Table des matières

Table des matières 1

1. Introduction 2

2. Modèle MVC 4

3. Installation des outils et initialisation du projet 5

3.1. Node.js et npm (node package manager) 5

3.2. Installer Gulp 5

3.3. Initialiser un dossier de projet SemanticUI avec npm 6

3.4. Lancement et monitoring du script Controleur : utilisation de pm2 6

3.5. Outils de dev 6

4. Fonctionnement de Semantic UI 7

4.1. Principe 7

4.2. Personnalisation 8

4.2.1. Thèmes et modification de la charte graphique 8

4.2.2. Liste des fichiers modifiés 9

5. Description de l'arborescence de fichiers du projet 9

5.1. Fichiers et dossiers principaux 9

5.2. Dossier «interface» 11

5.2.1. Css 11

5.2.2. img 11

5.2.3. js 12

5.2.4. videos 14

5.2.5. views 14

5.3. Dossier «node_modules» 17

5.4. Dossier «scripts» 18

5.5. Dossier «semantic» 19

6. Description du template générique (valable pour toutes les vues) 19

7. Sécurité et mise en production 20

7.1. Ouverture des liens 20

7.2. Sécurisation des formulaires 20

7.3. Sécurisation de la base de données GAMeRdb 20

7.4. Utilisation d'un reverse proxy (non effectué, conseillé.) 21

1. Introduction

L'introduction n'est qu'un récapitulatif des solutions front-end et back-end adoptées parmi celles disponibles. **Elle peut-être donc passée.**

Technologies front-end :

HTML5, CSS3 et JavaScript (ES6) ont été choisis d'emblée. Des scripts **JQuery** seront utilisés pour exploiter les tableaux dynamiques « **Datatables** ». Pour économiser le temps accordé à la modélisation des pages du site web, un framework front-end a été choisi : **Semantic UI**. Celui-ci porte bien son nom car les variables utilisées sont très parlantes. Ainsi, par rapport à d'autres frameworks, Semantic UI est facile à prendre en main, de plus celui-ci est très personnalisable et modulable. Pour exploiter SemanticUI en routine (compilation du code **LESS** automatique notamment), **Gulp** a été utilisé : celui-ci nécessite d'installer **NodeJS** à ces cotés pour fonctionner, ce qui n'est pas un problème car on l'utilisera aussi en dehors des développements pour la partie en back-end.

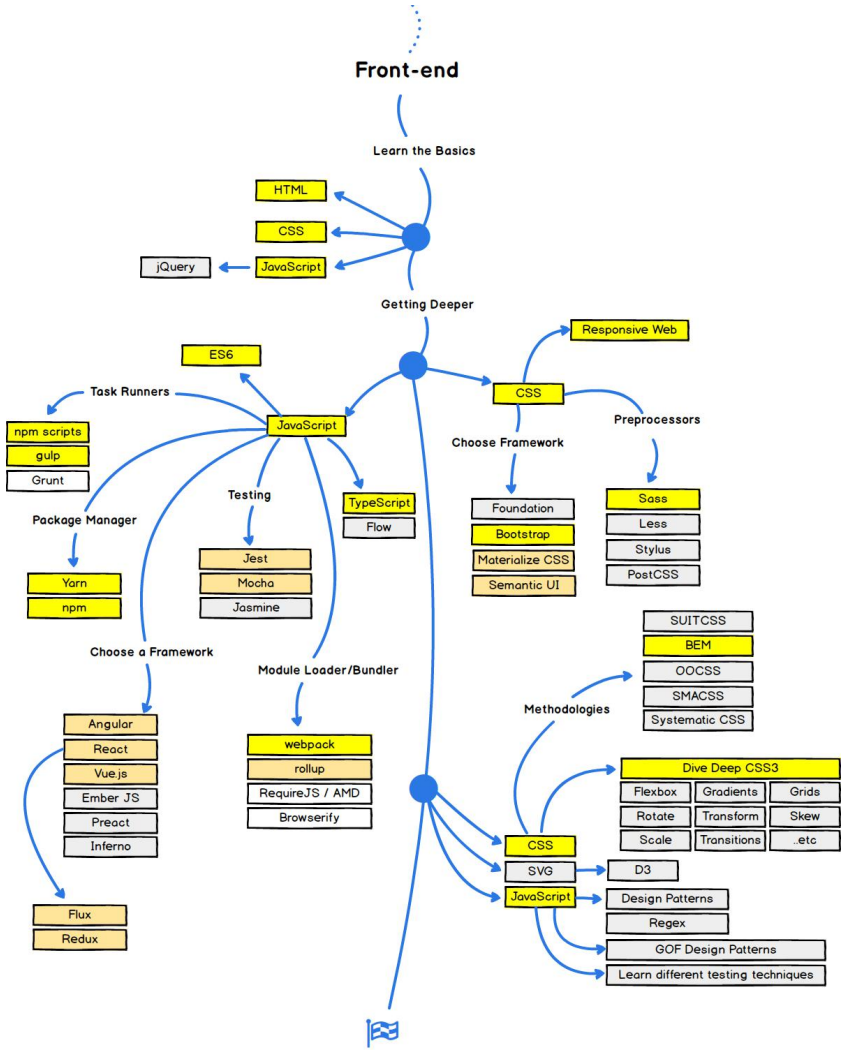


Figure 1 - Tirée de . Panel des technologies front-end en vogue. Jaune : recommandation personnelle, orange clair : conseillé, gris : autres possibilités.

