout moment sans devoir recharger la page.
- Créer facilement des nouvelles traductions pour l'application.

La solution des problématiques ci-dessus a été mise en place sous forme d'une petite bibliothèque pour l'internationalisation où on trouve:

- *Dict*: un objet considéré comme étant le dictionnaire que contient la liste des traductions et des informations à propos de l'état actuel de celle-ci
- *updateTranslation()*: une fonction pour mettre à jour le dictionnaire
- *translate(string)*: une fonction pour traduire du texte
- *Translator*: un composant codé en utilisant React qui devra contenir (cad. être le parent) de l'application web. Celui-ci s'occupe de mettre à jour l'application

quand la langue vient de changer.



L'image ci dessus montre les actions effectuées lors du démarrage de l'application. Avant la création de l'application web *Translator* s'occupe d'appeler la fonction *updateTranslation* (1) laquelle à son tour s'occupe de mettre à jour le dictionnaire.

À ce titre il fallait trouver un moyen fiable qui aurait permis à cette fonction de connaître la langue de la page. Pour ça on a choisi d'utiliser l'attribut `lang` de la page html:

`<html lang="fr" >`

Cet attribut doit être mise en place par l'API (ou le serveur qui s'occupe de servir les pages) et s'il n'est pas présent la page est considérée comme étant en anglais (langue par défaut).

Après avoir obtenu la langue de la page (2) la bibliothèque s'occupe de mettre à jour la valeur de *Dict* et demander le fichier des traductions au serveur (3). Les traductions de l'application sont organisées sous forme de fichiers JSON et font correspondre chaque expression en anglais à sa traduction:

```
{
```

```
"domain": "domaine",  
"address": "adresse",  
"network": "resau",  
...  
}
```

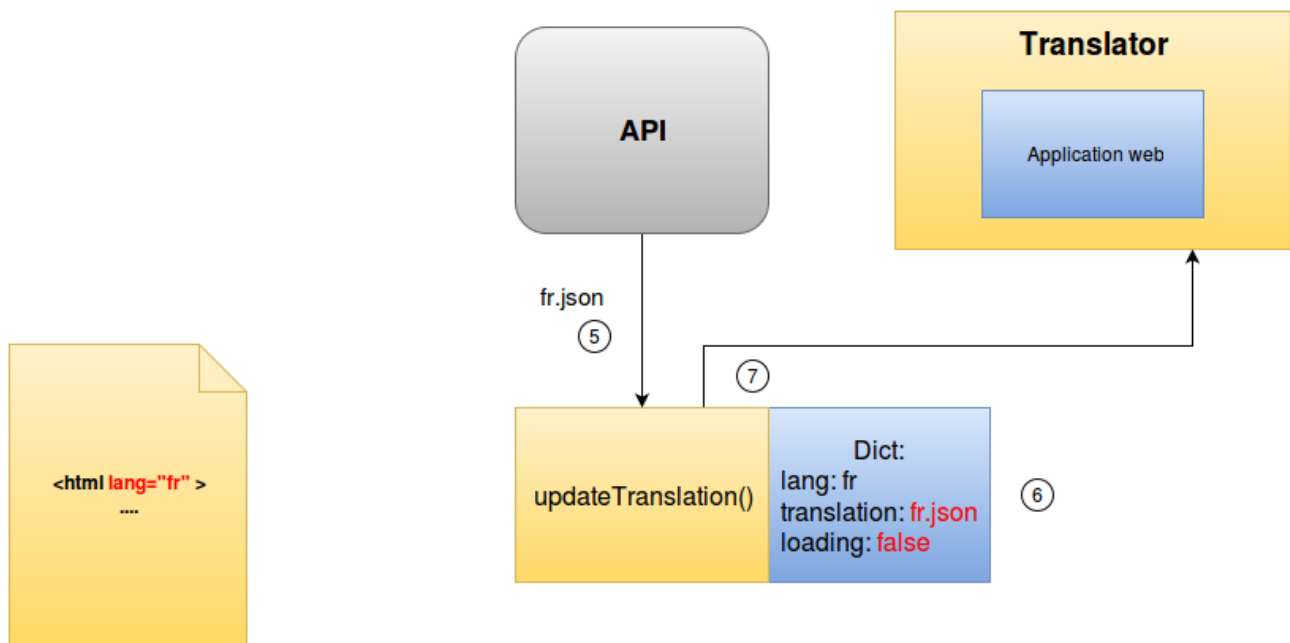
Chaque fichier contient la traduction intégrale de l'application dans une certaine langue, ce système comporte les avantages suivantes:

