

DOSSIER DE PROJET

Refonte complète du site de la Mission Locale Vendée Atlantique

Brian EPAUD
Promotion 2018-2019
Développeur Web et Web Mobile
ARINFO – La Roche sur Yon

AVANT-PROPOS

Ce dossier professionnel a été réalisé dans le cadre de ma période de professionnalisation qui a démarré le 17 novembre 2018. En effet, j'ai eu l'opportunité à l'automne dernier de pouvoir me former au métier de **Développeur Web** tout en restant salarié de ma structure (la Mission Locale Vendée Atlantique).

A la fin de mes études – une licence de psychologie obtenue en 2007 et une licence des sciences de l'éducation en 2009 – j'ai exercé durant 3 années comme *Conseiller en Insertion Professionnelle*, puis 6 années comme *Chargé des Relations Entreprises* en Mission Locale (accompagnement des jeunes de 16 à 25 ans vers l'emploi et la formation).

J'ai découvert la programmation et la création de sites internet en janvier 2018 lors d'une intervention de l'association **ADN OUEST** qui présentait aux différents conseillers et partenaires les différents cursus et les besoins dans le secteur du numérique. Plusieurs plateformes d'E-learning ont été évoquées, et par curiosité j'ai voulu me rendre compte du fonctionnement du code (qui me paraissait inaccessible à l'époque).

J'ai très vite été « contaminé » par toutes les possibilités offertes par la programmation et j'ai notamment été surpris par le nombre de ressources disponibles gratuitement pour appréhender les bases des différents langages.

Par chance, lors de mon entretien annuel, lorsque j'ai pu évoquer l'idée à ma Direction, celle-ci a pu me proposer de suivre cette formation sous forme de période de professionnalisation (alternance) afin de réaliser la refonte du site de la **Mission Locale Vendée Atlantique** dans l'optique de répondre aux attentes de nos financeurs :

- Augmenter le nombre de premiers accueils et notamment des jeunes dits « Invisibles »,
- Entretenir la relation entreprise en proposant d'augmenter la visibilité des offres d'emploi,
- Augmenter l'accessibilité aux offres de nos partenaires et notamment des outils proposés sur l'Emploi Store de **Pôle Emploi**.

Cette démarche s'inscrit aussi dans l'idée de moderniser l'image de la Mission Locale Vendée Atlantique afin que l'offre de service soit plus accessible aux jeunes via les différents formats numériques.

La formation n'a personnellement pas été simple à suivre entre la formation (et l'ensemble des connaissances à emmagasiner) et d'absorber le reste de mon poste

sur deux ou trois jours. Mais le projet final me tenait à cœur et m'a permis de maintenir ma motivation durant les 10 mois.

Je tiens à remercier *Philippe YIM*, mon formateur, pour la qualité de sa formation, sa disponibilité et sa pédagogie. Je remercie également *Joachim BRASIER*, développeur et apprenti designer durant ma formation à **ARINFO**, qui m'a conseillé durant les derniers mois et a pris le temps de me guider à travers les différentes notions à aborder. Enfin je remercie ma Directrice *Véronique CANTIN* pour m'avoir permis de faire cette formation, *Stéphanie* pour son œil averti, sa patience, et ses nombreuses relectures et ma femme *Sandra*, qui a su me soutenir et m'encourager dans les moments de doutes et de remises en question.

TABLE DES MATIERES

Avant-propos	3
Introduction / Résumé du projet	7
Compétences du référentiel couvertes par le projet	8
Maquetter une application	8
Réaliser une interface utilisateur web statique et adaptable.....	8
Développer une interface utilisateur web dynamique	8
Créer une base de données et développer les composants d'accès à celle-ci	9
Développer la partie back-end d'une application web ou web mobile	9
Présentation de la cliente et des documents techniques	10
Construction du cahier des charges	10
Veille concurrentielle	10
Définition de l'arborescence	11
Droits utilisateurs.....	13
Charte graphique.....	14
Les logos	14
Les couleurs	15
Les Polices.....	15
La réglementation générale de protection des données (RGPD)	16
Méthodologie de travail	16
Développement et intégration Front-end	18
Unified Modeling Language (UML)	18
Diagramme de cas d'utilisation.....	18
Diagramme de classes.....	19
Diagramme de séquences	20
Maquettage.....	21
Wireframes	21
Le Design.....	23
Maquette HTML 5 / CSS 3 / JAVASCRIPT	25
Développement Back-end	28
Technologies et modules NPM utilisés.....	28

Templating	29
Construction de la base de données et modalités d'accès	31
Résolution d'un problème avec une documentation anglophone	34
Mise en place des routes et des requêtes.....	37
Résultat Front-end	37
Organisation du Back-end et des différentes requêtes	39
Authentification et gestion des droits utilisateur	47
Sécurité	50
Autres fonctionnalités	57
Tests Unitaires	60
Mise en production	62
Hébergement	62
Optimisation des performances et Référencement	64
Référencement (SEO)	64
Bonnes pratiques.....	65
L'accessibilité	65
Performances.....	66
Cookies / RGPD	66
Analyse des données	68
Google Analytics.....	68
Google Search Console	69
Rendu final	70
Documentation de l'API	72
Conclusion et perspectives	74

INTRODUCTION / RESUME DU PROJET

Dans le cadre de ses orientations 2019-2021, les Mission Locales axent de plus en plus l'accompagnement des jeunes de 16 à 25 ans vers des outils numériques, que cela soit pour la communication (réseaux sociaux et accompagnement à distance avec leur conseiller référent) ou que ce soit pour le positionnement sur les offres d'emploi. A ce titre, la **Mission Locale Vendée Atlantique** a souhaité suivre la tendance en réfléchissant à la refonte de son site internet qui était devenu obsolète. Cette nouvelle application doit être en mesure d'attirer à la fois de nouveaux jeunes et de maintenir son réseau employeur. La refonte du site s'est faite de A à Z, en commençant par la définition du cahier des charges qui recense les besoins de la cliente en exposant les contraintes techniques, puis par la réalisation de la maquette graphique (des wireframes à la maquette HTML/CSS) et enfin tout le développement du site avec le déploiement des composants d'accès à la base de données et l'ajout au fur et à mesure des différentes fonctionnalités qui permettent aujourd'hui à la cliente d'être quasiment 100% autonome dans la gestion de son site : ajout, modification et suppression d'actualités, d'offres d'emploi, d'outils numériques et la gestion des utilisateurs par l'administrateur.

De leur côté, une fois inscrits, les jeunes peuvent importer jusqu'à 5 CV afin de postuler en ligne sur les offres d'emploi qu'ils peuvent d'ailleurs mettre en favoris dans leur espace personnel. Les employeurs, eux, peuvent éditer des offres d'emploi et voir la liste de toutes leurs offres dans leur espace.

Ce projet fut long et complexe, car il m'a amené à jongler entre les différents rôles qui gravitent autour d'un projet (chef de projet, designer, développeur, etc.) et de m'adapter en fonction de mes interlocuteurs.

Finalement, le projet et son déroulement furent très appréciés par la cliente, au point que plusieurs Missions Locales du département aimeraient se l'approprier.

CONTEXTE TECHNIQUE ET METHODOLOGIE DE PROJET

COMPETENCES DU REFERENTIEL COUVERTES PAR LE PROJET

Le projet de refonte du site de la Mission Locale Vendée Atlantique m'a permis de couvrir 6 des 8 compétences attendues dans les deux activités-type du référentiel du Titre Professionnel « Développeur Web et Web Mobile ».

MAQUETTER UNE APPLICATION

Après avoir moi-même rédigé le cahier des charges avec la cliente, j'ai effectué de la veille concurrentielle afin d'observer les sites des autres Missions Locales. Parallèlement j'ai consulté plusieurs sites afin de m'inspirer des tendances actuelles en termes de fonctionnalités et de design.

J'ai mis en place une modélisation UML pour mettre en forme les différentes fonctionnalités.

Puis j'ai commencé à schématiser et dessiner à mains levées la structure du site. J'ai ensuite transposé les croquis en format numérique par l'intermédiaire du logiciel **Balsamiq**.

J'ai alors pu réaliser les wireframes des versions desktop (Ordinateur) et mobile.

REALISER UNE INTERFACE UTILISATEUR WEB STATIQUE ET ADAPTABLE

Une fois les wireframes terminés, en collaboration avec un étudiant Designer Web, j'ai créé un design à l'aide du logiciel **Lunacy** afin de le valider auprès de la cliente. J'ai ensuite pu développer la maquette en utilisant les langages **HTML/CSS** conjugué à l'utilisation du framework **Bootstrap** pour faciliter l'adaptabilité de l'interface en fonction de la taille de l'écran utilisé.

Je suis resté vigilant à la fois aux recommandations du W3C pour les bonnes pratiques de la rédaction du code, et à la fois aux règles d'accessibilité du WCAG.

DEVELOPPER UNE INTERFACE UTILISATEUR WEB DYNAMIQUE

Pour améliorer l'interactivité du site Web j'ai ajouté plusieurs scripts. J'ai utilisé aussi des API afin de pouvoir récolter des informations de sources extérieures (cartographie par exemple).

De même, j'ai utilisé plusieurs scripts tout au long du développement de l'application pour rendre l'expérience utilisateur plus cohérente et plus fluide.

CREER UNE BASE DE DONNEES ET DEVELOPPER LES COMPOSANTS D'ACCES A CELLE-CI

Une fois la maquette statique présentée et validée par la cliente, j'ai pu initialiser les différents modèles de la base de données qui correspondaient aux diagrammes de classes du modèle UML.

J'ai choisi d'utiliser **MongoDB** qui est un système de gestion de base de données orienté document. A l'aide du module **Mongoose** j'ai pu lier le SGBD entre mon back-end et mon front-end en réalisant plusieurs schémas spécifiques (actualités, offres d'emploi, etc.).

Ainsi j'ai pu réussir à manipuler facilement les données collectées.

DEVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB OU WEB MOBILE

J'ai choisi d'utiliser **NodeJS** pour le développement de la partie back-end.

Lorsque mes différents modèles furent créés pour ma base de données, j'ai entrepris la construction des différentes routes d'accès aux différentes pages que j'avais préalablement découpées avec un moteur de templating (**Handlebars**).

J'ai démarré par le développement de l'administration afin de pouvoir ajouter du contenu facilement.

J'ai veillé aux points de vigilance concernant l'authentification et la sécurité des utilisateurs.

J'ai ensuite développé les différentes routes et fonctionnalités de toutes les pages pour réussir à mettre le site internet en production.

PRESENTATION DE LA CLIENTE ET DES DOCUMENTS TECHNIQUES

Pour commencer la refonte du site internet, j'ai recueilli les besoins de la cliente représentée par la Directrice de la Mission Locale Vendée Atlantique. C'est une association loi 1901, qui s'occupe de l'accompagnement des jeunes de 16 à 25 ans vers l'emploi et la formation qualifiante.

La structure possédait un site internet datant d'une quinzaine d'années, et qui ne comportait aucune solution de gestion de contenu :



Les demandes principales de la cliente étaient donc les suivantes :

- Moderniser l'image de la Mission Locale Vendée Atlantique pour l'adapter au public jeune,
- Permettre aux jeunes d'utiliser le site pour postuler aux offres d'emploi,
- Permettre aux entreprises de déposer des offres d'emploi afin d'augmenter la visibilité de leurs besoins en recrutement,
- Permettre à la Mission Locale d'entretenir les relations partenariales en diffusant des actualités, et des outils d'aide à la recherche d'emploi,
- Et enfin que la Mission Locale soit autonome dans sa gestion du contenu du site.

CONSTRUCTION DU CAHIER DES CHARGES

VEILLE CONCURRENTIELLE

Après avoir pris note des différents besoins de la cliente, j'ai démarré la rédaction du cahier des charges en établissant un état des lieux du plan du site

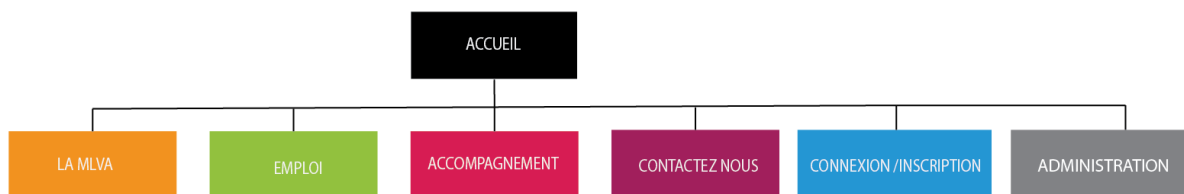
existant de la Mission Locale Vendée Atlantique (cf. Annexe 1) tout en effectuant une veille sur les sites des autres Missions Locales de France.

Je suis notamment allé consulter les sites suivants :

- Mission Locale de Dijon : <https://www.mldijon.asso.fr/>
- Mission Locale Corse : <http://www.missions-locales-corse.org/>
- Mission Locale TechnoWest : <https://www.mltechnowest.com/>

DEFINITION DE L'ARBORESCENCE

J'ai ensuite défini l'arborescence des différentes parties de l'application web. Cette arborescence est détaillée dans le cahier des charges (cf. Annexe n°2 :Cahier des Charges – chapitre 5.1), mais voici l'arborescence générale :



Le site comporte plusieurs sections, elles-mêmes composées de sous-rubriques :

- **Section MLVA :**

- Actualités
- Présentation de la structure
- L'offre de service
- L'équipe de la MLVA

Cette section permet de recenser les informations relatives à l'association, que ce soit pour les actualités ou que ce soit pour la présentation de la Mission Locale, son offre de service et son équipe.

- **Section Emploi :**

- Offres d'emploi
- Outils d'aide à la recherche d'emploi

Cette section fait référence à une fonctionnalité principale du site. En effet, la liste des offres d'emploi et des outils d'aide à la recherche d'emploi sera agrémentée par les employeurs et/ou les modérateurs.

- **Section Accompagnement :**

- Santé
- Logement
- Handicap
- Garantie Jeunes
- Mobilité
- Accès aux droits

Cette section permet aux jeunes de s'informer sur les dispositifs et sur les conseils thématiques donnés par les conseillers de la MLVA.

- **Section Contact :**

- Nous trouver
- Nous écrire

Toutes les coordonnées de chaque antenne et permanence sont visibles à partir d'une carte (avec un accès direct au GPS pour les smartphones). De plus, chaque visiteur pourra écrire à la MLVA pour des renseignements ou prendre un RDV.

- **Espace Connexion :**

Page qui permet, lorsque l'utilisateur est inscrit, de s'identifier à partir de son adresse mail et d'un mot de passe. S'il ne l'est pas, il a la possibilité de s'inscrire. S'il a oublié son mot de passe, il peut le réinitialiser.

- **Espace Inscription :**

- Pour les jeunes
- Pour les professionnels

Page d'inscription divisée en deux parties : un formulaire d'inscription pour les jeunes et un formulaire d'inscription pour les employeurs.