Technologies back-end :

La technologie **NodeJS** a été choisie pour la gestion de la partie back-end (partie modèle et contrôleur du schéma MVC). L'exploitation de **NodeJS** (création du serveur, gestion des routes, génération des vues) aurait pû etre facilitée par l'utilisation du framework **ExpressJS** mais à été abandonnée en cours de route pour des raisons techniques ET pédagogiques . **Npm** (Node package manager, équivalent du pip de Python) a été installé pour la gestion des packages node (Gulp, ExpressJS…) La base de données **MongoDB** exploite le paradigme **NoSQL**.

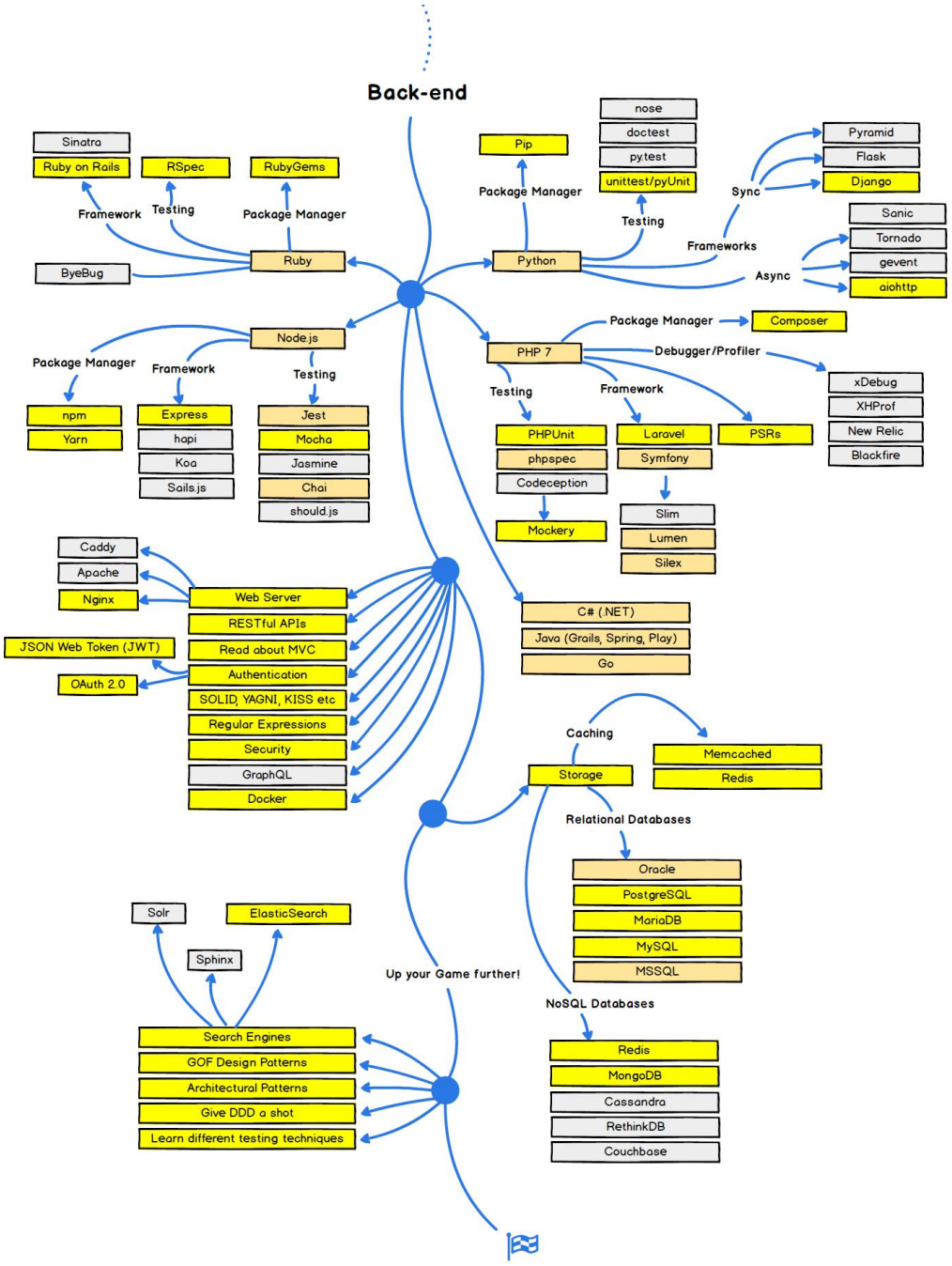


Figure 2 - Tirée de . Panel des technologies back-end en vogue. Jaune : recommandation personnelle, orange clair : conseillé, gris : autres possibilités.