- L'utilisateur ne chargera le fichier qu'une seule fois, le fichier sera mis en cache et valable pour toutes les pages de l'application: le résultat est une navigation beaucoup plus agréable
- Créer une nouvelle traduction veut dire simplement créer un nouveau fichier
- Il y a beaucoup de mots et expression en commune entre différentes pages de l'application

Si la langue indique par l'attribut `lang` est anglais ou s'il n'y a aucun attribut `lang` alors l'application n'aura pas besoin de charger aucun fichier de traduction et utilisera la langue par défaut de l'application.

Enfin la fonction *updateTranslation* termine son exécution et redonne la main à l'application sans attendre le fichier des traductions (4). À ce point, pour gagner du temps, l'application peut être déjà affichée.

L'application se charge d'appeler la fonction *translate* sur tout le texte sensible d'être traduit, cette fonction ne renvoie que des espaces si la valeur *loading* du dictionnaire est vrai.



À ce stade l'application est dans un état transitoire, tous les éléments sont déjà affichés, mais le texte ne l'est pas (il y a des espaces). La plupart du temps la durée de cet état est négligeable: corresponde au temps nécessaire pour télécharger le fichier des traductions que, sauf la première fois, sera déjà en cache.

Après avoir reçu le fichier, le témoin est passé à nouveau à la fonction *updateTranslations* (5) qui, pour une deuxième fois, se charge de mettre à jour le dictionnaire (6). Enfin, il ne reste que communiquer à toute l'application de mettre à jour l'affichage (7). Maintenant, l'application est prête pour être utilisée.

Que serait-il arrivé en cas d'erreur dans le téléchargement des traductions ou si un mot n'est pas présent dans le dictionnaire ? Même en cas d'erreur l'application aurait effectué une mise à jour, mais cette fois la fonction *translate* n'aurait pas trouvé les traductions et donc renvoyé le texte par défaut: l'application reste toujours utilisable.

Les pages réalisées

La partie la plus importante de tout le travail réalisé pendant mon stage a été celui de réaliser des pages parfaitement opérants pour la nouvelle interface de Netmagis. L'importance de cette partie repose sur le fait qu'elle représente la mise en œuvre de tout ce que j'ai décrit dans les paragraphes précédents, en fait chaque page utilise les éléments susmentionnés pour construire une application plus complexe.

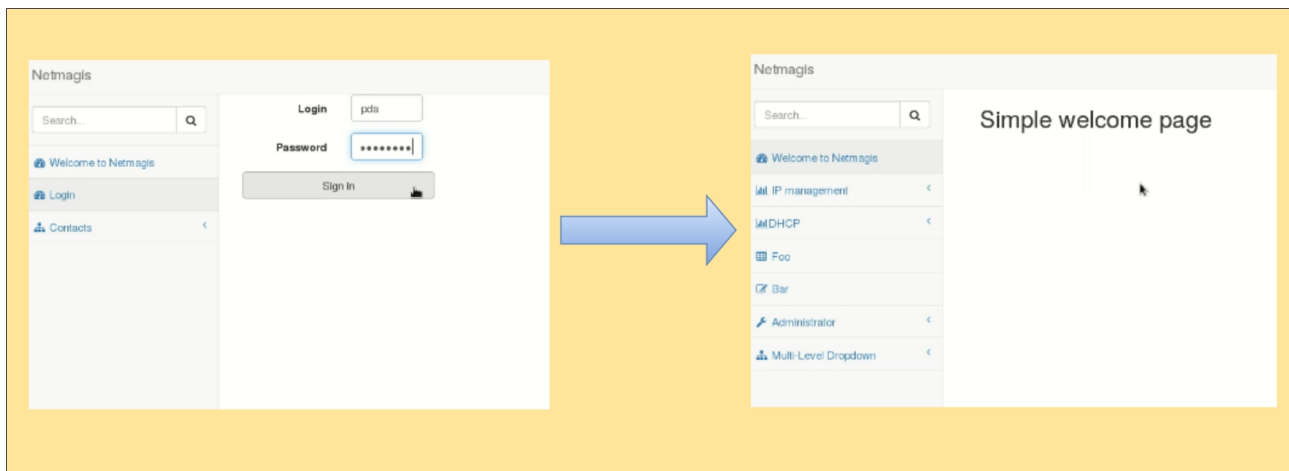
Pour maximiser ma contribution dans le développement de l'interface web je me suis dédié, sous suggestions de mon tuteur de stage, à des pages emblématiques pour