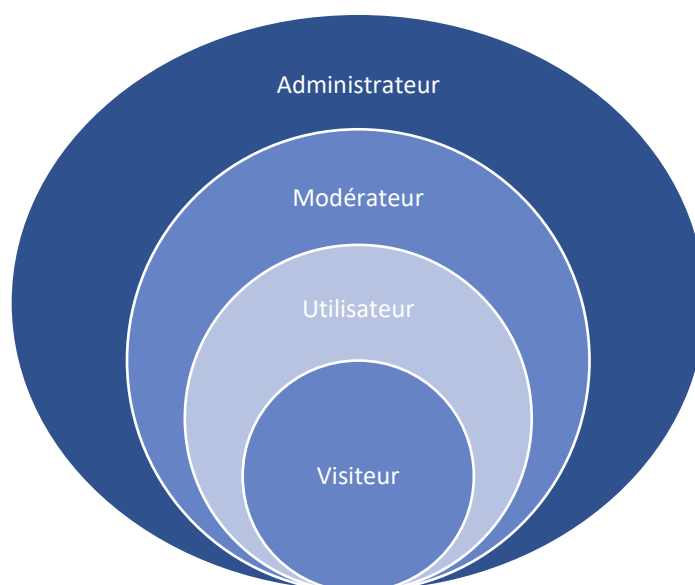
- **Espace Administration :**

Pour les administrateurs, cette section permet :

- La gestion des actualités (ajout, édition, suppression)
- La gestion des offres d'emploi (ajout, édition, suppression)
- La gestion des outils (ajout, édition, suppression)
- La gestion des droits utilisateurs (édition du rôle, suppression)

DROITS UTILISATEURS

Le cahier des charges a aussi permis de définir les 4 différents rôles des utilisateurs du site dont les droits différeront :



Les **administrateurs** sont les utilisateurs avec le plus haut niveau de droits sur le site. Ils correspondent aux membres de la structure capables de maintenir le site stable, d'en modifier les rubriques et d'administrer des utilisateurs.

Les **modérateurs** sont les utilisateurs qui pourront à partir de l'administration, ajouter du contenu dans le site comme des offres d'emploi, des actualités, ou des outils de recherche d'emploi. Ils ne pourront en revanche pas gérer les droits utilisateurs.

Les **utilisateurs** représentent les visiteurs qui se créent un compte (deux types : jeunes et employeurs) afin de profiter de leur espace personnel. Pour les uns cela permet de postuler aux offres d'emploi présentes sur le site, de suivre ses différentes candidatures, de créer son profil professionnel et de gérer son compte. Pour les autres, ils pourront déposer des offres d'emploi sur le site, suivre la liste des offres déjà publiées sur le site, de consulter la CV thèque, et de gérer son compte.

Les **visiteurs** n'ont aucun droit spécifique, ils peuvent naviguer sur le site et consulter les informations, actualités et/ou les offres d'emploi mais ne pourront pas postuler. Chaque visiteur peut devenir utilisateur à partir du moment où il s'inscrit sur le site.

CHARTRE GRAPHIQUE

L'Union Nationale des Missions Locales (UNML) possède une charte graphique qui a été actualisée en juin 2019 (cf. Annexe n°3 – Charte graphique UNML).

LES LOGOS

Celle-ci m'a d'abord permis de m'appuyer sur un logo déclinable en deux versions :



version
verticale



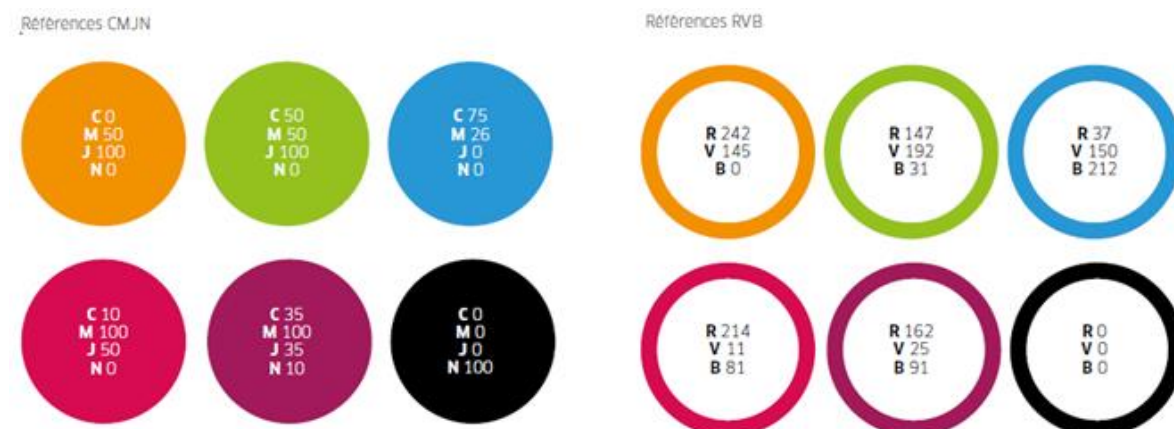
version
horizontale

J'ai décidé d'opter pour la version verticale. Parallèlement, j'ai utilisé le logo existant de la MLVA :



LES COULEURS

La charte graphique de l'UNML, met à disposition 6 couleurs :



La charte graphique prévoyait soit les codes RGB ou soit les codes CMJN. J'ai privilégié les codes hexadécimaux qui présentent l'avantage de se charger un peu plus rapidement sur les navigateurs. Pour opérer la conversion j'ai utilisé le site htmlcolorcodes.com :

```
$blue: #2897D5; $red: #D60B52; $orange: #F39200; $green: #95C11F; $purple: #A3195B;
```

LES POLICES

Pour la police d'écriture, la charte proposait :

Pour la papeterie et les supports de communication

TÉLÉCHARGEABLE SUR BEFONTS.COM

Akrobat

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789

Texte courant. Corps 9 pt. Borpossedi cum aces esedi sus apitiis ditis apis eaquid qui conimus num fugiae volesti asperoriae. Lit sunti iuntotatur aute aliquid ellabor eprattem none sit offictur occulla boreiur asinveresti dest utescit estibus, sume exceptat. Ugia sandebit as suntet ipsaepero tem laboreptaera que volum que rae repraee nonectia suntiss magnis quis rehentur qui berum facerum quamet hiliqua sperume sit.

Arial

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789

Texte courant. Corps 8,5 pt. Borpossedi cum aces esedi sus apitiis ditis apis eaquid qui conimus num fugiae volesti asperoriae. Lit sunti iuntotatur aute aliquid ellabor eprattem none sit offictur occulla boreiur asinveresti dest utescit estibus, sume exceptat. Ugia sandebit as suntet ipsaepero tem laboreptaera que volum que rae repraee nonectia suntiss magnis quis rehentur qui berum facerum quamet id unt ilis eum si rat hiliqua sperume sit.

J'ai sélectionné en supplément la police ALEGREYA (téléchargée gratuitement sur Google Font) pour les headers afin de faire une distinction avec le reste du site.

LA REGLEMENTATION GENERALE DE PROTECTION DES DONNEES (RGPD)

Cette réglementation est primordiale dans le projet de refonte du site. En effet, deux éléments sont à prendre en compte :

- Les données des utilisateurs qui seront recueillies lors de l'inscription/connexion
- Les coordonnées des salariés de la Mission Locale qui doivent aussi donner leur accord.

Pour respecter le premier élément, j'ai tout d'abord complété la charte de confidentialité afin que chaque utilisateur puisse avoir connaissance de l'utilisation de leurs données à travers le site (Cf. chapitre Cookie / RGPD).

Pour cela je suis resté vigilant à bien notifier :

- Les coordonnées, ainsi que l'éditeur du site, et de l'hébergeur.
- Le type de données récoltées lors de la navigation sur le site web : noms, prénoms, email, téléphone, adresse postale, adresse IP, etc.
- Pourquoi ces données sont collectées ?
- Combien de temps les données restent stockées ?
- Les mesures de sécurité mises en place pour assurer la protection de ces données, ainsi que la manière dont ils peuvent exercer leur droit de modification ou de suppression de ces données.

Pour ce dernier point nous verrons ce que j'ai mis en place dans un prochain chapitre.

Concernant les données des salariés de la Mission Locale, sachant que leurs identités, leurs fonctions et leurs coordonnées seront visibles sur le site web, il était important de les informer et d'obtenir leur accord. A cet effet, j'ai fait remplir à chaque salarié un document d'information et de consentement (cf. Annexe n°4).

METHODOLOGIE DE TRAVAIL

Tout au long du projet, les différentes étapes ont été conçues en étroite collaboration avec la cliente afin que celle-ci puisse suivre les différentes étapes de création de celui-ci (méthodologie **AGILE**) via notamment un lien vers mon repository **GITHUB** et un système de dossier partagé (**Microsoft SHAREPOINT**)

interne à la Mission Locale pour les documents techniques (Charte graphique, Cahier des charges, etc.).

Lorsque le sprint le nécessitait, d'autres acteurs pouvaient intervenir sur le projet (Designer WEB, salariés de la Mission Locale, etc.)

Une fois toutes les informations techniques collectées, j'ai mis en place un rétroplanning (diagramme de **GANTT** en figure 1) afin de jalonner les différentes échéances du projet :

https://prod.teamgantt.com/gantt/schedule/?ids=1601557#ids=1601557&user=&custom=&company=&hide_completed=false&date_filter=&color_filter=

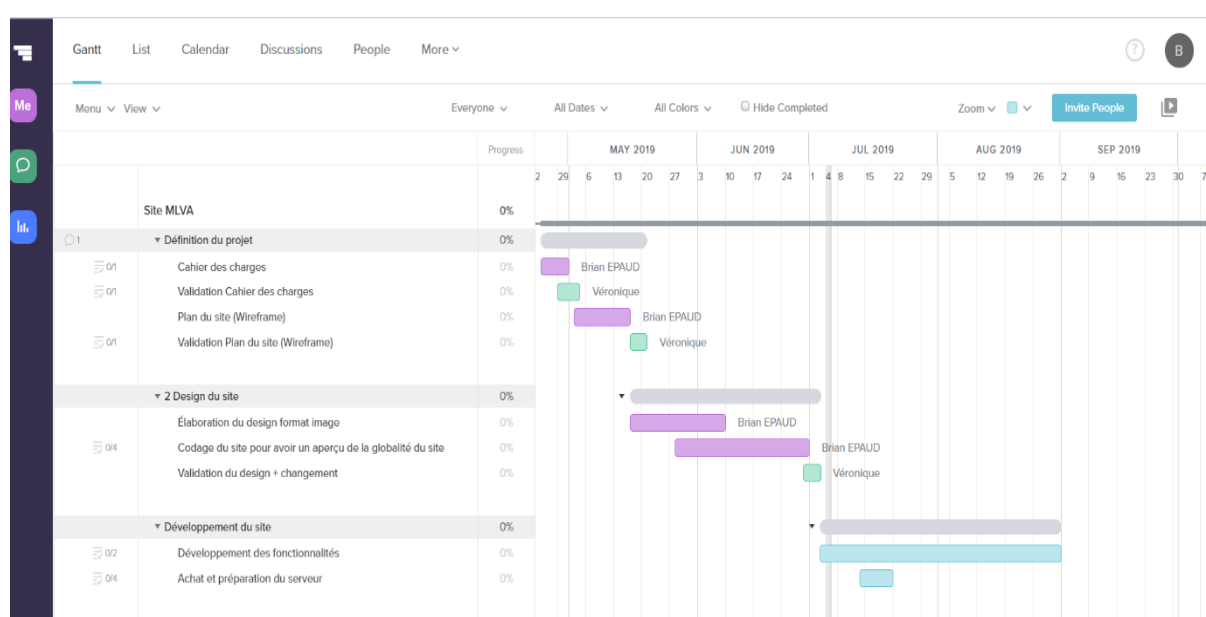


Figure 1 : Extrait du rétroplanning

REALISATION DU SITE WEB

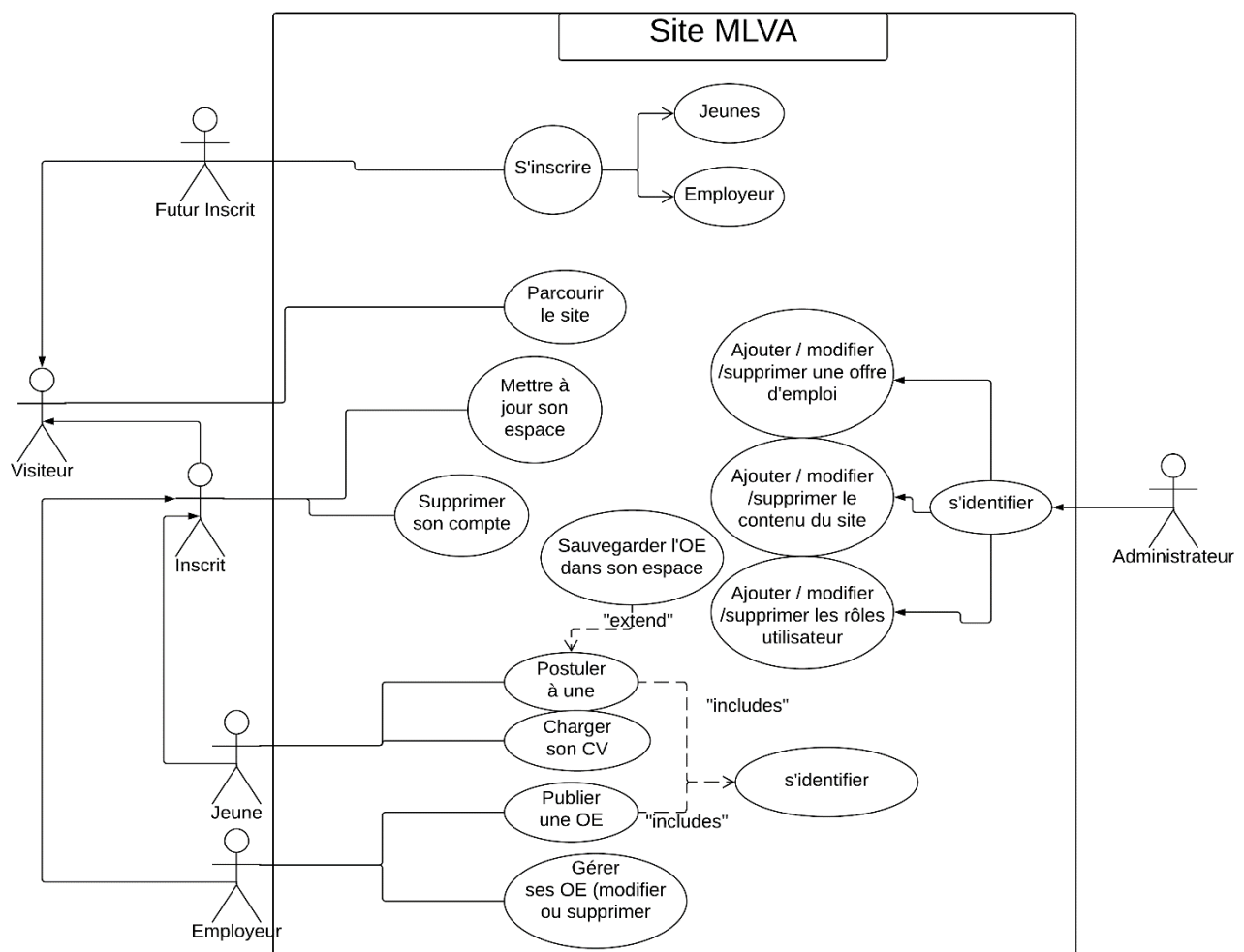
DEVELOPPEMENT ET INTEGRATION FRONT-END

UNIFIED MODELING LANGUAGE (UML)

Une fois l'étape de recueil des besoins de la cliente et de collecte des documents techniques terminée, j'ai établi mon UML pour faciliter ma réflexion sur la structure du futur site. Pour réaliser mes diagrammes (cas d'utilisation, diagrammes de classes et diagramme de séquences), j'ai utilisé le site **Lucidchart**. Ce site m'a permis de gagner beaucoup de temps dans la construction de mes différents schémas.

DIAGRAMME DE CAS D'UTILISATION

J'ai commencé par réfléchir aux cas d'utilisation (en fonction des utilisateurs) que le site allait rencontrer afin de générer un schéma me permettant de mieux visualiser les fonctionnalités attendues.



Les visiteurs non-inscrits peuvent parcourir le site.

Les visiteurs peuvent s'inscrire. Ce sont soit des jeunes, soit des professionnels. Ils peuvent ensuite gérer leur compte personnel (modification, suppression).