2. Modèle MVC

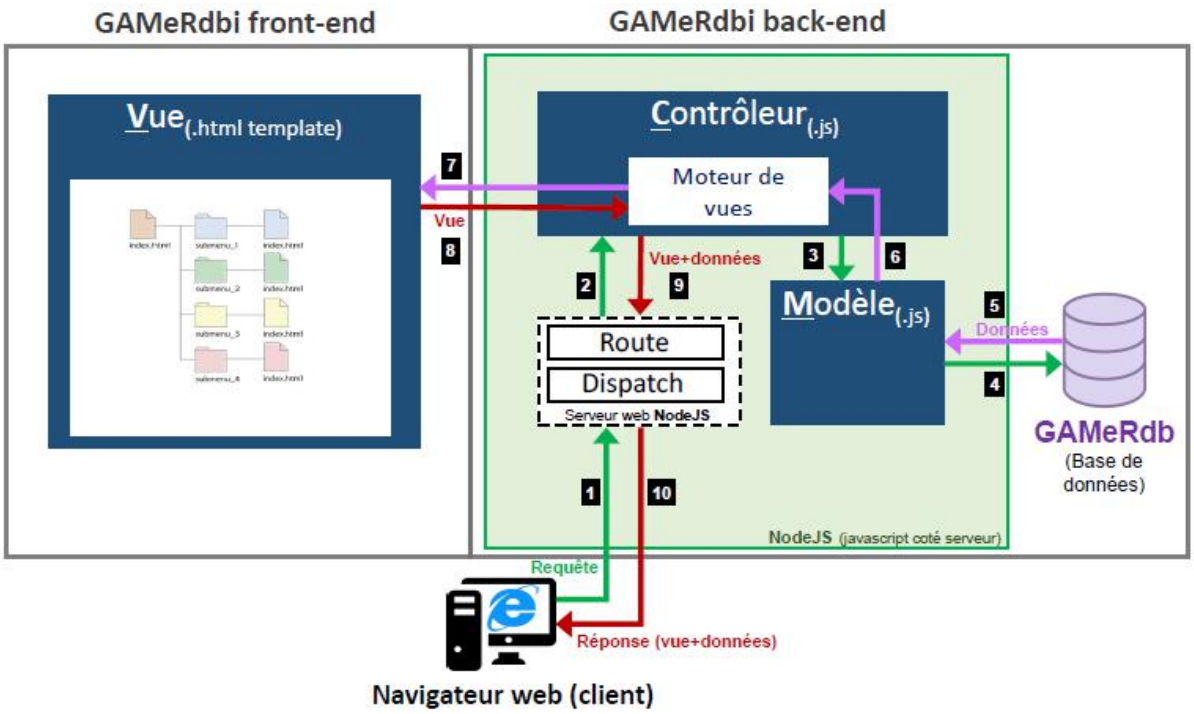


Figure 3 - Workflow simplifié du schéma MVC de l'application web

VUE : Template HTML facilement personnalisable à l'aide de frameworks front-end tels que Semantic UI. Après requête et reponse http la vue correspond à la présentation des donnés via le navigateur web.

MODELE : Principe similaire à une API. Le modèle contient exclusivement le code pour interroger les bases de données et les renvoyer à l'utilisateur.

CONTROLEUR : Couche logique(traitements) Fait le lien entre le modèle et la vue. Contient tout le raisonnement du code hors requêtes à la base de données.

Chemin de la requête http (cinématique) :

1. Requête http puis répartition et routage des paquets dans le serveur NodeJS encapsulé par ExpressJS
2. Traitement de la requête http dans le contrôleur (Controller.njs)
3. Transfert de la requête http vers le modèle (Model.njs) qui est exclusivement chargé d'effectuer les requêtes aux bases de données
4. Requête NoSQL dans le modèle
5. Récupération de la réponse NoSQL associée a la requête (données) par le modèle
6. Transformation des données au format HTML dans le moteur de vues (Views.njs) du contrôleur

7. Fusion de la réponse HTML avec le template HTML Semantic UI correspondant récupéré par le moteur de vues
8. Récupération de la vue mise à jour (fusionnée)
9. Utilisation des fonctionnalités route et dispatch (optionnel)
10. Réponse http contenant la vue mise à jour demandée.

En résumé, dans le modèle MVC implémenté : Controller.njs contrôle les scripts Views.njs et Model.njs qui ne peuvent s'éxecuter seuls. Views.js est chargé de modifier les vues (templates html) et interroge pour celà Model.js mais JAMAIS de manière autonome (car contrôlé par Controller.js). Model.njs est chargé d'interroger les bases de données mais JAMAIS de manière autonome. (idem)

Note : dans l'implémentation actuelle du script, **ExpressJS n'est plus utilisé, le principe reste le même.**

3. Installation des outils et initialisation du projet

3.1. Node.js et npm (node package manager)

Installer la version 8x qui est LTS (Long-term Supported). **En installant node.js, npm est installé automatiquement**

```
#Depuis les pc Ubuntu :
curl-sL https://deb.nodesource.com/setup_8.x |sudo-Ebash-
sudo apt install -y nodejs

#Depuis le serveur CentOs :
wget https://nodejs.org/dist/v8.9.4/node-v8.9.4.tar.gz
./configure
make -j 4
```

3.2. Installer Gulp

Gulp permettra de compiler le code LESS de SemanticUI. Cette tâche pourra être aussi effectuée en routine automatiquement à chaque modification des fichiers de SemanticUI en activant le mode « surveillance » de Gulp (plus de détails en partie 3) .

Ligne de commande pour l'installation de Gulp depuis les serveurs de l'ANSES :

```
#Depuis les pc Ubuntu :

sudo -E npm --proxy http://user:pwd@alfortproxy.afssa.fr:8080 --without-ssl --insecure -g install gulp

#Depuis le serveur CentOs :

sudo -E npm --without-ssl --insecure -g install gulp
```

Ligne de commande pour l'installation sans proxy depuis PC personnel :

```
sudo npm  -g install gulp
```

3.3. Initialiser un dossier de projet SemanticUI avec npm

```
#Initialiser npm (evite des problemes bizarre de path ensuite). genere un fichier package.json

>npm init

#Installer semantic-ui

>npm install --save semantic-ui

#Compilier fichiers LESS pour creer un dossier dist avec js et css files

>cd semantic/

>gulp build
```

Gulp build permet de générer les fichiers CSS et JS dans un **nouveau dossier qui est créé : le dossier « dist »**

3.4. Lancement et monitoring du script Controleur : utilisation de pm2

Pm2 est un outil de monitoring qui permet de lancer des scripts en tant que service (daemon) : les processus sont donc relancés en cas de crash. Pm2 permet aussi de surveiller les ressources exploités par les scripts lancés avec pm2. Il apporte aussi la possibilité d'effectuer de la journalisation applicative (surveillance et/ou sauvegarde des logs).