l'application: les pages dont les dynamiques résument plus ou moins tout ce qu'on peut trouver dans l'interface web de Netmagis.

Les pages réalisées sont les suivantes:

Page de login

Cette page consiste en un simple formulaire pour l'authentification. En cas de succès dans l'authentification l'utilisateur est redirigée vers la page principale de l'application, autrement un message d'erreur est affiché.



Add

Cette page fournit une interface ergonomique pour guider l'utilisateur l'adjonction d'un ou plusieurs hosts dans un réseau. Cette interface comprend par exemple des suggestions dynamiques baser sur l'input de l'utilisateur et les réseaux disponibles:

The screenshot shows the 'Add an host' form in the Netmagis application. At the top, there are two tabs: 'Add single host' (selected) and 'Add address block'. The form is titled 'Add an host' and contains several input fields and a button. The fields are: 'Name' (with a dropdown menu showing 'example.com'), 'TTL' (text input), 'Ip address' (text input with a dropdown menu showing '1'), 'Mac address' (text input with a dropdown menu showing '123.130.12.1'), 'Machine' (text input with a dropdown menu showing 'PC/Windows'), 'Comment' (text input), 'Resp. name' (text input), 'View' (text input with a dropdown menu showing 'external'), 'use SMTP' (checkbox), and 'Resp. mail' (text input). An 'Add' button is located at the bottom right of the form.

L'utilisateur peut ajouter plusieurs machines dans des adresses adjacent. Pour ça il doit d'abord chercher une plage d'adresses libres suffisamment grand:


The screenshot shows a web interface for adding many hosts. At the top, there are two tabs: 'Add single host' and 'Add address block'. The 'Add many hosts' section is active. It contains a 'Network' field, an 'or' separator, and a 'Block' field. The 'Block' field is open, showing a dropdown menu with three options: '123.130.12.33 (size: 70)', '123.130.12.33 (size: 70)', and '123.130.12.210 (size: 45)'. There is also a 'Select' button. To the right, there is a field for 'Address count' with the value '43' and a 'Search' button.

Une fois choisit un bloc d'adresses l'utilisateur peut finalement ajouter toutes les machines, sans devoir se préoccuper d'insérer toutes les informations à chaque fois. Les informations sont reportées automatiquement et l'adresse Ip incrémentée. De plus, si comme dans beaucoup des cas, les noms associés aux machines se terminent par un numéro, ce numéro est incrémenté automatiquement pour la machine suivante:

[Add single host](#) [Add address block](#)

Add many hosts

Name	<input type="text" value="subdomain2"/>	<input type="text" value="example.com"/>	TTL	<input type="text" value="1000"/>
Ip address	<input type="text" value="123.130.12.34"/>		View	<input type="text" value="internal"/>
Mac address	<input type="text"/>		<input checked="" type="checkbox"/> use SMTP	
Machine	<input type="text" value="PC/Windows"/>			
Comment	<input type="text"/>			
Resp. name	<input type="text" value="Luigi Coniglio"/>		Resp. mail	<input type="text" value="luigi.coniglio@etu.unistra.fr"/>
<input type="button" value="Add"/>				



[Add single host](#) [Add address block](#)













Add many hosts

Name	<input type="text" value="subdomain3"/>	<input type="text" value="example.com"/>	TTL	<input type="text" value="1000"/>
Ip address	<input type="text" value="123.130.12.35"/>		View	<input type="text" value="internal"/>
Mac address	<input type="text"/>		<input checked="" type="checkbox"/> use SMTP	
Machine	<input type="text" value="PC/Windows"/>			
Comment	<input type="text"/>			
Resp. name	<input type="text" value="Luigi Coniglio"/>		Resp. mail	<input type="text" value="luigi.coniglio@etu.unistra.fr"/>
<input type="button" value="Add"/>				

Dhcpranges











Cette page permet de gérer les plages d'adresses disponibles pour l'allocation dynamique d'adresse ip via serveur Dhcp. Ici l'interface est représentée par une table ou chaque ligne peut être éditée séparément. L'utilisateur peut choisir le réseau cible en utilisant un menu déroulant et gérer les plages d'adresses (lignes) déjà présentes dans la table ou en ajouter des nouveaux. Chaque action de l'utilisateur se traduit en une requête vers l'api.