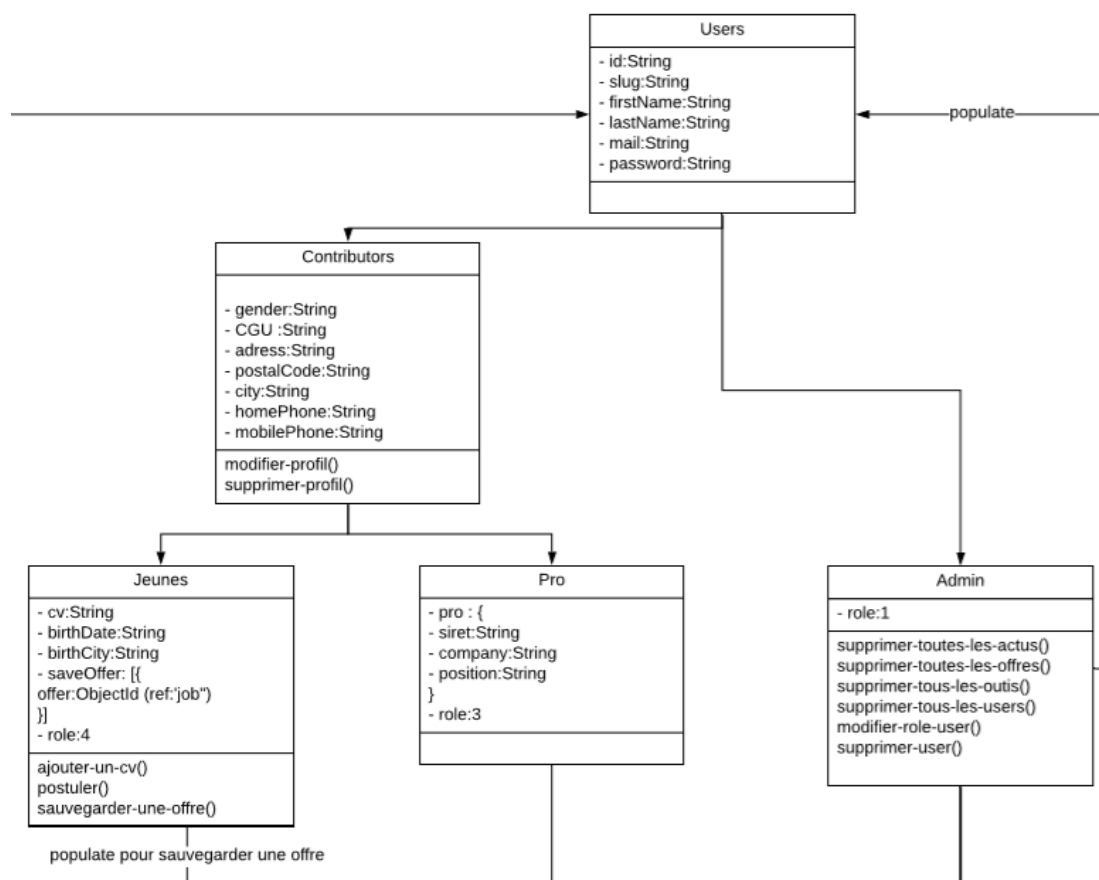
Les jeunes inscrits peuvent postuler aux offres d'emploi présentes sur le site.

Les professionnels inscrits peuvent publier une offre d'emploi, puis la modifier ou la supprimer.

Les administrateurs (et modérateurs) peuvent aussi ajouter du contenu dans le site (actualités, offres d'emploi, outils) et gérer les contenus intégrés au site (coordonnées, trombinoscope, etc.).

DIAGRAMME DE CLASSES

J'ai ensuite affiné ma réflexion et la structure du site en réalisant mes diagrammes de classes pour se rendre compte des relations entre les différents éléments. Ces diagrammes ont déterminé par la suite les documents de mes collections pour la base de données.



Exemple d'une partie du diagramme

DIAGRAMME DE SEQUENCES

Enfin j'ai réalisé quelques diagrammes de séquences pour visualiser le déroulement des fonctionnalités principales (ici connexion et ajout d'une offre d'emploi) :

DIAGRAMME DE SÉQUENCE BASIQUE POUR LA CONNEXION JEUNE

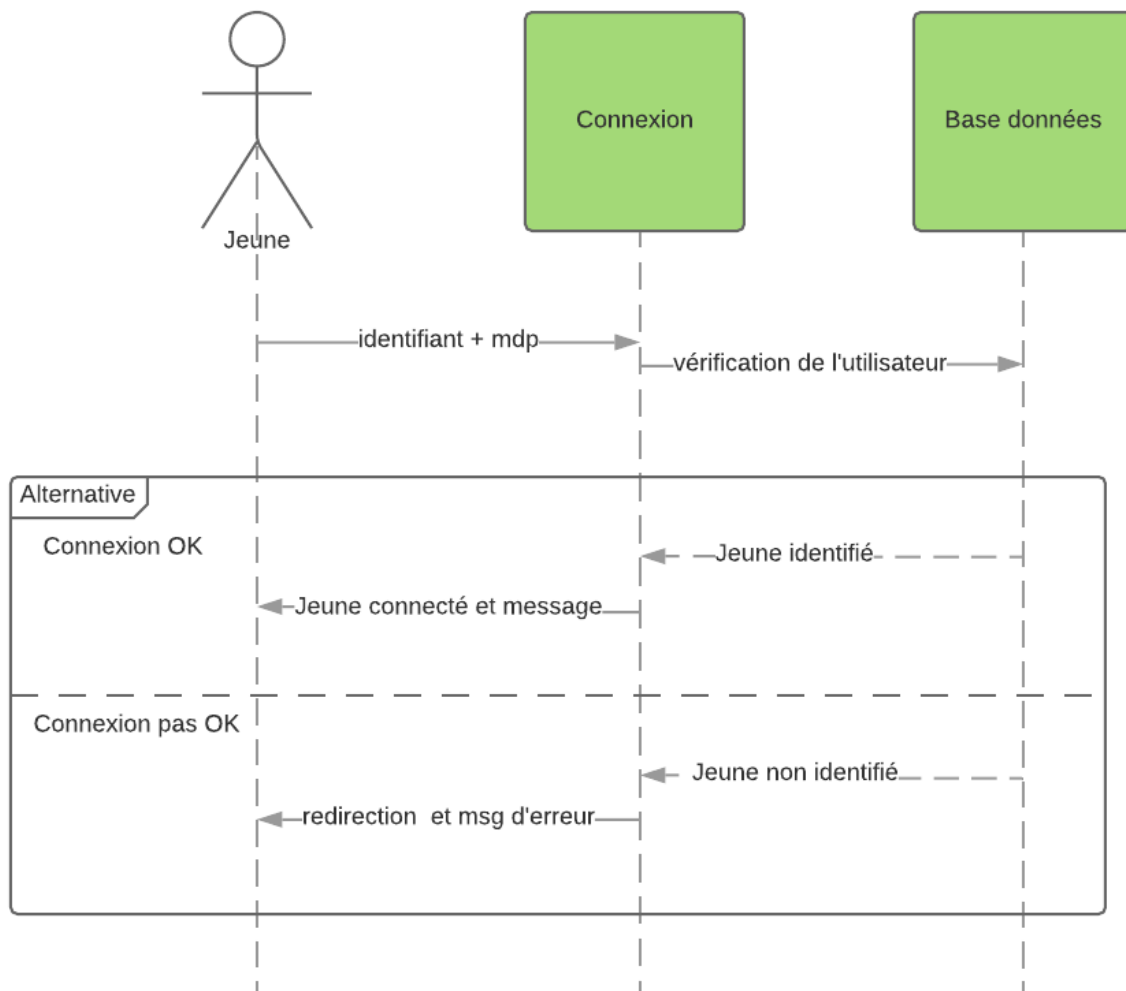
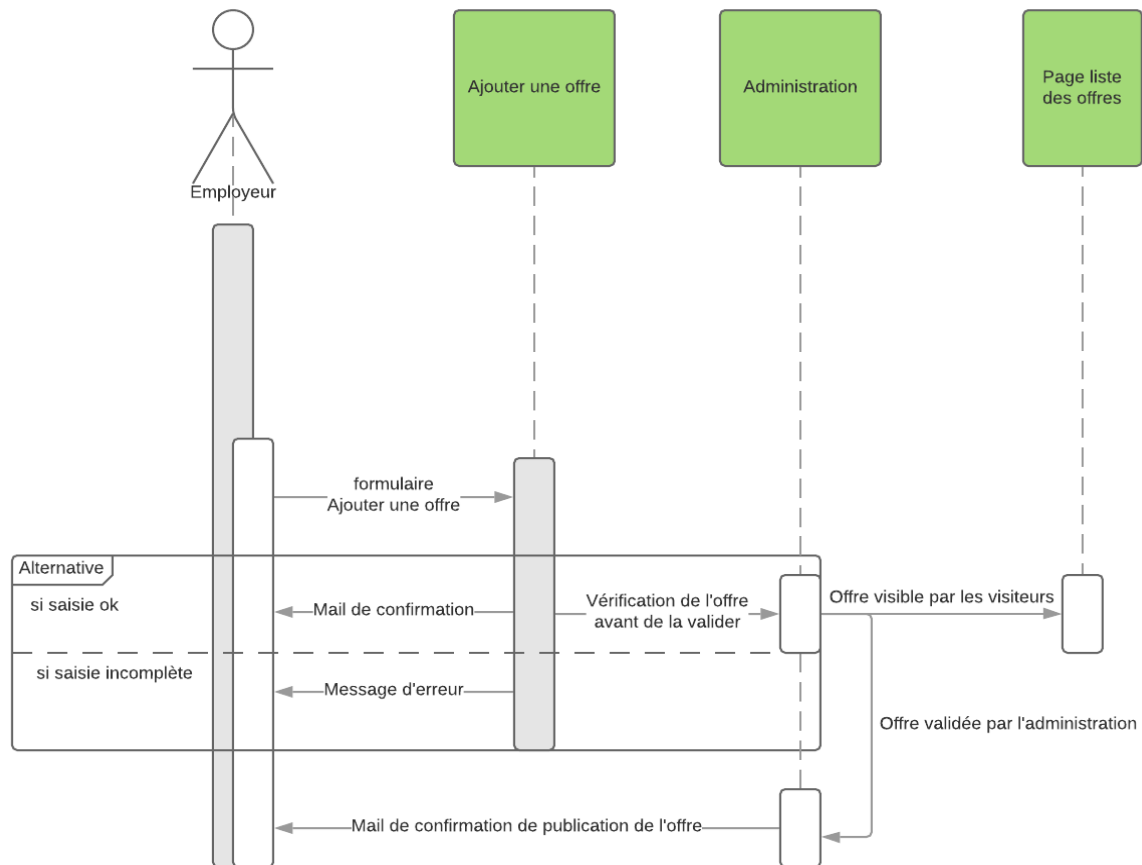


DIAGRAMME DE SÉQUENCE BASIQUE POUR L'AJOUT D'UNE OFFRE D'EMPLOI PAR UN EMPLOYEUR



Le scénario d'ajout d'une offre d'emploi est représentatif des autres scénarios (ajout actualité, outils, etc.) puisque la logique reste la même.

La seule distinction réside dans la validation de l'offre pour permettre sa publication qui n'existe que pour les offres d'emploi déposées par les employeurs.

Les différents diagrammes UML me paraissaient très théoriques lors de la conception, mais ils se sont avérés finalement très utiles dans la phase de développement.

MAQUETTAGE

WIREFRAMES

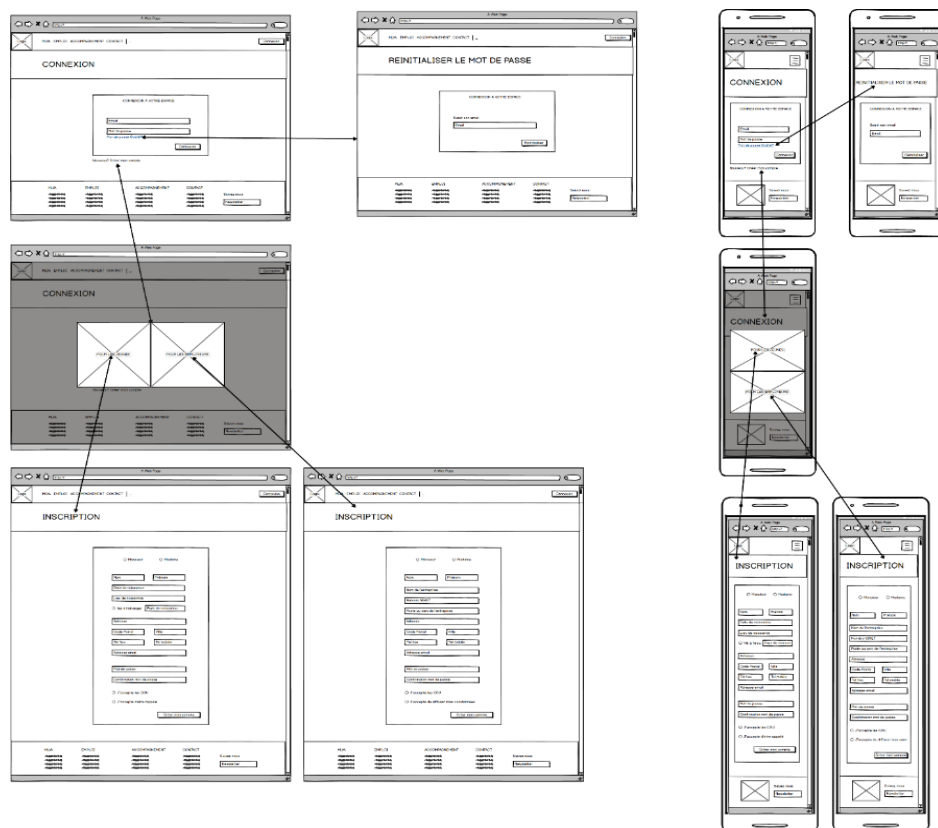
La modélisation des besoins terminée, j'ai alors pu entamer la phase de structuration de l'application.

Avant de commencer la production de la maquette, j'ai exploré les tendances de Web Design afin de m'inspirer de certains éléments. Pour se faire, j'ai notamment consulté les sites **Lapa Ninja** (<https://www.lapa.ninja/>) et **Dribbble** (<https://dribbble.com/>).

Après avoir sélectionné quelques idées, j'ai tout d'abord réalisé des croquis à mains levées afin de me donner une idée de la composition des éléments de l'application.

J'ai ensuite mis au propre mes brouillons en créant des Wireframes avec le logiciel **Balsamiq** (cf. Annexe n°6). J'ai alors pu imaginer les différents emplacements des éléments (navigation, boutons, image, zone de texte) composant le site en fonction de la taille du média utilisé, ordinateur ou smartphone, afin d'avoir un aperçu de la structure responsive de l'application web.

Voici un exemple de wireframe (page de connexion et d'inscription) en version desktop et en version mobile :



Arrivé à la fin de cette étape, j'ai présenté à la cliente cette structure ainsi que les fonctionnalités que j'avais pu ressortir dans mon étude préalable.

Elle a rapidement validé à la fois la structure globale du site et à la fois les diverses fonctionnalités qui correspondaient à ses attentes.

Afin de mieux se rendre compte du résultat final, et ainsi le présenter aux membres du bureau de l'association, la cliente m'a demandé de réaliser le design de quelques pages du site.

LE DESIGN

Comme lors de l'étape précédente, afin de m'inspirer des tendances actuelles, j'ai choisi plusieurs exemples de design afin de sélectionner les parties qui m'intéressaient.