Dans le cadre de ce projet, c'est Controller.js qui est lancé et monitoré par pm2. Controller.js écoute par défaut à'l'adresse suivante :

var listenPort = 3000;

var listenIp = 'sas-pp-lscalc1';

```
#installation globale de pm2 sur la machine :
sudo npm install pm2 -g
# Lancer le script controller.js en tant que «GAMerdbi»
pm2 start Controller.js  --name GAMeRdbi
# Lister tous les processus lancés via pm2
pm2 list
# Surveiller les logs de tous les processus lancés par pm2
pm2 monit
# Stopper GAMerdbi (transparent pour le client, requêtes traitées avant extinction)
pm2 stop GAMerdbi
# Relancer GAMerdbi (transparent pour le client, requêtes traitées avant extinction)
pm2 restart GAMerdbi
# Supprimer le processus GAMerdbi
pm2 delete GAMeRdbi
```

3.5. Outils de dev

Brackets (http://brackets.io/) pour l'IDE, avec les modules suivants: Brackets Git, CSS linter, Accessibility (W3C linter), Color picker, Bracket themes, Bracket File icons, Visible tabs, Indent guide, Bracket trees.

Firefox web developer : https://addons.mozilla.org/fr/firefox/addon/web-developer/

GitFlow : https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Gulp (décrit en 2.2)

4. Fonctionnement de Semantic UI

Pour plus de détails (par ordre de complexité/niveau de détail):

4.1. Principe

L'utilisation d'un framework front-end permettra mettre en place l'interface sur des bases solides tout en réduisant le temps de développement. Semantic UI se veut, comme son nom l'indique, moins programmatique, davantage naturel, les variables intégrées au framework sont alors très parlantes et organisées autour d'une véritable sémantique : par exemple, pour passer un une image centrée portera le nom de classe « ui centered medium image », redimensionnée (small) et non centrée « ui small image ».

L'arborescence des fichiers de SemanticUI est la suivante :

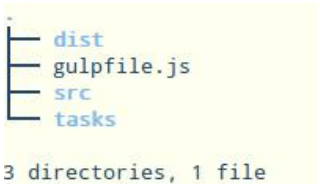


Figure 4 - Arborescence des fichiers du dossier semantic

Le dossier « dist » contient le code CSS et JS obtenu après compilation du code LESS (voir partie 3.2). Le dossier src contient les fichiers de SemanticUI éditables pour toute personnalisation de la charte graphique de base. Ce sont des fichiers à l'extension «**. variables** » qui contiennent des variables de pré-processing du code CSS (**code LESS**) lors de sa compilation, ou encore à l'extension «**.overrides** » qui permet d'ajouter une surcouche au CSS compilé (**code CSS**). Le dossier tasks contient les fichiers nécessaires au fonctionnement de Gulp.

4.2. Personnalisation

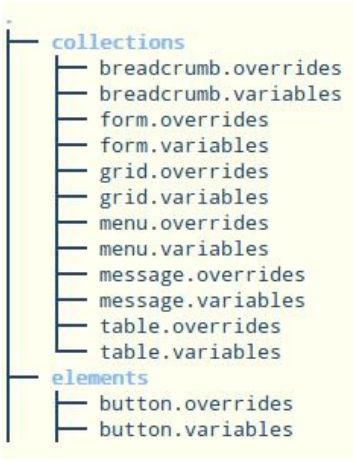


Figure 5 - Arborescence de fichiers d'un dossier de themes de semantic

Les modifications effectuées sur ces fichiers peuvent être surveillées par Gulp, qui compilera automatiquement un nouveau code CSS et JS, à chaque modification d'un fichier « overrides » ou « variables » (autobuild). Pour lancer cette commande il faut se trouver dans le dossier semantic et lancer la commande « gulp watch » :

```
>cd semantic/ #si pas déjà fait
>gulp watch
```

4.2.1. Thèmes et modification de la charte graphique

SemanticUI permet de modifier ses éléments graphiques de diverses manières.

Thèmes

Tout d'abord, plusieurs thèmes sont disponibles : amazon, basic, bookish, boostrap3, chubby, classic, colored, default (thème utilisé par défaut), duo, github, gmail, material, et 10 de plus fournis par défaut…

Composants

Ces thèmes peuvent être appliqués à l'intégralité du futur site web ou limités à certains composants. Il existe 4 types de composants dans SemanticUI :

- **Element :** boutons, icones, containers, zones de texte et autres. Ce sont des composants qui peuvent être groupés / additionnés dans une page.
- **Collection :** Ce sont des regroupements de composants de type Element (formulaire, menu, etc.)
- **Views :** ce sont des Collections spécifiques inspirées de sites spécifiques et connus et présentés de la même manière (fil d'actualités, commentaires).
- **Modules :** ce sont des composants dont leur rôle est entièrement défini par leur nom. La plupart sont dynamiques

Modification de la charte graphique

Le rendu du site web utilisant SemanticUI peut donc être modifié de diverses manières :

- En modifiant les thèmes utilisés pour chaque module dans le fichier site.variables contenu dans le dossier semantic/src/site/globals
- En modifiant le rendu final du site en ajoutant des règles supplémentaires au CSS dans le fichier site.overrides contenu dans le dossier semantic/src/site/globals
- En effectuant des modifications pour un module (sans distinction de thème) en modifiant les fichiers overrides et variables contenus dans le dossier semantic/src/site/nom_du_module
- En effectuant des modifications pour un module d'un thème spécifique en allant modifier les fichiers nom_du_module.variables et nom_du_module.override contenu dans le dossier semantic/src/themes/nom_du_theme/nom_du_module.