Select a network172.16.0.0/16

Min	Max	Domain	Default lease duration	Maximum lease duration	DHCP profile	
172.16.10.200	172.16.10.204	example.com	0	0		 
172.16.10.205	172.16.10.206	example.org	0	0		 
<input type="text" value="172.16.10.210"/>	<input type="text" value="172.16.10.211"/>	example.com	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value=""/>	 
172.16.10.213	172.16.10.213	example.com		0		 
172.16.10.214	172.16.10.214	example.com	0	0		 
<input type="text"/>	<input type="text"/>	example.com	<input type="text"/>	<input type="text"/>	<input type="text" value=""/>	 

+

Select a network172.16.0.0/16

Min	Max	Domain	Default lease duration	Maximum lease duration	DHCP profile	
172.16.10.200	172.16.10.204			0		 
172.16.10.205	172.16.10.206			0		 
172.16.10.210	172.16.10.211	example.com	0	0		 
172.16.10.213	172.16.10.213	example.org	0	0		 
172.16.10.214	172.16.10.214	example.com	0	0		 

+

Views

Cette page offre à l'administrateur une table pour la gestion des views DNS disponibles et leur ordre de tri. La table est très semblable à celle de la page `dhcpranges` dans son utilisation, mais le fonctionnement interne est complètement différent. En fait, contrairement à l'application précédente, ici Netmagis ne traite pas chaque ligne comme étant une entité en soi. Pour cette raison à chaque modification l'application communique tout le contenu de la table à l'API et pas seulement la ligne modifié. Cette différence n'est pas perceptible par l'utilisateur.

Sort class	Name	Select def		
100	external	<input type="checkbox"/>		
300	internal	<input checked="" type="checkbox"/>		
	external	<input type="checkbox"/>		

Conclusion

L'expérience de ce stage a été très enrichissante pour moi. J'ai eu, tout d'abord, la chance de connaître mieux le travail effectué à l'intérieur d'une équipe de recherche. Lors du développement de l'application web de Netmagis j'ai eu l'occasion de vivre personnellement les enjeux qu'il y a dans la conception d'un logiciel.

Entre les difficultés auxquelles j'ai dû faire face certaines ont été particulièrement formatives comme celle de réussir indépendamment à évaluer, choisir et utiliser des technologies pour moi nouvelles tel que React ou Web Pack dans un temps restreint.

Devoir développer une application autour d'une API en cours de développement (et donc avec des parties pas encore définies et/ou documenté) m'a appris à implémenter des bibliothèques génériques dont le fonctionnement est adapté à toutes les nécessités.

Un des défis principaux par rapport à l'application web a été celui d'utiliser une interface, celle de l'API REST de Netmagis, dont le fonctionnement ne correspondait pas très souvent à ce qui aurait été optimal pour l'application web (ce que se traduit par exemple, dans certaines pages, par l'enchaînement de plusieurs requêtes vers l'API pour obtenir toutes les informations nécessaires).

Les objectifs du stage, consistant dans la création d'une ou plusieurs maquettes pour l'interface web de Netmagis, ont été atteints: a cet égard je suis particulièrement fier d'avoir apporté ma contribution dans l'évolution d'un logiciel duquel j'apprécie les nombreuses fonctionnalités.

Bibliographie

[1] P. David et J. Benoit. Une application pour décentraliser la gestion du dns. Dans JRES2003, Lille, 2003

[2] P. David et J. Benoit. Le système d'information du réseau Osiris: de la fibre optique jusqu'aux services. Dans JRES2005

[3] P. David, J. Benoit et S. Boggia. Gérer son système d'information réseau avec Netmagis. Dans JRES2015