Je me suis notamment inspiré du design de cet exemple sur **LapaNinja** : <https://www.lapa.ninja/post/stuart/>

J'ai continué ma veille concurrentielle, et exploré des sites proposant des fonctionnalités similaires (sites d'agences de travail temporaire, site d'informations jeunesse, etc.)

J'ai consulté les sites d'Indeed, Monster, et Adecco.

En accord avec la cliente, j'ai réfléchi à un design épuré et simpliste pour rendre l'accessibilité et l'utilisation de celui-ci plus facile. Il fallait que le site puisse plaire et attirer à la fois les jeunes et à la fois les professionnels.

Parallèlement, j'ai pris le parti de consulter un étudiant en Web Design afin d'avoir son avis et suivre ses précieux conseils.



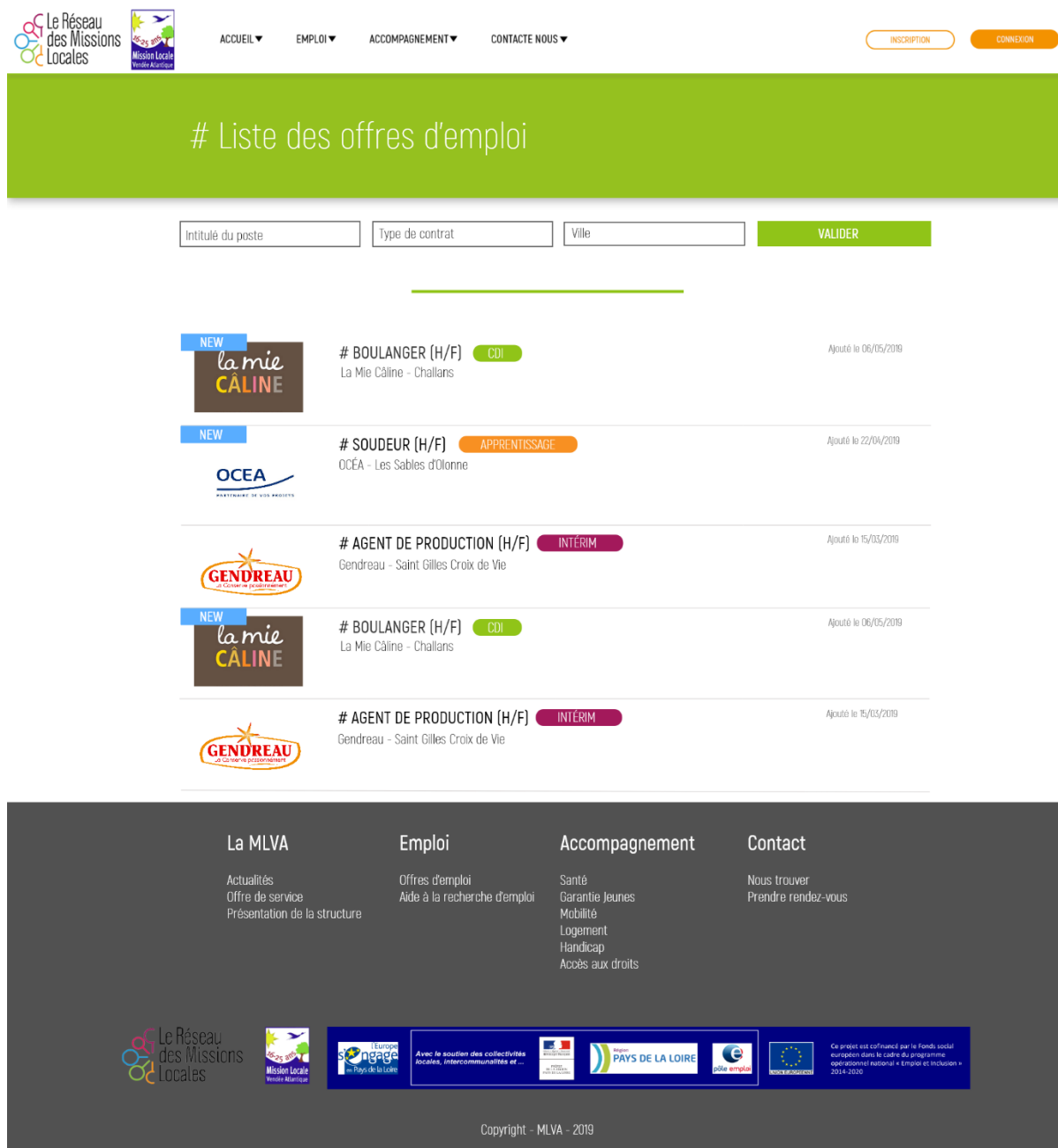
Design - Page d'accueil 1

Pour rassurer la cliente, j'ai donc créé différentes pages afin d'établir un design moderne qui respectait à la fois la charte graphique de l'UNML (Union Nationale des Missions Locales – Annexe n°3) et à la fois la structure préétablie des wireframes.

Pour réaliser cette maquette, j'ai utilisé le logiciel **Lunacy** (le versant de **Sketch** sur Windows).

Après quelques jours d'apprentissage à l'utilisation du logiciel, je me suis lancé dans la construction de la page d'accueil et la page de la liste des offres d'emploi.

Les images proviennent du site **Adobe Stock** et les icônes du site **FlatIcon.com** et **FontAwesome**.



La cliente fut ravie de pouvoir se rendre véritablement compte du rendu visuel du site. Elle l'a validé après l'avoir montré aux Élus composant le bureau de la Mission Locale Vendée Atlantique.

La maquette en **HTML 5**, **CSS3** et **Javascript** fut une étape longue et importante du projet. En effet, il fallait présenter la maquette complète lors de l'Assemblée Générale de l'association le 6 juin 2019.

J'ai démarré par la page d'accueil, qui pour moi est la page la plus importante du site car elle doit donner envie à la fois aux jeunes et à la fois aux employeurs de naviguer le plus longtemps possible.

J'ai utilisé le **HTML 5** pour intégrer le contenu du site en veillant à utiliser les balises optimales pour le référencement naturel (Header, Nav, Main, Footer, « alt » sur les images, attribut « title », etc.)

Pour le style **CSS**, j'ai tout d'abord appliqué un « Reset CSS », qui malgré l'utilisation de **Bootstrap** m'a permis de mettre mes feuilles de style à « zéro ».

```
/* http://meyerweb.com/eric/tools/css/reset/ v2.0 | 20110126 License: none */
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote,
pre, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp,
strike, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label,
legend, table, caption, tbody, tfoot, tr, th, td, article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup, menu, nav, output, ruby, section,
summary, time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section
{
    display: block;
}
body {
    line-height: 1;
}
table {
    border-collapse: collapse;
    border-spacing: 0;}
```

J'ai donc par la suite, choisi d'utiliser le framework **Bootstrap 4** pour ma maquette. Son utilisation m'a permis de développer plus rapidement la partie frontend en utilisant notamment les différentes classes proposées.

L'utilisation de ce framework possède aussi l'immense avantage d'être responsive. J'ai utilisé les différentes classes disponibles pour adapter le frontend en fonction du média utilisé :

- col-xl pour les très grands écrans de plus de 1200px
- col-lg pour les grands écrans de plus de 992px
- col-md pour les écrans moyens de plus de 768px
- col-sm pour les petits écrans de plus de 576px

```
<!-- BLOCS YOUNG & RECRUITER -->

<section class="container header">
  <div class="row ">
    <div class="col-md-6 col-sm-12 p-0 m-0">
      <div class="bloc young">
        
        <h3>Pour les jeunes</h3>
        <p>Je prends contact avec des employeurs qui recrutent</p>
        <a href="/emploi/liste-des-offres"><button class="young-btn"><span>Voir
les offres d'emploi</span></button></a>
      </div>
    </div>
    <div class="col-md-6 col-sm-12 p-0 m-0">
      <div class="bloc recruter">
        
        <h3>Pour les employeurs</h3>
        <p>Je crée mon offre en moins de 2 minutes!</p>
        <a href="/emploi/ajouter-offre"><button class="recruter-btn" type="button">
          <span>Publier une offre d'emploi</span>
        </button></a>
      </div>
    </div>
  </div>
</section>
```

Parallèlement, afin d'optimiser mon code et le rendre « plus élégant », j'ai choisi d'utiliser le préprocesseur **SASS**. Cela me permet de modifier plus rapidement des éléments courants comme les couleurs, les polices, en les rassemblant dans des variables. Mais cela m'a aussi permis d'utiliser des fonctions facilitatrices pour le développement (lighten, darken, etc.).

Je gardais en tête l'idée que le site puisse être le plus flexible possible.

Pour répondre à certains besoins de la cliente, j'ai utilisé des librairies externes :

- **Leaflet**, une bibliothèque javascript pour utiliser les cartographies afin d'indiquer la localisation des différentes antennes.
- **Juicer**, un intégrateur de page de réseau social. Cela permet d'intégrer les publications de la page Facebook de la Mission Locale Vendée Atlantique.

MAJ 09/08/2019 : Finalement j'ai décidé de supprimer **Juicer**, car le rendu n'était pas satisfaisant, et son utilisation affectait considérablement les performances du site.

- **CKeditor**, éditeur de texte open source de type WYSIWYG pour les textareas
- **FontAwesome**, pour l'utilisation des icônes
- **jQuery**, bibliothèque Javascript facilitant grandement la mise en place de fonctions

Afin de fluidifier certaines parties du site et de le rendre plus interactif (impression d'une offre d'emploi, inscription, etc.), j'ai mis en place quelques scripts Javascript.

Pour rendre mon code plus « propre » et facile à lire, j'ai pris soin de bien indenter chaque page avec l'extension **Beautify** de Visual Studio Code. De même, j'ai été vigilant à commenter mon code le mieux possible, en français et en anglais, pour qu'il puisse être réutilisé facilement par un autre développeur et surtout pour ne pas oublier la logique du code.

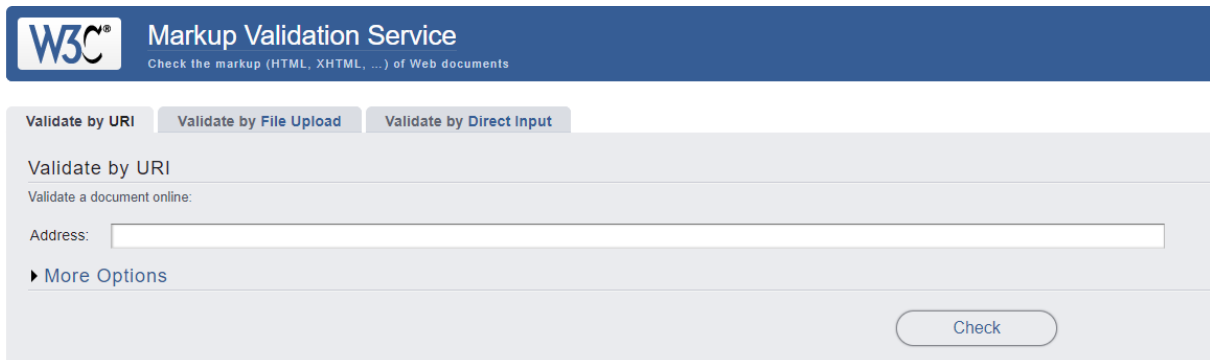
Pour la compatibilité entre les différents navigateurs et selon les recommandations du W3C, j'ai utilisé l'extension **AUTOPREFIXER** de Visual Studio Code. En effet, cette extension m'a permis d'ajouter les préfixes CSS très facilement des différents navigateurs :

- o- pour **Opera**
- moz- pour Gecko (**Mozilla**)
- webkit- pour Webkit (**Chrome, Safari, Android...**)
- ms- pour Microsoft (**Internet Explorer**)

```
.header .button1 {
  font-family: 'akrobat', sans-serif;
  font-weight: 700;
  font-size: 15px;
  text-transform: uppercase;
  color: #fff;
  letter-spacing: 0.025em;
  border: none;
  background: rgb(149, 193, 31);
  border: #fff 2px solid;
  padding: 5px 0 5px 0;
  cursor: pointer;
  border-radius: 20px;
  min-width: 220px;
  overflow: hidden;
  position: absolute;
  margin-top: 30px;
  left: 50%;
  transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
}

.header .button1 {
  font-family: 'akrobat', sans-serif;
  font-weight: 700;
  font-size: 15px;
  text-transform: uppercase;
  color: #fff;
  letter-spacing: 0.025em;
  border: none;
  background: rgb(149, 193, 31);
  border: #fff 2px solid;
  padding: 5px 0 5px 0;
  cursor: pointer;
  border-radius: 20px;
  min-width: 220px;
  overflow: hidden;
  position: absolute;
  margin-top: 30px;
  left: 50%;
  -webkit-transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
}
```

Pour finir, je m'assure des bonnes pratiques de codage grâce au **validator** du W3C :



Et en fonction des résultats, je rectifie les erreurs de codage.

Une fois la maquette terminée, elle fut présentée à l'Assemblée Générale de l'association. Le site reçut un très bon accueil et les retours furent très bons. Je pouvais alors me lancer dans le développement final du site.

DEVELOPPEMENT BACK-END

TECHNOLOGIES ET MODULES NPM UTILISES

Pour la partie front-end, j'ai découpé la maquette **HTML/CSS/JS** avec le moteur de templating **Handlebars**.

Pour le développement back-end de l'application, j'ai choisi d'utiliser **NODE JS** pour plusieurs raisons :

- La rapidité d'exécution : requêtes asynchrones
- L'avantage d'utiliser un seul et même langage : Javascript
- La multitude de modules **NPM** existants
- La démocratisation de son utilisation

Et enfin, pour la base de données, j'ai opté pour l'utilisation de **MongoDB**. La gestion de cette base de données est simple et surtout flexible en modifiant très facilement le contenu des collections.

Pour faciliter le développement de la partie front-end et back-end, j'ai par conséquent utilisé plusieurs modules via le gestionnaire de package **NPM** :

```
"algoliasearch": créer un moteur de recherche et faciliter le filtrage
"bcrypt": pour crypter les mots de passe
"body-parser": pour récupérer les infos dans les requêtes (au format JSON)
"connect-flash": pour la gestion des messages de succès ou d'erreurs
"connect-mongo": Gestionnaire de session via la base de données
```

```

"express": module pour créer un serveur local
"express-fileupload": gestion de chargement de fichier
"express-handlebars": moteur de templating
"express-session": gestion des sessions utilisateur
"handlebars-helpers": création d'outils d'aide au développement
"handlebars.moment": gestion des données de type « date »
"jsonwebtoken": création de token
"jwt-decode": décryptage de token
"lodash": utilitaire pour faciliter l'écriture du code
"method-override": paramétrer les méthodes des différentes requêtes
"mongoose": connexion à la base de données
"mongoose-algolia": connexion entre la base de données et algolia
"multer": gestion de chargement des fichiers
"nodemailer": gestion des envois de mail
"validator": outil de validation de données

```

Pour finir, j'ai choisi d'organiser mon code sous forme de **Model View Controller**. Le dossier **View** représente les pages Handlebars visibles par le client, les **Models** correspondent aux différentes collections de la base de données et les **Controllers** (modifiés en Routes dans mon code) concernent les différentes requêtes de chaque route.

TEMPLATING

Concernant la partie front-end, comme précisé ci-dessus, j'ai utilisé le moteur de templating **Handlebars**. Celui-ci permet de faire la partie views de mon MVC.

J'ai construit un layout principal, puis j'ai créé un fichier handlebars par page du site qui vient compléter le layout à chaque fois que la page est appelée.