4.2.2. Liste des fichiers modifiés

Dans chaque fichier, les lignes ajoutées ou modifiées sont précédées d'un commentaire explicatif.

```
semantic/src/site/globals/site.overrides : ajout de règles CSS
semantic/src/site/globals/site.overrides : modification des couleurs par défaut
semantic/src/site/elements/header.variables : modification des couleurs par défaut (couleur du texte du header lors de son survol)
semantic/src/site/collections/menu.variables : modification des couleurs par défaut (couleur du texte du menu: actuel header)
semantic/src/site/elements : modification des couleurs par défaut.
```

5. Description de l'arborescence de fichiers du projet

5.1. Fichiers et dossiers principaux

On trouve à la racine du dossier GAMeRdbi 4 types de fichiers :

- Fichier readme au format Markdown
- Licence Github par défaut
- Les fichiers json de configuration de npm (package.json).
- Les fichiers json de configuration de SemanticUI (semantic.json)

On trouve à la racine du dossier GAMeRdbi 4 dossiers principaux:

- **interface** : tous les fichiers front-end de l'application web exception faite de SemanticUI (pour des raisons techniques)
- **node_modules** : tous les fichiers de dépendances npm propres au projet
- **scripts** : tous les fichiers back-end de l'application web (scripts, fichier serveur, raccourcis vers le NAS)
- **semantic** : tous les fichiers de base et de configuration de SemanticUI

Figure 7 - Arborescence de fichiers détaillée du dossier GAMeRdbi

La suite du chapitre 4 va consister à détailler dans ces 4 dossiers principaux le rôle de chaque fichier/groupe de fichiers et/ou les fonctionnalités qui y ont été implémentées.

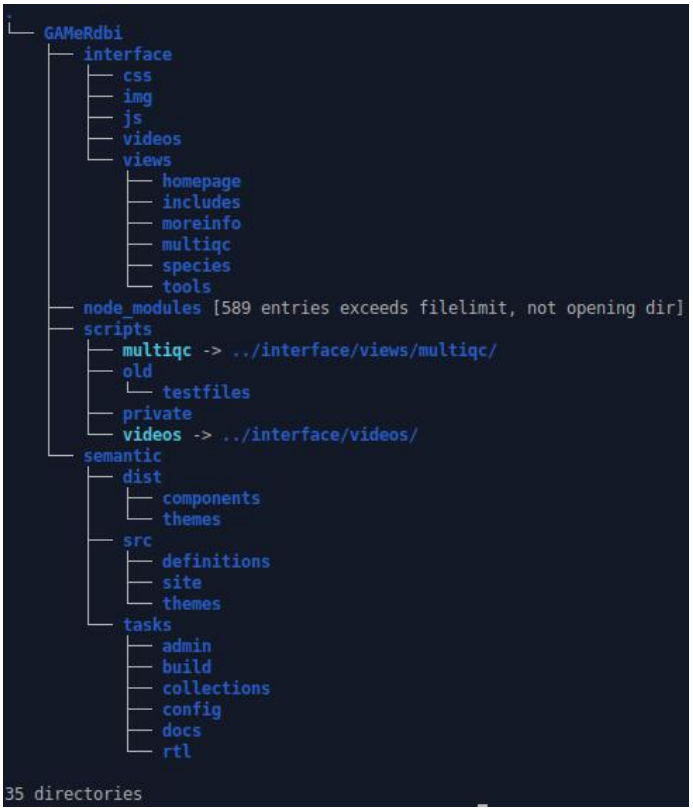


Figure 8 - Arborescence de fichiers détaillée du dossier GAMeRdbi

Code couleur pour la suite:

Bleu : fichier externe fourni par les créateurs d'une bibliothèque et code source non retouché.

Noir : fichier crée en interne.

5.2. Dossier «interface»

Contient tout les fichiers destinés au front-end de l'application web.

5.2.1. Css

Contient tout le code CSS qui sera utilisé côté client.

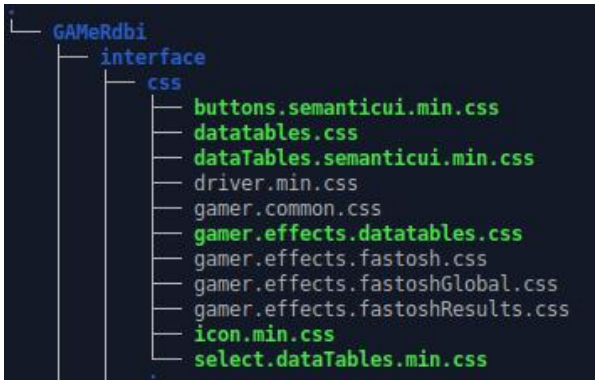


Figure 9 - Arborescence de fichiers détaillée du dossier css

buttons.semanticui.min.css : Initialisation et mise en forme de base des boutons dans semanticUI.

datatables.css : Initialisation et mise en forme de basede tableaux dynamiques «DataTables» .

dataTables.semanticui.min.css : Pour l'intégration de boutons au style semanticUI dans les dataTables.

driver.min.css : utilisé pour mettre en surbrillance des éléments de la page web pour les tutoriels intéractifs.

gamer.common.css : Mise en forme d'élements présents dans plusieurs vues (pages html) de l'application web. Evite d'avoir à répéter le même code pour plusieurs pages.

gamer.effects.datatables.css : Mise en forme commune des pages exploitant datatables. Evite d'avoir à répéter le même code pour plusieurs pages.

gamer.effects.fastosh.css : Mise en forme de la page d'accueil de l'outil «Fastosh».

gamer.effects.fastoshGlobal.css : non exploité.

gamer.effects.fastoshResults.css : mise en forme des pages «Phylogeny» et des pages de résultats d'exécution de Fastosh.

icon.min.css : Intégration des packs d'icones de SemanticUI

select.dataTables.min : Intégration d'élements de personnalisation pour la mise en forme de la selection dans les dataTables.