L'utilisation d'handlebars me permet ensuite d'appeler les différents contenus de mes collections (voir chapitre suivant) en les appelant sous forme de boucle via la syntaxe : `{{#each nomDuModel}} {{/each}}`.

```

<div class="container bodyPage">
  <div class="row">
    <div class="col-md-12">
      {{#each articles }}
      <a href="/mlva/article/{{slug}}">
        <div class="box">
          <div class="row">
            <div class="col-md-3 cover-size">
              <div class="cover ">
                
              </div>
            </div>
          </div>
        </div>
      </a>
      {{/each}}
    </div>
  </div>
</div>

```

```

<div class="col-md-9">
  <div class="box-header">
    <time class="createdTime mt-2">Ajouté le {{moment createDate
format="DD-MM-YYYY"}} par {{author}}</time>
    <h3 class="title mt-4"># {{title}} </h3><br>
  </div>
  <div class="d-flex flex-row flex-wrap ml-3"> <span
class="box-tag-1 mr-1 w-auto">{{category.category}}</span>

  <!-- BADGE NOUVEAU DURÉE : 3 jours / NEW TAG DURING : 3 days - -->
    <span class="box-tag-3 mr-1 w-auto">{{city}}</span>
    {{#ifBigger convertedDate ../timestamp}}
    <span id="new-tag" class="label p-1 label-info green text-
white">NOUVEAU</span>
    {{/ifBigger}}
  </div>
  <div class="actu-description">
    {{{description}}}
  </div>
</div>
</div>
</div>
</a>
{{/each}}
</div>
</div>

```

L'intérêt d'**Handlebars** réside aussi dans l'utilisation des « **Helpers** ». Dans l'exemple ci-dessus, j'ai utilisé un helper `{{#ifBigger}}` qui me permet de mettre en place des conditions directement dans ma page. Dans cet exemple, j'ai défini ce helper comme cela :

```

Handlebars.registerHelper("ifBigger", function(v1, v2, options) {
  if (v1 > v2) {
    return options.fn(this);
  }
  return options.inverse(this);
});

```

Il permet dans ce cas de comparer la date de création de l'article convertie en Timestamp avec une durée que j'ai défini (3 jours). Si elle est supérieure, un badge « NOUVEAU » apparaît sur l'actualité, si elle est inférieure le badge disparaît.

CONSTRUCTION DE LA BASE DE DONNEES ET MODALITES D'ACCES

La base de données a été réalisée avec **MongoDB**, et se calque aux différents modèles paramétrés :

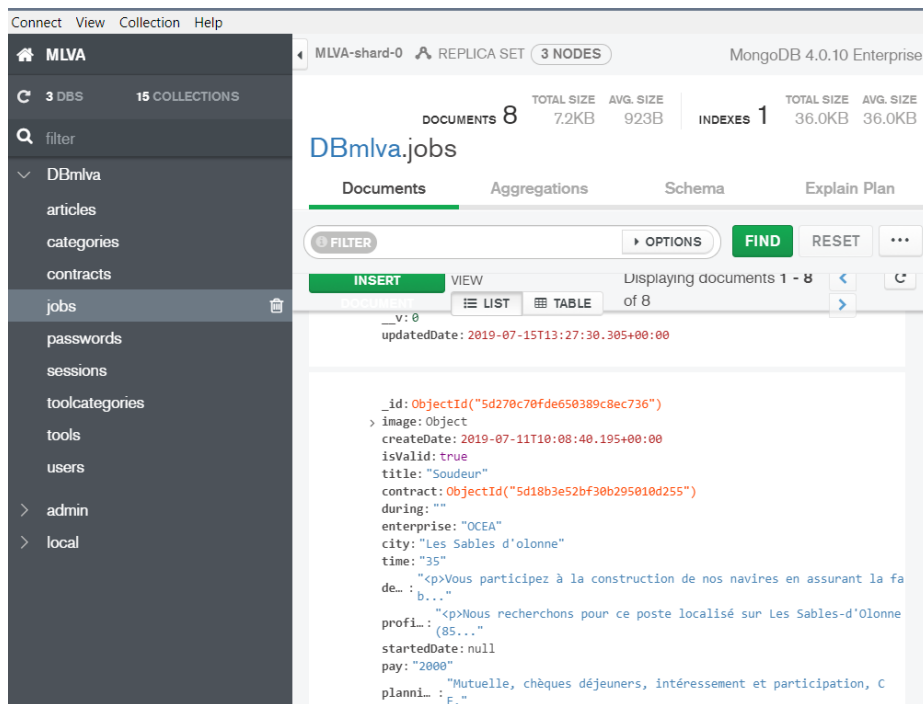
- **Articles** : pour les actualités
- **Categories** : pour les catégories des actualités (Emploi, Formation, etc.)
- **Jobs** : pour les offres d'emploi
- **JobContract** : pour les différents types de contrat de travail (CDI, CDD, etc.)
- **Tools** : pour les outils d'aide à la recherche d'emploi
- **ToolCategories** : pour le type d'outil (CV, Lettre de Motivation, etc.)
- **Users** : pour les informations liées aux utilisateurs
- **Password** : pour générer un token pour réinitialiser le mot de passe

Pour gagner du temps, je me suis servi de deux outils pour travailler la base de données :

- Extension VSC : **Cosmo Azure** (accès direct à la base de données via l'éditeur de texte)

The screenshot shows a VS Code editor window with a dark theme. The top bar indicates the file path: '5d270c70fde650389c8ec736-cosmos-document.json - Mission Locale NodeJS - Visual Studio Code'. The left sidebar has the 'EXPLORER' view open, showing a file tree for 'AZURE: COSMOS DB'. The tree includes folders like 'Attached Database Accounts', '127.0.0.1:27017/mlvadb (MongoDB)', 'mlva-shard-00-02-b28sr.mongodb.net...', 'DBmlva', 'articles', 'categories', 'contracts', 'jobs', 'passwords', 'sessions', 'toolcategories', 'tools', and 'users'. The 'jobs' folder is expanded, and the file '5d270c70fde650389c8ec736' is selected. The main editor area shows the JSON content of this file. The JSON is a document with various fields, including 'image' (a logo), 'name' (a logo name), 'originalName', 'path' (an upload path), 'createdAt' (a timestamp), 'createDate' (another timestamp), 'isValid' (true), 'title' ('Soudeur'), 'contract' (an object with 'soid'), 'during' (empty), 'enterprise' ('OCEA'), 'city' ('Les Sables d'olonne'), 'time' ('35'), 'desc' (a detailed description in French about ship construction), and 'profile' (a job description in French for a candidate). The status bar at the bottom shows 'master*' and 'Live Share' information.

- **Mongo Compass** : permet d'avoir une vue d'ensemble de la base de données et d'en manipuler les différentes collections



Pour effectuer la liaison entre la base de données **MongoDB** et **ExpressJS**, le module **Mongoose** m'a été très utile puisqu'il m'a permis de créer des schémas de collection :

```
const mongoose = require('mongoose')
const ArticleSchema = new mongoose.Schema({
  title: { type: String, required: true },
  ref: { type: String },
  slug: { type: String },
  city: String,
  author: String,
  description: String,
  image: { name: String, originalName: String, path: String, createdAt: Date },
  category: { type: mongoose.Schema.Types.ObjectId, ref: 'category' },
  createDate: { type: Date, default: new Date() }
})
const Article = mongoose.model('Article', ArticleSchema)
module.exports = Article
```

Ces schémas me permettent de construire le contenu des différents éléments du site.

Par exemple ici, une actualité aura comme contenu :

- **Title** : un titre (sous forme de chaîne de caractère, et cette donnée est requise)
- **Ref** : une référence (sous forme de chaîne de caractère)
- **Slug** : permet de générer un identifiant plus lisible notamment dans l'URL
- **City** : la ville concernée par l'actualité (sous forme de chaîne de caractère)
- **Author** : l'auteur de l'actualité (sous forme de chaîne de caractère)
- **Description** : contenu de l'article (sous forme de chaîne de caractère)
- **Image** : une image descriptive, avec un nom, un nom initial, un chemin d'accès et la date d'ajout
- **Category** : la catégorie de l'actualité qui se lie à un autre schéma « category »
- **CreateDate** : la date de création de l'actualité

PLANIFICATION DES SAUVEGARDES DE LA BASE DE DONNEES (APRES LA MISE EN PRODUCTION)

Une fois en ligne, pour éviter la perte des données du site internet, j'ai préféré planifier des sauvegardes régulières de la base de données, afin, en cas d'accident ou de piratage, pouvoir récupérer les données les plus récentes.

MongoDB possède cette fonctionnalité.

Après avoir lu la documentation, j'ai commencé par créer un dossier à la racine de mon serveur : « /db_backups ».

Puis j'ai ajouté ce simple script qui permet de paramétrer l'enregistrement du fichier de sauvegarde :

```
#!/bin/sh
DIR=`date +%m%d%y`
DEST=/db_backups/$DIR
mkdir $DEST
mongodump -h <your_database_host> -d <your_database_name> -u <username> -p
<password> -o $DEST
```

Une fois le chemin de sauvegarde créé, il faut configurer le CRON, afin que la sauvegarde se déroule automatiquement à un moment précis. J'ai donc modifié le fichier cron en utilisant la commande « `sudo crontab -e` » pour ajouter ce paramètre :

```
30 23 * * * ../scripts/db_backup.sh
```

Cette configuration signifie que la sauvegarde sera effectuée tous les mois, toutes les semaines et tous les jours à 23h30 dans le dossier « `db_backups` ».

Si besoin, pour récupérer une des sauvegardes, j'utilise la fonction `mongorestore`, native de **MongoDB** :

```
$ mongorestore --drop -d <database-name> <directory-of-dumped-backup>
```

Je n'ai plus qu'à spécifier la version de la sauvegarde (jour, horaire, etc.), le nom de la base de données concernée et la localisation de la sauvegarde.

RESOLUTION D'UN PROBLEME AVEC UNE DOCUMENTATION ANGLOPHONE

Pour les trois éléments principaux, actualités, offres d'emploi et outils, je devais pouvoir les filtrer en fonction de leurs catégories. En effet, la liste de ces catégories est exhaustive mais peut être amenée à évoluer. La cliente voulait donc que ces choix soient modifiables.

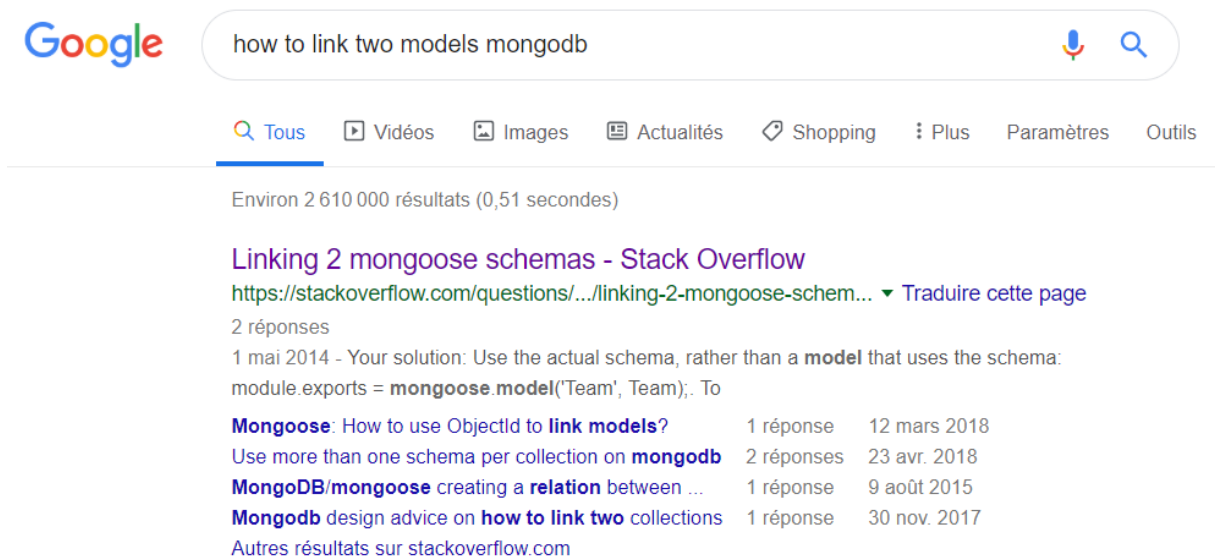
- Catégorie pour les actualités : emploi, formation, santé, etc...
- Type de contrat pour les offres d'emploi : CDI, CDI, intérim, etc...
- Catégorie pour les outils : CV, lettre de motivation, etc.

J'ai donc commencé par faire de nouveaux modèles pour chaque catégorie, en me disant que je pourrai les « appeler » dans mes pages. Mais très rapidement je me suis rendu compte en organisant mon administration que ce n'était pas si simple d'appeler deux modèles différents sur une même page.

J'ai donc entrepris de rechercher la solution sur internet, en écrivant les mots clés suivants dans « Google » :

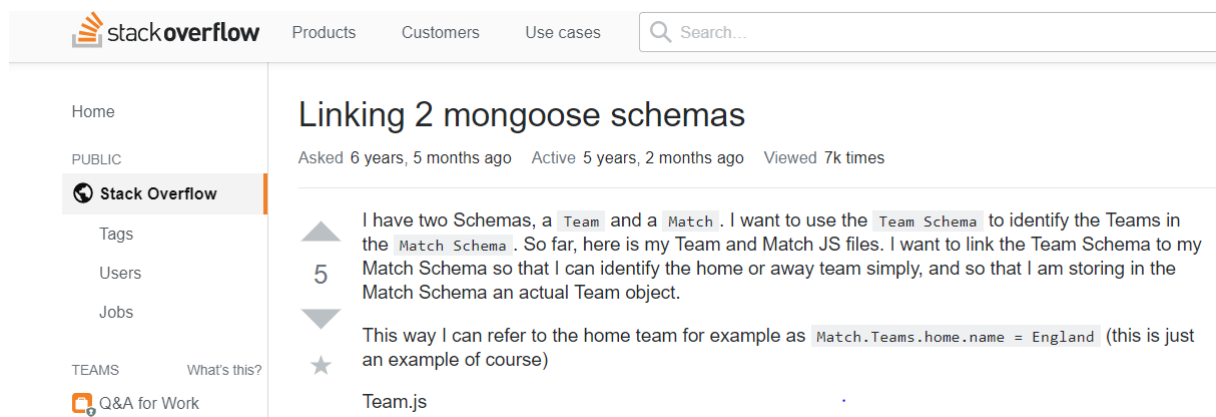
« *how to link two models with mongodb* »

La première réponse de la recherche était :



Google search results for "how to link two models mongodb". The search bar shows the query and a microphone icon. Below the search bar are navigation links: Tous, Vidéos, Images, Actualités, Shopping, Plus, Paramètres, and Outils. The results show approximately 2,610,000 results in 0.51 seconds. The top result is "Linking 2 mongoose schemas - Stack Overflow" with a URL starting with "https://stackoverflow.com/questions/.../linking-2-mongoose-schem...". It has 2 answers and is dated May 1, 2014. The snippet mentions using the actual schema instead of a model. Below the main result are several related links with their respective answer counts and dates: "Mongoose: How to use ObjectId to link models?" (1 réponse, 12 mars 2018), "Use more than one schema per collection on mongodb" (2 réponses, 23 avr. 2018), "MongoDB/mongoose creating a relation between ..." (1 réponse, 9 août 2015), and "MongoDB design advice on how to link two collections" (1 réponse, 30 nov. 2017). A link to "Autres résultats sur stackoverflow.com" is also present.

La réponse était ancienne, mais provenait du site **Stackoverflow**, qui est une référence pour les questions/réponses en informatique.