5.2.2. img

Contient toutes les images du site qui sont appelées dans les vues.

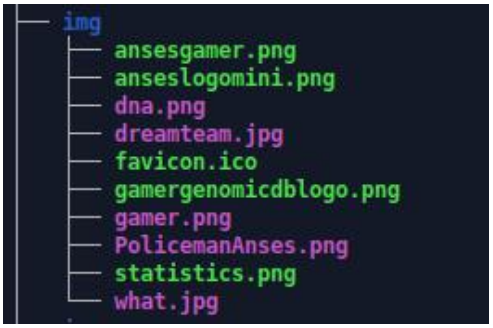


Figure 10 - Arborescence de fichiers détaillée du dossier images

5.2.3. js

Contient tout le code Javascript additionnel non spécifique aux frameworks (Semantic UI, NodeJS…). Par exemple le code JS Jquery qui sera utile à l'exploitation des Datatables. La plupart des scripts contenus dans ce dossier nécessiteront d'être exécutés **après chargement du DOM**, l'atrribut « defer » est utilisé pour cela lors de leur déclaration dans les balises <script></script>. (une seconde solution aurait pu consister à placer les scripts en fin du code html, mais cette solution rend le code moins lisible et « htlm5 firiendly » que la première solution)

Figure 11 - Arborescence de fichiers détaillée du dossier js

Blob.js et canvas-toBlob.js: Gestion de blobs coté client. Exploité dans les pages «References» et «Phylogeny» et «Fastosh» pour la génération d'archives coté client.

buttons.xxx.js : Fichiers permettant l'ntégration de boutons ajoutant des fonctionnalités supplémentaires à DataTatables. (fonctionnalité xxx peut être génération de pdf, excel, bouton de téléchargement)

dataTables.xxx.min.js : Fichiers JS permettant ntégration de DataTables au design de SemanticUI.

drilldown.js: Support de la fonctionnalité drilldown (multi-level chart) par Highcharts.

driver.min.js: utilisé pour mettre en surbrillance des éléments de la page web pour les tutoriels intéractifs.

Filesaver.min.js: utilisé pour la génération de téléchargements côté client (exploite les fonctionnalités de blob.js et canvas-toBlob.js, et jszip.js).

Filesaver.js: non exploité (code non minifié de Filesaver.min.js)

gamer.common.js : Code Javascript exécuté dans plusieurs vues (pages html) de l'application web. Evite d'avoir à répéter le même code pour plusieurs pages. Utilisé pour détecter un navigateur web non supporté, initialiser les fonctions et librairies utilisées dans toutes les vues, gérer le formulaire de contact...

gamer.datatables.nom_espece.js : Gestion des fonctionnalités des pages «genomes» pour l'espèce en question.

gamer.datatables.refs.nom_espece.js : Gestion des fonctionnalités des pages «reference» pour l'espèce en question.

gamer.datatables.salmonellatuto.js : Gestion des fonctionnalités de la page de tutoriel dynamique.

gamer.fastosh.js : Gestion de la page d'accueil de l'outil «Fastosh».

gamer.fastoshResults.js : Gestion des pages de résultats d'exécution de Fastosh.

gamer.fastoshGlobal.js : Gestion pages «Phylogeny».

gamer.home.js : Gestion de la page d'accueil de l'appliation web.

highcharts.js : Support de Highcharts (graphiques dynamiques).

Jquery xxxxx .js : Intégration de fonctionnalités de JQuery

jsphylosvg-min.js et **raphael.min.js** : Gestion de la visualisation de fichiers newick.

Jszip.min.js : Utilisé pour la génération d'archives côté client.

Jszip.js : non exploité (version non minifiée de jszip.min.js)

pdfmake.js et **vfs_fonts.js** : génération de pdf côté client.

underscore.js : bibliothèque d'algorithmes optimisés pour le parsing de fichiers json.

5.2.4. videos

Vidéos descriptives des espaces de travail implémentés dans l'application web.

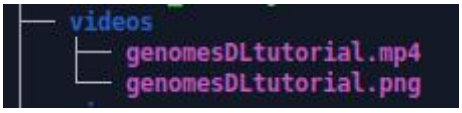


Figure 12 - Arborescence de fichiers détaillée du dossier videos

5.2.5. views

Pages 403,404, et 500.html : pages d'erreur

Dossier homepage :

Index.html :Page d'accueil

Dossier includes :

Ce dossier contient un dossier « includes » qui contient des morceaux de codes html répétitifs, ces bouts de code seront fusionnés à la volée aux templates générés (dans les autres dossiers) nécessitant l'utilisation de ce code : ceci est possible grâce à une fonction include similaire à celle de php.

footer.html : bas de page

genomes_help.html : popup visible en cliquant sur «help» dans les espaces de travail de l'application web.

header.html : en-tête

news_latest.html : actualité la plus récente

news_old.html : actualités

Dossier moreinfo

about.html : à propos de l'application web et de l'équipe

faq.html : foire aux questions

How_to_ivarcall2.html : non exploité

How_to_raxml.html : non exploité

news.html : non exploité

Dossier multiqc:

Contient les multiqc globaux générés automatiquement pour chaque espèce.

Dossiers species (exemple pour Clostridium) :

Contient les vues des espaces de travail spécifiques à chaque espèce.

Dossier moreinfo :

About : description de l'objectif du site web, etc […]

Faq : foire aux questions

How_to_xx : intégration des pdf de formation à xx

News : page de news

Dossier tools:

Contient les vues de chaque outil intégré à l'application web (à l'heure actuelle, uniquement Fastosh)

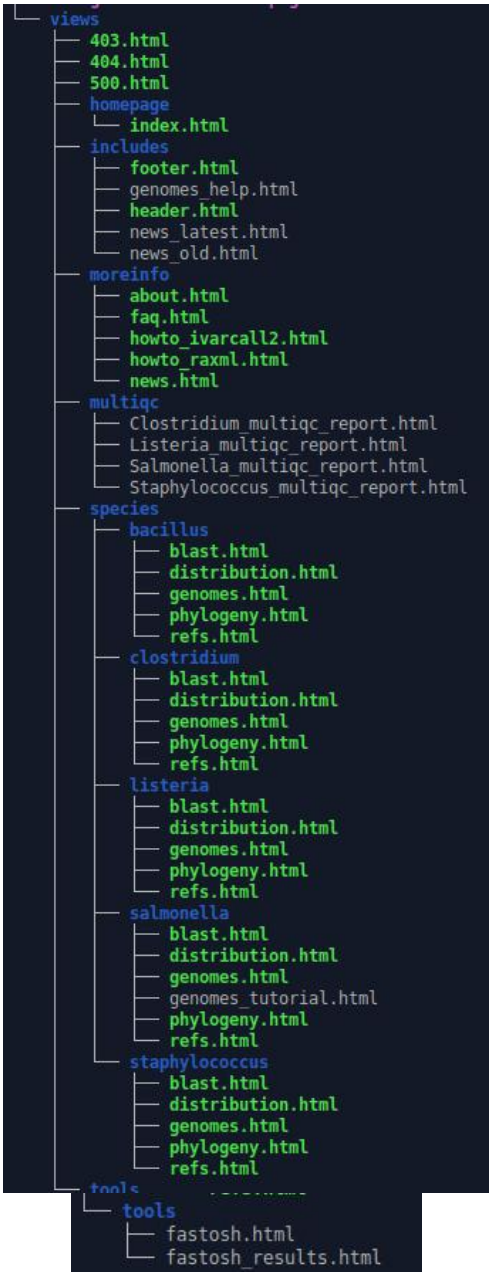


Figure 13 - Arborescence de fichiers détaillée du dossier views

5.3. Dossier «node_modules»

Fichiers utiles au fonctionnement de NodeJS : modules NodeJS par défaut et supplémentaires. (Bientôt ExpressJS, etc.) . La liste complète est disponible dans le fichier **package.json** mis à jour automatiquement par npm à chaque ajout de module si celui-ci n'est pas installé en mode global (npm install -g)

pm2: gestionnaire d'applications NodeJS utilisée pour :

-Faire tourner ses applications en tant que service (daemon).

-Redémarrer automatiquement l'application en cas de crash (ce n'est pas de cas d'une application NodeJs native.)

-Faire du monitoring des ressources consommées par l'application.

-En développement, utiliser le mode « watch » pour une relance de l'application des qu'une modification est détectée dans les fichiers de l'application.

validator

Sécurisation des formulaires : Protection contre le XSS par validation/nettoyage des requêtes passant par celui-ci.

TemplatesJS

Utilisé pour le templating (php-like) : pour les includes (header, footer) et rendu des données extraites de GAMeRdb.

Express (Expressjs)

Pour la gestion des routes dynamiques. N'a finalement jamais été utilisé car nécessitait de trop lourdes modifications de l'organisation du site (arborescence et nom des fichiers, deplacement de SemanticUI, modification de la majorité du code du Modèle et du Controleur) afin de pouvoir être exploité.

Http-server

Gestion « simplifiée » du serveur http.

Commander

Parseur d'arguments

Autres

Dépendances téléchargées automatiquement dans le cadre de l'installation des modules cités ci-dessus.

5.4. Dossier «scripts»

Contient tous les fichiers back-end de l'application web (scripts, fichier serveur, raccourcis vers le NAS)

Controller.js : Controleur (à lancer via pm2)

Fastosh_web.py : Génération de phylogénies en fonction des paramètres envoyés par le client.

Fastosh.js : non exploité

Fastosh_interface.py: non exploité

Model.js : Modèle

Multiqc : lien symbolique vers le dossier multiqc. Utile pour faciliter la gestion des routes par Controller.js

old : sauvegarde de scripts dépréciés mais dont le contenu pourrait être ré-utilisable à l'avenir. (contient nottament des scripts pour la génération de phylogénies avec métadonnées)

private: retour de bugs utilisateurs et autres fichiers sensibles non uploadés sur github

testModel.js : non exploité

tmp : lien symbolique vers le dossier tmp du NAS (pour la génération des résultats à renvoyer au client). Utile pour faciliter la gestion des routes par Controller.js

videos : lien symbolique vers le dossier videos. Utile pour faciliter la gestion des routes par Controller.js

ZipAndCall.sh : script de compression de fichiers coté back-end en fonction des paramètres envoyés par le client. (par exemple: archivage de fichiers fasta et genbank pour 5 souches, puisque demandé par le client)