Stack Overflow question page for "Linking 2 mongoose schemas". The page header includes the Stack Overflow logo, navigation links (Products, Customers, Use cases), and a search bar. The left sidebar shows navigation options: Home, PUBLIC, Stack Overflow (selected), Tags, Users, Jobs, TEAMS, What's this?, and Q&A for Work. The main content area shows the question title "Linking 2 mongoose schemas" with metadata: "Asked 6 years, 5 months ago", "Active 5 years, 2 months ago", and "Viewed 7k times". The question body states: "I have two Schemas, a Team and a Match. I want to use the Team Schema to identify the Teams in the Match Schema. So far, here is my Team and Match JS files. I want to link the Team Schema to my Match Schema so that I can identify the home or away team simply, and so that I am storing in the Match Schema an actual Team object." The question has 5 votes (indicated by a triangle and the number 5). Below the question, there is a partial answer starting with "This way I can refer to the home team for example as Match.Teams.home.name = England (this is just an example of course)" and a code snippet "Team.js".

Un utilisateur détaille sa problématique en expliquant qu'il a deux schémas différents 'team' et 'match'. Il aimerait lier le schéma « team » avec le schéma « match » afin de pouvoir facilement distinguer l'équipe extérieure et l'équipe domicile. Et que le nom de l'équipe soit stocké dans la base de données « match ».

Cette situation ressemblait beaucoup à ma situation, j'ai donc regardé les deux solutions proposées par les autres internautes :

▲ Your solution: Use the actual schema, rather than a model that uses the schema:

2

```
module.exports = mongoose.model('Team', Team);
```

▼ To

```
module.exports = {
  model: mongoose.model('Team', Team),
  schema: Team
};
```

and then `var definition = require('path/to/js');` that, and use `definition.schema` instead of a model directly


Et

▲ You don't want to nest schemas.

1 Try Population in Mongoose: <http://mongoosejs.com/docs/populate.html> That will solve your problem.

share improve this answer

answered Apr 30 '14 at 22:09

 **Yann Eves**
113 ● 7

add a comment

Par reflexe, j'ai cliqué sur le lien proposé dans la deuxième solution, qui suggère l'utilisation d'une documentation pour utiliser la méthode « Population » de **Mongoose** :

MongoDB has the join-like `$lookup` aggregation operator in versions `>= 3.2`. Mongoose has a more powerful alternative called `populate()`, which lets you reference documents in other collections.

Population is the process of automatically replacing the specified paths in the document with document(s) from other collection(s). We may populate a single document, multiple documents, plain object, multiple plain objects, or all objects returned from a query. Let's look at some examples.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const personSchema = Schema({
  _id: Schema.Types.ObjectId,
  name: String,
  age: Number,
  stories: [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});

const storySchema = Schema({
  author: { type: Schema.Types.ObjectId, ref: 'Person' },
  title: String,
  fans: [{ type: Schema.Types.ObjectId, ref: 'Person' }]
});

const Story = mongoose.model('Story', storySchema);
const Person = mongoose.model('Person', personSchema);
```

So far we've created two `Models`. Our `Person` model has its `stories` field set to an array of `ObjectId`s. The `ref` option is what tells Mongoose which model to use during population, in our case the `Story` model. All `_id`s we store here must be document `_id`s from the `Story` model.

La documentation explique que « MongoDB possède une méthode d'agrégation depuis sa version 3.2 mais que Mongoose propose une alternative plus puissante appelée « populate() » qui permet de se référer à d'autres documents dans des collections différentes.

« Population » permet de remplacer automatiquement des chemins spécifiques dans un document par des documents issus d'autres collections. On peut indiquer un seul document, plusieurs documents, un objet, plusieurs objets ou bien tous les objets retournés depuis une requête. Prenons l'exemple ci-dessus.

Nous avons ici créé deux modèles. Le modèle « person » a un paramètre « stories » qui renvoi un tableau « ObjectId ». L'option « ref » indique à Mongoose à quel modèle se référer. Dans l'exemple, on se réfère au modèle « story ». Tous les « id » stockés doivent provenir des documents du modèle « story ».

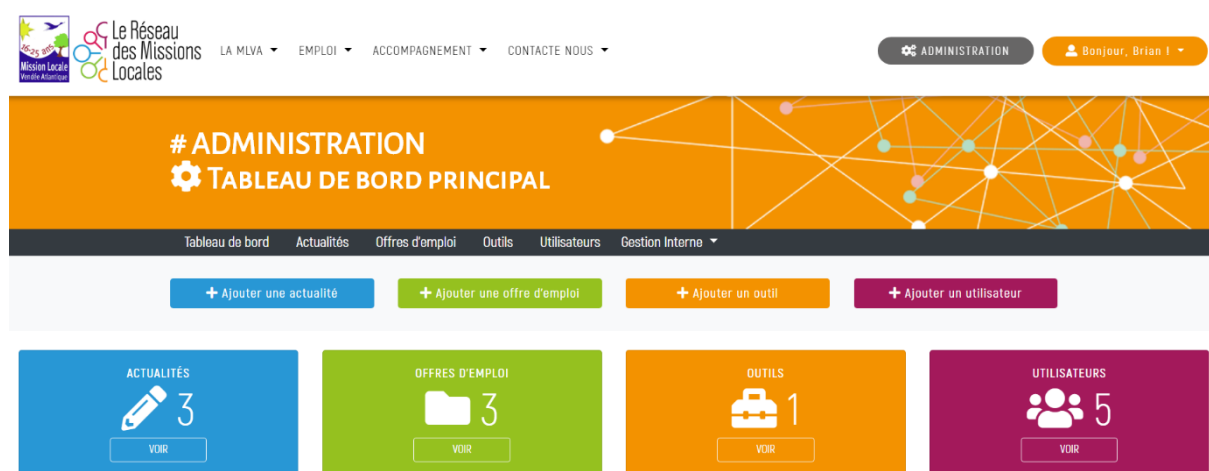
La lecture de cette documentation m'a permis d'optimiser l'organisation de mes collections et d'afficher plusieurs documents de collections différentes. Cette méthode a été un élément clé de mon code car je l'ai utilisé dans plusieurs situations (administration, pages liste, recherche et item individuel (actualités, offres d'emploi, outils), etc.)

MISE EN PLACE DES ROUTES ET DES REQUETES

RESULTAT FRONT-END

Une fois les différents « models » construits, j'ai pu mettre en place mon fichier « **app.js** » qui correspond au fichier de lancement du serveur express.

J'ai alors pu entamer la construction de mes routes en démarrant par la partie ADMINISTRATION du site :




























Le Tableau de bord principal de l'administration du site se matérialise comme ci-dessus pour le client.

A partir de ce tableau de bord, chaque salarié sera autonome dans la gestion des trois items principaux :

- Ajout d'actualités
- Ajout d'offres d'emploi
- Ajout d'outils

La partie utilisateur sera consultable par les modérateurs (les salariés de la MLVA) mais ils n'auront pas la main dessus (voir chapitre Sécurité).

Prenons l'exemple des offres d'emploi (la logique reste la même pour les autres items) ; lorsqu'un administrateur ou un modérateur utilise le tableau de bord, il a une vue d'ensemble des offres d'emploi qui ont été ajoutées sur le site :

Tableau de bord Actualités Offres d'emploi Outils Utilisateurs Gestion Interne ▾								
+ Ajouter une offre d'emploi			+ Ajouter un type de contrat			+ Tout supprimer		
Date	Image	Intitulé du poste	Entreprise	Type de contrat	Voir	Publiée	Modifier	Supprimer
16-07-2019		Electricien	ECCS	CDD				
16-07-2019		Maçon	GEIQ BTP	Intérim				
15-07-2019		Boulangier	La Mie Caline	Contrat d'apprentissage				
12-07-2019		Développeur WEB	Showroom Privé.com	Contrat de professionnalisation				
11-07-2019		Soudeur	OCEA	CDI				
Afficher par : ▾ 1 / 2 ➔								

A partir de ce tableau de bord, l'utilisateur peut ajouter une offre, ajouter un type de contrat, et si l'utilisateur est un administrateur il a aussi la possibilité de tout supprimer.

Pour chaque offre, l'utilisateur peut voir l'offre, la modifier ou la supprimer. Mais surtout, il a la main sur la publication ou non de l'offre sur le site. En effet, lorsqu'un employeur édite une offre d'emploi, il reçoit un mail lui indiquant que sa demande est prise en compte et que l'offre sera publiée très rapidement. Ce laps de temps permet de vérifier les informations de l'offre et d'en modifier les données si nécessaire. Après vérification, il suffit de valider la publication. L'employeur reçoit alors un mail lui signalant que l'offre est bien en ligne.

Pour ajouter une offre, les formulaires de l'administration et de l'employeur sont quasiment identiques (il y a la partie « coordonnées » en plus pour les employeurs) :

1 Type de Poste

Intitulé du poste proposé*:

Type de Contrat Proposé*:

Durée du contrat*:

Durée hebdomadaire:

[Retour](#) [Suivant](#)

2 Coordonnées

Nom de l'entreprise*:

Ville, Commune concernée*:

[Choisir un fichier](#) | Aucun fichier choisi

[Précédent](#) [Suivant](#)

3 Missions Proposées*

[Précédent](#) [Suivant](#)

4 Profil Recherché*

[Précédent](#) [Suivant](#)

5 Modalités et Avantages

Salaire proposé:

Date de démarrage:

Horaires prévus:

Avantages ou compléments d'informations:

[Précédent](#) [Valider](#)

ORGANISATION DU BACK-END ET DES DIFFERENTES REQUETES

Concrètement, du côté serveur, comment cela se passe ? Pour faire fonctionner les formulaires et les tableaux de bord, j'ai mis en place mes différentes routes :

```

const express = require("express");
const router = express.Router();
const dbJobs = require("../../database/models/Jobs");
const dbJobContract = require("../../database/models/JobContract");
const _ = require("lodash");
const fs = require("fs");
const generateSlug = require("../../utils/slugify");
const generateRef = require("../../utils/ref");
const nodemailer = require("nodemailer");
const { smtpUser, smtpPass } = require("../../config/key");
// MULTER

const path = require("path");
const appDir = path.dirname(require.main.filename);
const multer = require("multer");
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "./public/uploads/jobs");
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + "-" + file.originalname);
  }
});
const uploadJob = multer({ storage: storage });

```

On commence en appelant tous les composants dont j'ai besoin pour les différentes routes.

Nous allons rester sur l'exemple de l'affichage des offres, de l'ajout, la modification et la suppression d'une offre d'emploi :

```

router.route("/admin/offres/jobDashboard")
  .get((req, res) => {
    const titlePage = "Administration MLVA - Liste des offres d'emploi";
    const limit = Number(req.query.limit) || 5;
    let page;
    if (req.query.page > 0) {
      req.query.page;
    } else {
      page = 1;
    }
    dbJobs
      .find()
      .populate("contract")
      .sort({ createDate: -1 })
      .skip(limit * page - limit)

```



```

.limit(limit)
.exec((err, job) => {
  if (!err) {
    dbJobs.countDocuments().then(count => {
      dbJobContract.find((err, contract) => {
        res.render("admin/offres/jobDashboard", {
          job: job,
          contract: contract,
          titlePage: titlePage,
          current: page,
          pages: Math.ceil(count / limit),
          next: Number(page) + Number(1),
          prev: Number(page) - Number(1)
        });
      });
    });
  } else {
    res.send(err);}
});
})
.delete((req, res) => {
  dbJobs.deleteMany(err => {
    if (!err) {req.flash('succes_msg', "Toutes les offres d'emploi ont été supprimées.")
    res.redirect("/admin/offres/jobDashboard");
    } else {
      req.flash('errors_msg', "Une erreur est survenue lors de la suppression des offres d'emploi, veuillez réessayer.")
      res.send(err);
    }
  });
});
});

```

Cette première route « `/admin/offres/jobDashboard` » correspond à la page tableau de bord des offres d'emploi. A partir de cette route, plusieurs requêtes sont effectuées :

- **.GET** : la méthode GET, permet d'afficher la page située à `/admin/offres/jodDashboard` créée avec Handlebars. Dans cette page, je commence en lui donnant un nom : « Administration MLVA – Liste des Offres d'emploi ».

Puis j'ai mis en place la pagination :

- En définissant une limite de 5 éléments dans le tableau
- En paramétrant la page par défaut s'il y a moins de 5 éléments

Puis par l'intermédiaire de **Mongoose**, je fais appel aux différents éléments de ma base de données, et plus précisément dans ce cas, à ma collection `Jobs` (défini par `dbJobs`) :

- La fonction **.find** me permet d'accéder à la collection
 - La fonction **.populate** me permet de lier en même temps une deuxième collection « `contract` » pour voir le type de contrat
 - La fonction **.sort** me permet de trier les offres d'emploi par date de création
 - La fonction **.skip** me permet dans le cas de la pagination d'ignorer un certain nombre d'éléments et d'envoyer le reste
 - La fonction **.limit** me permet de fixer une limite d'affichage du nombre d'élément dans le tableau (ici 5)
 - Et enfin la fonction **.exec** permet d'exécuter l'ensemble des méthodes mentionnées précédemment et de lancer le callback qui peut se traduire comme tel :
 - S'il n'y a pas d'erreur, alors compte le nombre d'éléments présents dans la collection. Ensuite appelle moi la collection « `contract` » pour afficher le type de contrat de chaque offre et enfin tu m'affiches la page « `/admin/offres/jobDashboard` » en prenant en compte les données des collections `job` et `contract`.
La constante `titlePage`, de changer le contenu de la balise `<title>` de manière dynamique.
Les constantes `current`, `pages`, `next` et `prev` permettent l'affichage de la pagination.
 - Sinon, affiche moi l'erreur.
- **.DELETE** : la méthode **.delete** me permet de supprimer l'ensemble des offres d'emploi. Je commence par appeler la collection `dbJobs`, puis j'utilise la méthode **.deleteMany**.
Si la suppression fonctionne, je reçois un message via le module **req.flash** et la librairie **Noty** (<https://ned.im/noty/#/>) m'indiquant que la suppression est validée. Sinon, à l'inverse, je reçois un message d'erreur m'invitant à réessayer.

Pour l'ajout d'une offre, une nouvelle route « `/admin/offres/addJob` » est définie. Nous retrouvons deux méthodes sur cette route : **.GET** pour afficher la page Handlebars, et **.POST** pour envoyer les informations dans la base de données.