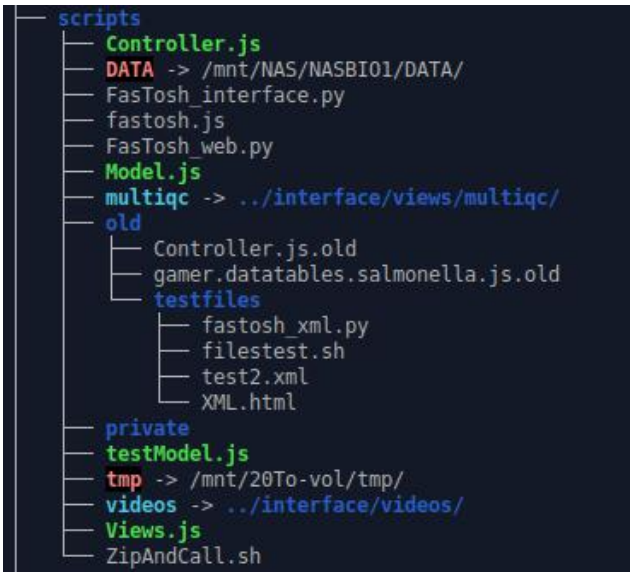


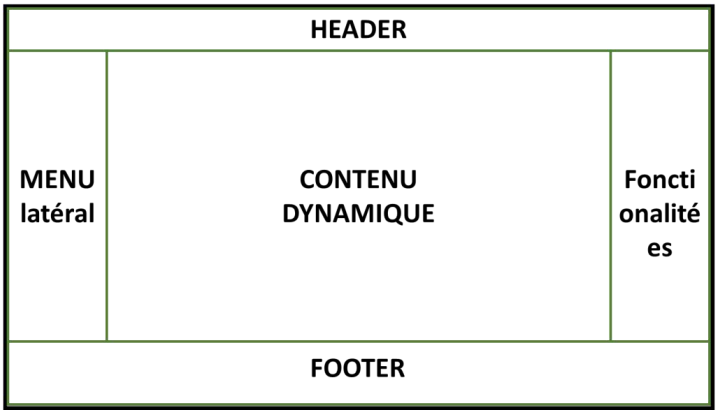
Figure 14 - Arborescence de fichiers détaillée du dossier scripts

5.5. Dossier «semantic»

Figure 9 - Arborescence de fichiers détaillée du dossier semantic

Son fonctionnement à déjà été détaillé en partie 3.

6. Description du template générique (valable pour toutes les vues)



Le header permettra d'accéder aux Espaces de travail dédiés à chaque espèce pathogène Bacillus, Clostridium, Listeria, Salmonella, Staphylococcus), celui-ci est fixé en haut de la page et s'affiche constamment.

Les 2 menus latéraux n'apparaissent pas dans la page d'accueil et sont indépendamment optionnels. Le menu latéral de gauche est optionnel et permet de naviguer entre les diverses vues (Génomes, Références, CC/ST, Phylogénie…) d'un Espace de travail d'une espèce pathogène. Le menu latéral de

droite est optionnel et affiche les options disponibles lors de l'utilisation d'un outil pour certains outils à définir (options de lancement, télécharger au format, etc…).

Le footer est fixé en bas de la page et constitue une interface vers divers liens utiles et pour contacter les développeurs du site web.

Autre informations :

- Les templates HTML ont été crées en respectant autant que possible les normes HTML5 et les bonnes pratiques suggérées par le W3C.

- Les urls pointant vers un site externe s'ouvrent automatiquement dans un nouvel onglet (target="_blank", et rel="noopener noreferrer")

7. Sécurité et mise en production

7.1. Ouverture des liens

Les urls pointant vers un site externe s'ouvrent automatiquement dans un nouvel onglet. rel="noopener noreferrer" est utilisé afin d'éviter les relatives à **`window.opener`**

7.2. Sécurisation des formulaires

Les formulaires sont protégés contre le « Cross-Site Scripting » (XSS) à l'aide d'un analyseur/sanitiseur de requêtes (npm-validator) qui analysera toutes les données en entrée (front-end) et en sortie (back-end) des formulaires. Ceci constitue une première barrière pour se prémunir de la compromission des données contenues dans GAMeRdb ou accès aux données sensibles de l'interface.

7.3. Sécurisation de la base de données GAMeRdb

La configuration par défaut de MongoDB suffit à priori pour la plupart des utilisations en phase de développement. Cette configuration est disponible dans le fichier /etc/mongodb.conf.

Toutefois, la configuration par défaut de MongoDB est très permissive. Elle permet toutes les actions possibles sur la base de données sans besoin d'authentification, contrairement à la plupart des autres SGBD MySQL qui exigent une authentification par défaut. Afin d'éviter les risques de compromission des données il est donc nécessaire de modifier cette configuration pour toute mise en production. C'est ce qui à été fait avec la nouvelle configuration qui apporte les améliorations suivantes :

- Filtrage par ip: accès à la db uniquement par le serveur (en localhost)

- Accès a la db par authentification

- Création d'un administrateur global(toutes les bases de données)*

- Création d'un administrateur local (à une base de donnée, celle ou il à été crée)*

- Création d'un utilisateur standard*

* détail :

db.createUser({user: "localAdmin", pwd: "qwerty", roles: ["userAdmin"]})
db.createUser({user: "Kindle", pwd: "Amazon", roles: ["read"]})
db.createUser({user: "Manager", pwd: "reganaM", roles: ["readWrite"]})

7.4. Utilisation d'un reverse proxy (non effectué, conseillé.)

Utilisation de Nginx qui est le plus léger et le plus performant (par rapport à Apache) pour servir des fichiers statiques.

Avantages de l'utilisation d'un reverse proxy (Nginx, Apache..):

Pouvoir utiliser le port 80 en production sans avoir à ce soucier de donner les privlièges ROOT à l'application NodeJS. Le port 80 est le port standard pour une connection http (443 en https), par conséquent en cas d'utilisation de ce port, l'url d'accès au site s'en retrouvera raccourcie car il ne sera plus nécessaire d'y préciser le n° de port.

Ouvre la possibilité de répartir la charge du site entre plusieurs serveurs.

Prévient des attaques DDOS (non pertinent en localhost)

Compression des reponses http

Possibilite de filtrer l'accès au site, mettre en place rapidement un accès par authentification.