```

router
  .route("/admin/offres/addJob")
  .get((req, res) => {
    const titlePage = "Administration MLVA - Ajouter une offre d'emploi";

    dbJobContract.find((err, contract) => {
      if (!err) {
        res.render("admin/offres/addJob", {
          contract: contract,
          titlePage: titlePage
        });
      }
    });
  });
})
.post(uploadJob.single("image"), async (req, res) => {
  const file = req.file;
  const slug = await generateSlug(
    dbJobs,
    req.body.title + req.body.enterprise
  );
  const ref = await generateRef(dbJobs, "MLVA-OE");
  const newJob = new dbJobs({
    ...req.body,
    slug,
    ref });
  if (file) {
    newJob.image = {
      name: file.filename,
      originalName: file.originalname,
      path: file.path.replace("public", ""),
      createdAt: Date.now()
    };
  }
  newJob.save(err => {
    if (!err) {
      req.flash('succes_msg', "L'offre d'emploi a bien été enregistrée, n'oubliez pas de la publier")
      res.redirect("/admin/offres/jobDashboard");
    } else {
      req.flash('errors_msg', "Une erreur est survenue lors de l'enregistrement de l'offre, veuillez réessayer.")
      res.send(err);
    }
  });
});
});

```

La méthode **.POST** permet donc d'envoyer les informations contenues dans le formulaire vers la base de données :

On commence par paramétrer l'upload d'image avec le module **Multer** :
`uploadJob.single("image")`

Puis on met en place un callback en asynchrone afin d'augmenter la rapidité de la requête.

On enregistre plusieurs constantes :

File = req.file permet de récupérer le fichier uploadé

Slug = await generateSlug permet grâce à un module indépendant (cf., dossier `utils/slugify.js`) de générer un slug unique

Ref= await generateRef permet également de générer une référence unique pour chaque offre d'emploi (cf., dossier `utils/ref.js`)

Ensuite on initialise un nouveau document en récupérant toutes les informations contenues dans les champs du formulaire (cf. ci-dessus) en fonction de leur attribut « name ».

Si une image a été chargée, on demande au programme de :

- Nommer l'image
- Lui indiquer le chemin où enregistrer l'image
- Enfin, de dater l'upload

Pour finir, toutes les informations recueillies sont sauvegardées dans un nouveau document de la collection. S'il y a une erreur, elle s'affiche avec un message d'erreur, sinon la page est redirigée vers `/admin/offres/jobDashboard` avec un message de succès.

Pour finir mon explication des différentes requêtes, voici l'explication des méthodes **.PUT** et **.DELETE** :

```
router.route("/admin/offres/:id")
  .get((req, res) => {
    const titlePage = "Administration MLVA - Modifier une offre d'emploi";
    dbJobs
      .findOne({ _id: req.params.id })
      .populate("contract")
      .exec((err, job) => {
        if (!err) {
          dbJobContract.find((err, contract) => {
```

```

    res.render("admin/offres/editJob", {
      _id: job.id,
      title: job.title,
      enterprise: job.enterprise,
      contract,
      jobContract: job.contract,
      city: job.city,
      during: job.during,
      desc: job.desc,
      profile: job.profile,
      time: job.time,
      image: job.image,
      startDate: job.startDate,
      pay: job.pay,
      planning: job.planning,
      modalities: job.modalities,
      titlePage: titlePage,
      updatedAt: job.updatedDate
    });
  });
} else {
  res.send("err");
}
});
})
.put(uploadJob.single("image"), async (req, res) => {
  const file = req.file;

  let newData = {
    ...req.body,
    updatedAt: Date.now()
  };
  if (file) {
    const { image } = await dbJobs.findById(req.params.id);
    fs.unlink(`${appDir}/public/uploads/jobs/${image.name}`, err => {
      if (err) console.log(err);
    });
    newData.image = {
      name: file.filename,
      originalName: file.originalname,
      path: file.path.replace("public", "")
    };
  }
  dbJobs.updateOne({ _id: req.params.id }, { $set: newData }, err => {
    if (!err) {

```

```

    req.flash('succes_msg', "L'offre d'emploi a bien été mise à jour")
    res.redirect("/admin/offres/jobDashboard");
  } else {
    req.flash('errors_msg', "Une erreur est survenue lors de la modification de
l'offre, veuillez réessayer.")
    res.send(err);
  }
});
})
.delete((req, res) => {

dbJobs.deleteOne({ _id: req.params.id }, err => {
  if (!err) {
    req.flash('succes_msg', "L'offre d'emploi a bien été supprimée")
    res.redirect("/admin/offres/jobDashboard");
  } else {
    req.flash('errors_msg', "Une erreur est survenue lors de la suppression de
l'offre d'emploi, veuillez réessayer.")
    res.send(err);
  }
});
});
});

```

Comme pour les méthodes précédentes, on accède à la page de modification d'une offre d'emploi via la méthode **.GET**. La particularité cette fois-ci, c'est que l'on demande à chaque champ d'être prérempli en fonction des informations enregistrées préalablement.

- **.findOne ({_id :req.params.id})**, permet de sélectionner l'offre en fonction de son paramètre. Ici le paramètre utilisé est l'ID
- **.populate('contract')**, permet d'accéder à la fois à la collection 'dbJobs' et à la fois à la collection 'contract'
- **.exec** une nouvelle fois, permet de lancer les méthodes mentionnées ci-dessus, puis :
 - S'il n'y a pas d'erreur, affiche-moi la page située à `/admin/offres/editJob` avec les informations déjà saisies :
 - **_id :job.id,**
 - **Title : job.title**
 - Etc.
 - Sinon précise moi l'erreur.

Lorsque la page de modification de l'offre est chargée avec les données demandées, on peut changer les informations présentes dans les différents inputs pour sauvegarder les modifications.

Pour cela, on recommence le même programme que la méthode **.POST** pour récupérer les informations mais pour finir au lieu de sauvegarder un nouveau schéma, on utilise la méthode **.updateOne** pour mettre à jour la collection.

Pour finir, j'utilise la méthode de Mongoose **.deleteOne({_id :req.params.id})** pour supprimer l'offre d'emploi sélectionnée.

En résumé, la construction de mon API REST en utilisant les différentes méthodes de requêtes http, **.GET**, **.POST**, **.PUT** et **.DELETE** m'a permis la mise en place de tout le fonctionnement de l'application WEB, que ce soit pour l'affichage des différentes pages ou que ce soit pour la manipulation des différents contenus liés à la base de données.

Pour m'assurer du bon fonctionnement de mes requêtes j'ai pris soin d'utiliser le logiciel **POSTMAN**.

En effet ce logiciel est très utile pour vérifier que les requêtes de chaque route fonctionnent correctement.

AUTHENTIFICATION ET GESTION DES DROITS UTILISATEUR

Lorsque toutes les routes furent terminées, j'ai pu m'occuper du système d'authentification. Pour cela j'ai utilisé le module **NPM express-session**.

Ce module permet de générer un Cookie dans le navigateur à chaque connexion.

En effet, après s'être inscrit grâce au formulaire, la connexion au site est possible en saisissant son adresse mail et son mot passe.

Avec **express-session**, pour identifier un utilisateur unique, au moment de sa connexion, on lui administre un cookie équivalent à l'ID de l'utilisateur (contenu dans la base de données) :

```
req.session.userId = user._id;
```

Puis pour utiliser ces cookies, on utilise le module **connect-mongo**, qui va permettre de sauvegarder les cookies dans la base de données plutôt que dans la mémoire vive du navigateur.

```
// MONGO STORE
const mongoStore = MongoStore(expressSession);
// EXPRESS SESSION
app.use(
  expressSession({
    secret: "securite",
    resave: false,
    saveUninitialized: true,
    name: "cookie",
    store: new mongoStore({ mongooseConnection: mongoose.connection })
  })
);
```

De plus, au moment de son inscription, chaque utilisateur se voit attribuer un « rôle » :

- Jeune (rôle 4)
- Ou employeur (rôle 3)

Le rôle administrateur est attribué manuellement, de même que le rôle modérateur est validé par un administrateur.

J'ai pris le parti d'appeler pour chaque page, les informations de chaque utilisateur :

```
/ INFO USER
app.use("*", async (req, res, next) => {
  res.locals.user = req.session.userId;
  res.locals.errors = req.flash("errors_msg");
  res.locals.success = req.flash("success_msg");
  if (req.session.userId) {
    const {
      firstName,
      lastName,
      city,
      createDate,
      adress,
      postalCode,
      birthDate,
      birthCity,
      phone,
      mail,
      role
    } = await dbUsers.findById(req.session.userId);

    if (firstName || lastName || city || createDate || role) {
      res.locals.firstName = firstName;
      res.locals.lastName = lastName;
    }
  }
});
```



```

res.locals.city = city;
res.locals.createDate = createDate;
res.locals.adress = adress;
res.locals.postalCode = postalCode;
res.locals.phone = phone;
res.locals.mail = mail;
res.locals.birthDate = birthDate;
res.locals.birthCity = birthCity;
res.locals.role = role;
}
}
next();
});

```

De cette manière, à l'aide des Helpers (voir chapitre précédent), j'ai pu ouvrir ou non l'accès à certaines pages :

```

{{#ifEqual role 3}}
  <div class="text-uppercase d-flex ">
    <a title="Ajouter une offre d'emploi" href="/emploi/ajouter-offre" id='add-job-
btn' class="blue" ><i class="far fa-plus-square"></i><span> Ajouter une offre
d'emploi</span></a>
  </div>
{{/ifEqual}}
{{#ifLess role 3}}
  <div class="text-uppercase d-flex ">
    <a title="Administration du site" href="/admin/main/mainDashboard"
id='admin-btn' ><i class="fas fa-cogs"></i> <span>
ADMINISTRATION</span></a>
  </div>
{{/ifLess}}
{{#if user}}
  <div class="nav-item dropdown">
    <a title="Mon espace" id="login-btn" class="dropdown-toggle text-
capitalize" href="#" role="button" data-toggle="dropdown"
aria-haspopup="true" aria-expanded="false">
      <i class="fas fa-user-alt"></i><span > Bonjour, {{firstName}} ! </span>
    </a>
    <div class="dropdown-menu dropdown-menu-right mt-2 " aria-
labelledby="dropDown">
      {{#ifEqual role 3}}
        <a class="dropdown-item" href="/login/espace-professionnel"><i class="fas
fa-cog"></i> Mon espace</a>
      {{/ifEqual}}
      {{#ifDiff role 3}}
        <a class="dropdown-item" href="/login/espace-personnel"><i class="fas fa-
cog"></i> Mon espace</a>
      {{/ifDiff}}
    </div>
  </div>
{{/if user}}

```

```

        {{/ifDiff}}
        <a class="dropdown-item" href="/login/logout"><i class="fas fa-power-
off"></i> Se déconnecter</a>
    </div>
</div>
{{else}}
<a href="/login/connexion" id="login-btn">Connexion</a>
{{/if}}
</div>
</nav>

```

Prenons l'exemple de la navbar. Si un utilisateur est connecté ({{#if User}}) le bouton connexion se transforme en « Bonjour 'Nom de l'utilisateur' », et donne accès à un dropdown pour aller vers son espace (personnel, si le rôle est différent du rôle 3 ou professionnel si le rôle est équivalent à 3) ou se déconnecter.

De plus, si l'utilisateur connecté a le rôle 3 (= employeur) il peut accéder au bouton « ajouter une offre ». S'il a le rôle 1 ou 2 (administrateur ou modérateur) il a accès au tableau de bord de l'administration.

SECURITE

Pour effectuer ma veille, j'ai étudié les différentes actualités et informations parues sur le site de l'**OWASP**. J'ai voulu commencer à lire toutes les attaques possibles et vu le nombre, j'ai préféré me recentrer sur les principales failles en cherchant dans Google : « owasp attack 2019 top ten ».

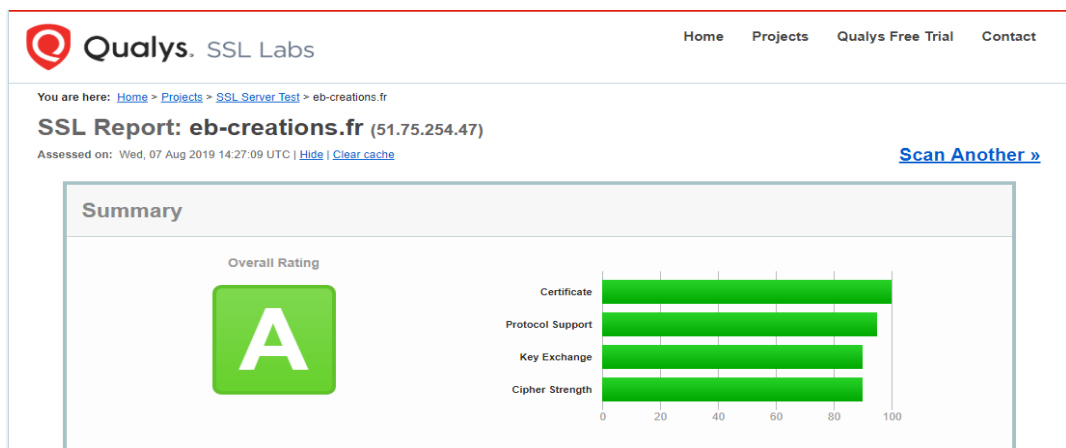
Au fur et mesure de mes recherches j'ai découvert cet article :

<https://geekflare.com/online-scan-website-security-vulnerabilities/>

Il recense les outils d'évaluation de sécurité pour les sites Web. J'en ai effectué plusieurs pour la sécurisation de l'API :

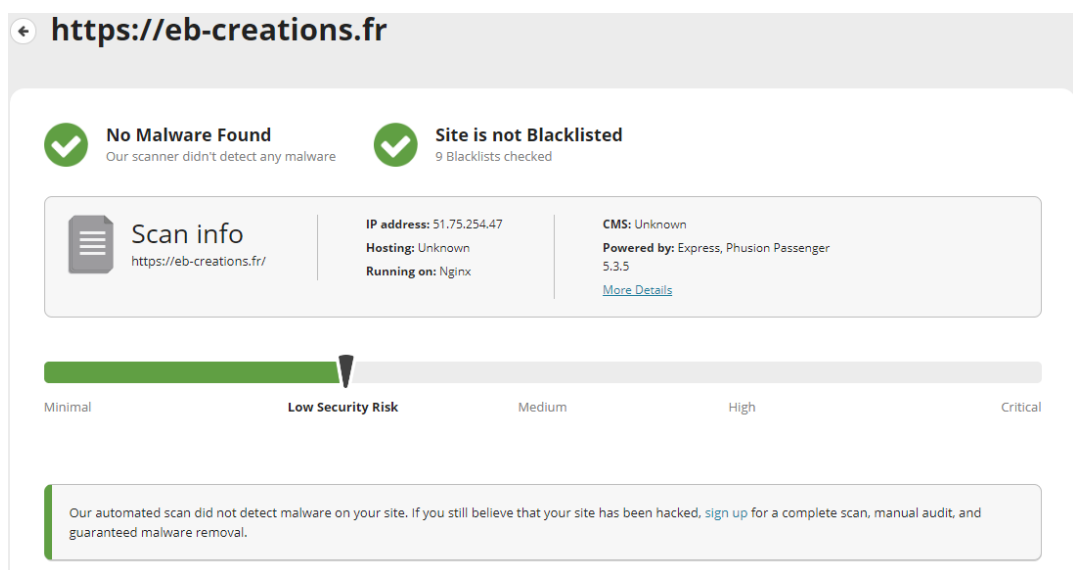
- **Utilisation du protocole HTTPS (après la mise en production) :**

En effet, la première étape de sécurisation du site consiste à utiliser ce protocole afin de crypter les données transmises et reçues pour qu'elles ne soient pas lues. Pour le mettre en place, j'ai utilisé **Let's Encrypt** qui permet de délivrer une certification de validation de domaine. Il est mis en place directement sur le serveur. Afin de vérifier la bonne sécurité de mon serveur j'ai effectué un test sur le site **SSL Labs** qui a donné un très bon résultat :



- Tests de sécurité (après la mise en production):

J'ai ensuite effectué un scanner pour éviter les programmes malveillants, les SPAM, les listes noires de sites WEB etc. avec le site **SUCURI**. Aucun problème détecté :



- Authentification des utilisateurs :

La deuxième étape consiste à sécuriser l'authentification des utilisateurs. Comme nous l'avons vu précédemment, un utilisateur s'identifie par son adresse email et son mot de passe. Pour sécuriser ce dernier, j'ai décidé d'utiliser le module **Bcrypt** intégré directement dans le « model Users » :

```
Userschema.pre("save", function(next) {
  const user = this;
```

```
bcrypt.hash(user.password, 10, (error, encrypted) => {
  user.password = encrypted;
  next();
});
});
```

Grâce à lui, au moment de l'inscription d'un utilisateur, le mot de passe est « haché » pour le crypter. Dans l'exemple ci-dessus, on lui demande de le hacher 10 fois. Dans la base de données, le mot de passe ressemble alors à :

```
"password": "$2b$10$5iTMjXoqlT9Qh1LGSP8To.LmeIBeobEYPbo3jxSNUc10DsAuXofo0",
```

Dans la même logique, au moment d'une inscription, on effectue une validation/confirmation du mot de passe en imposant une structure à celui-ci (8 caractères avec au minimum une majuscule, une minuscule et un chiffre). L'utilisateur doit le notifier deux fois pour s'assurer qu'ils soient identiques. Pour cette opération, j'ai utilisé la ressource **Validate.JS** de **Jquery**.

Elle permet d'établir des règles (exemple : « rules :{password :{required :true }} », et d'afficher le message d'erreur correspondant : « message :{password :{required : "Veuillez indiquer votre mot de passe" }} ».

Enfin, pour sécuriser les connexions de chaque utilisateur, j'ai utilisé le module **express-session** pour générer un token / cookie à chaque session.

- Réinitialisation du mot de passe

Le token est d'ailleurs une nouvelle fois utilisé pour la réinitialisation du mot de passe. L'utilisateur qui en fait la demande, reçoit sur sa boîte mail un URL comportant un token avec les éléments de son compte. Le lien le redirige vers une page pour générer un nouveau mot de passe :

On commence par créer un nouveau model Password :

```
const mongoose = require("mongoose");

const { Schema } = mongoose;

const PasswordSchema = new Schema({
  token: {
    type: String,
    required: true,
    unique: true
  },
});
```

```

user: {
  type: Schema.Types.ObjectId,
  ref: "Users"
},
date: {
  type: Date,
  default: new Date()
}
});

const Password = mongoose.model("Password", PasswordSchema);
module.exports = Password;

```

Puis on crypte les informations de l'utilisateur dans le token :

```

.post(async (req, res) => {
  console.log(req.body);

  const user = await dbUsers.findOne({ mail: req.body.mail });

  if (!user) {
    req.flash(
      "errors_msg",
      "L'adresse email saisie ne correspond à aucun utilisateur"
    );
    return res.redirect("/login/mdpOublie");
  }
  // on crypte l'email qui sera envoyé dans le token
  const payload = {
    mail: req.body.mail
  };
  // on génère un token valable une heure
  const token = await jwt.sign(payload, "n8kjn47hj4", { expiresIn:
3600 });
  // on sauvegarde le token dans le model
  await dbPassword.create({ token, user: user._id });

```

L'utilisateur reçoit alors le mail crypté et peut enfin réinitialiser son mot de passe. S'il y a le bon token, alors il peut réinitialiser son mot de passe, sinon c'est une erreur 404 :

```

.get((req, res) => {
  // SI IL Y A PAS DE TOKEN : REDIRECTION SUR 404
  if (!req.query.token) {
    return res.redirect("/error404");
  }

```

```

    }
    // SINON ON AFFICHE LA PAGE DE REINITIALISATION AVEC LE TOKEN EN
    PARAMETRE
    res.render("login/reinitialiser", { token: req.query.token });
  })

```

Pour finir la réinitialisation du mot de passe on récupère le token et on le décode. On vérifie que les deux mots de passe sont identiques et on le recrypte avec **Bcrypt**. Le mot de passe est mis à jour et la requête avec le token est supprimée :

```

.post(async (req, res) => {
  // ON RECUPERE LES INFORMATIONS NECESSAIRES
  const { password, confirm } = req.body;
  const { token } = req.query;
  // ON VERIFIE QUE LA REQUETE AVEC LE TOKEN EXISTE
  const request = await dbPassword.findOne({ token });
  if (!request) {
    req.flash("errors_msg", "Cette demande a expiré ou n'existe
plus");
    return res.redirect("/");
  }
  // ON DECODE LE TOKEN
  const { mail, exp } = jwt_decode(token);
  const currentTime = Date.now() / 1000

  if(!Validator.equals(password,confirm)){
    req.flash('errors_msg', "Les mots de passe ne sont pas
identiques")
    return res.redirect(`/login/reinitialiser?token=${token}`)
  }
  if (exp < currentTime){
    req.flash("errors_msg", "Cette demande a expiré ou n'existe
plus");
    return res.redirect("/");
  }

  // ON VERIFIE QUE L'UTILISATEUR DANS LE TOEKN EXISTE
  const user = await dbUsers.findOne({ mail });
  if (!user) {
    req.flash("errors_msg", "Cette demande a expiré ou n'existe
plus");
    return res.redirect("/");
  }
  // ON CRYPT LE PASSWORD
  await bcrypt.hash(password, 10, (error, encrypted) => {
    // ON MET A JOUR LE MDP
    user.updateOne({ $set: { password: encrypted } }, err => {
      if (err) {

```

```

        console.log(err);
    }
    });
});
// ON SUPPRIME LA REQUETE QD LE MDP A ETE MIS A JOUR
request.deleteOne();
req.flash("success_msg", "Votre mot de passe a bien été réinitialisé");
return res.redirect("/login/connexion");
});

```

- Sécurisation des routes /admin et /login :

Pour éviter l'accès aux pages administration, j'ai préféré limiter l'accès à ces pages seulement aux administrateurs et aux modérateurs. Les visiteurs, jeunes ou employeurs n'y auront pas accès :

```

// SECURISER LES ROUTES ADMINISTRATION

router.use('/admin/*', async (req, res, next) => {
    const user = await dbUsers.findById(req.session.userId)

    if (!user || (user && user.role > Number(2))) {
        return res.status(401).render('error404')
    }

    next()
})

```

Au même titre que pour chaque utilisateur connecté, on ne peut retourner aux pages /login/connexion, inscription ou mot de passe oublié :

```

// SECURISER LES ROUTES LOGIN SI CONNECTÉ
const routes = [
    "/login/connexion",
    "/login/mdpOublie",
    "/login/inscription-jeune",
    "/login/inscription-pro"
];
router.use(routes, async (req, res, next) => {
    const user = await dbUsers.findById(req.session.userId);
    if (user) {
        return res.status(401).render("error404");
    }
    next();
});

```

- Sécurisation de la base de données et des clés API ou SMTP :

Pour protéger l'accès à la base de données **MongoDB**, j'ai séparé l'adresse du serveur pour qu'elle ne soit pas visible dans le code source. J'ai créé un dossier config et dans un fichier key j'ai ajouté l'accès à la base de données qui comprend notamment le mot de passe. Pour mon adresse SMTP et le mot de passe associé j'ai effectué la même chose :

```
module.exports = {  
  mongoURI: "mongodb+srv://mlvaAdmin:MotdePasseSecret85+@mlva-b28sr.mongodb.net/DBmlva?retryWrites=true&w=majority",  
  smtpUser: 'mail-secret@gmail.com',  
  smtpPass: 'MotdePasseSecret85'  
}
```

- SNYK (après la mise en production)

Pour finir ma batterie de test et de sécurisation des données critiques, j'ai testé le site sur **SNYK**, qui est une plateforme Open-source pour vérifier la vulnérabilité d'une application :

The screenshot displays the SNYK web interface. On the left, there is a sidebar with filters for 'Severity' (High, Medium, Low) and 'Status' (Open, Patched, Ignored). The main content area shows a 'MEDIUM SEVERITY' issue titled 'Denial of Service (DoS)'. The vulnerable module is 'autolinker', introduced through 'handlebars-helpers@0.10.0'. Detailed paths show the vulnerability chain from 'site-mlva@1.0.0' through 'helper-md@0.2.2' and 'remarkable@1.7.4' to 'autolinker@0.28.1'. Remediation is noted as 'No remediation path available'. An overview section describes 'autolinker' as a utility for linking URLs, email addresses, phone numbers, Twitter handles, and hashtags. At the bottom, there are buttons for 'More about this issue', 'Create a Jira issue', 'UPGRADE', and 'Ignore'.

La seule vulnérabilité provient du module Handlebars-Helpers dont le dépôt n'a pas été mis à jour récemment. Sachant que c'est un module essentiel de mon projet, je n'ai pas pu corriger cette faille.

AUTRES FONCTIONNALITES


Afin de perfectionner le site et compte tenu des demandes de la cliente, j'ai ajouté plusieurs fonctionnalités :



Algolia est une API développée par des français, qui permet de gérer facilement un système de recherche. En effet, il suffit de paramétrer dans le modèle souhaité la liaison avec Algolia et de définir les éléments de recherches (nom, ville, auteur, etc..)

Ensuite il suffit d'ajouter une barre de recherche dans la page configurée :

RECHERCHER UNE OFFRE :

Search by  algolia

✓ VALIDER

◀ RETOUR

L'avantage de cette API réside dans son rapport hebdomadaire. Toutes les semaines, je reçois un mail contenant les résultats des différentes recherches, et les mots clés utilisés par les visiteurs.



SendinBlue est une plateforme française de gestion de mail (transactionnel et/ou campagnes d'emailing). La cliente souhaite la mise en place d'une newsletter pour informer régulièrement ses partenaires des événements et actualités de la Mission Locale. J'ai fait une demande d'inscription, puis j'ai paramétré un formulaire pour récupérer les adresses mails qui seront utilisées pour les différentes campagnes de Newsletter :

LA NEWSLETTER



La Newsletter de la MLVA

Inscrivez-vous à notre newsletter pour suivre nos actualités et recevoir les actualités de votre bassin d'emploi.

Veuillez renseigner votre adresse email pour vous inscrire *

EMAIL

Veuillez renseigner votre adresse email pour vous inscrire. Ex. : abc@xyz.com



Je ne suis pas un robot



reCAPTCHA
Confidentialité - Conditions

S'INSCRIRE

L'avantage de cette plateforme est son utilisation indépendante. En effet, cela permet à la chargée de communication de gérer de manière autonome la construction et l'envoi de la newsletter. Le site permet simplement de collecter les contacts.



Nodemailer est un outil spécifique à NodeJs. Il permet l'envoi de mails en fonction des requêtes effectuées côté serveur.

Par exemple, lors de son inscription, un jeune ou un employeur reçoit un mail de confirmation d'inscription pour lui souhaiter la bienvenue.



Bonjour, Brian

Bienvenue sur le site de la Mission Locale Vendée Atlantique!
Toute l'équipe est prête à te recevoir et reste à ta disposition pour répondre à tes questions.
N'oublie pas de personnaliser ton espace personnel! Tu peux y ajouter tes CV pour postuler
aux offres d'emploi. Tu pourras aussi sauvegarder les offres qui t'intéressent pour les
retrouver facilement!
Alors n'hésite plus!

GO !

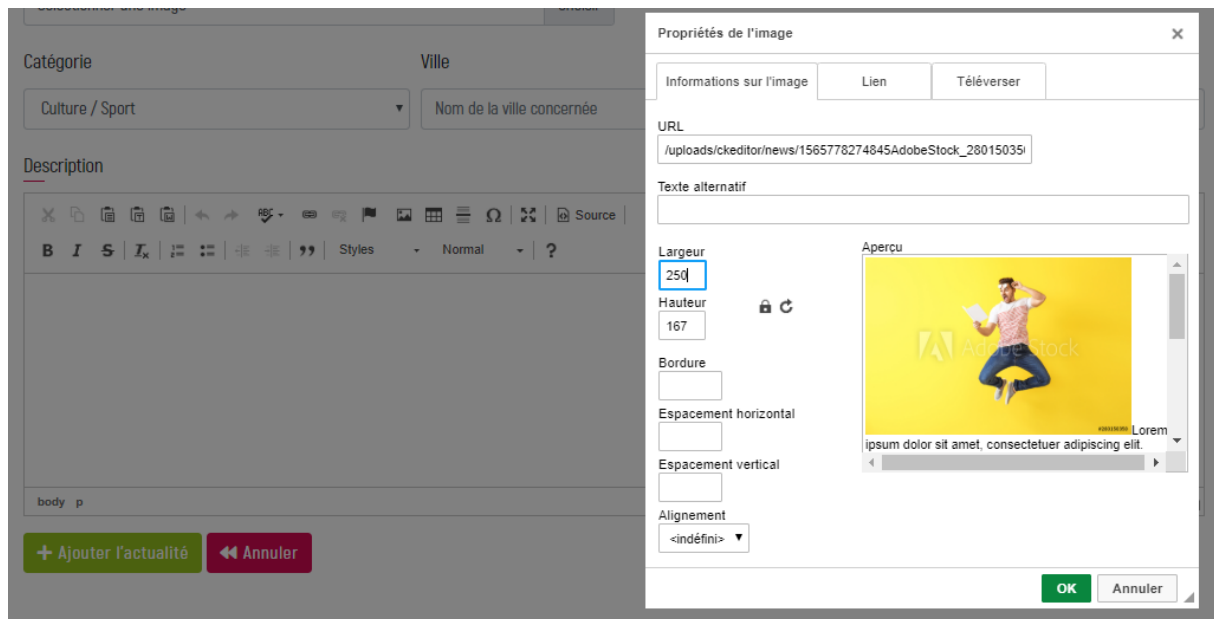


CKEditor 4

Pour améliorer l'expérience utilisateur, j'ai ajouté l'éditeur de texte **CKEditor** dans les différents formulaires comportant un textarea. Cela permet de personnaliser le contenu (ajouter des titres, des images, les listes, etc.).

Pour l'upload d'image, afin de ne pas prendre la version payante de CKeditor, j'ai ajouté, en cherchant sur internet, une nouvelle fonction (téléverser) trouvée sur :

<https://gist.github.com/RedactedProfile/ac48c270d2bbe739f9f3>



TESTS UNITAIRES

J'ai été amené à utiliser des tests unitaires pour perfectionner les slug des actualités et des offres d'emploi. En effet, si l'on prend l'exemple des offres d'emploi, j'ai paramétré le slug en utilisant l'intitulé du poste ajouté au nom de l'entreprise :

```
.post(uploadJob.single("image"), async (req, res) => {
  const file = req.file;
  const slug = req.body.title + req.body.enterprise );
```

Mais une problématique s'est vite révélée. En effet, si une entreprise proposait deux offres avec le même intitulé de poste, cela générerait un conflit puisque deux offres avaient le même slug.

J'ai donc recherché une solution et j'ai trouvé une solution en ajoutant un algorithme permettant d'ajouter un « -1 » si un slug existait déjà :

```
const slugify = (text) => _.kebabCase(text);

async function createUniqueSlug(Model, slug, count) {
  const user = await Model.findOne({ slug: `${slug}-${count}` });

  if (!user) {
    return `${slug}-${count}`;
  }

  return createUniqueSlug(Model, slug, count + 1);
}

async function generateSlug(Model, name, filter = {}) {
  const origSlug = slugify(name);
```

```

    const user = await Model.findOne(Object.assign({ slug: origSlug }, filter),
'id');

    if (!user) {
        return origSlug;
    }

    return createUniqueSlug(Model, origSlug, 1);
}

module.exports = generateSlug;

```

Pour tester le bon fonctionnement de cette fonction « slugify », j'ai utilisé le module **JEST** qui permet de réaliser des tests unitaires.

J'ai ajouté au projet un dossier « TEST » pour créer le fichier slugify.test.js :

```

//On utilise la constante mockUser pour générer une base de données virtuelle
const MockUser = {
  slugs: ['john-jonhson-jr', 'john-jonhson-jr-1', 'john'],
  findOne({ slug }) {
    if (this.slugs.includes(slug)) {
      return Promise.resolve({ id: 'id' });
    }

    return Promise.resolve(null);
  },
};

//Si aucun slug n'existe, renvoi john-johnson
describe('slugify', () => {
  test('no duplication', () => {
    expect.assertions(1);

    return generateSlug(MockUser, 'John Jonhson.').then((slug) => {
      expect(slug).toBe('john-jonhson');
    });
  });
  test('one duplication', () => {
    expect.assertions(1);
    return generateSlug(MockUser, 'John.').then((slug) => {
      expect(slug).toBe('john-1');
    });
  });
  test('multiple duplications', () => {
    expect.assertions(1);
    return generateSlug(MockUser, 'John Jonhson Jr.').then((slug) => {

```

```

    expect(slug).toBe('john-johnson-jr-2');
  });
});
});

```

Dans ce test on réalise en réalité trois tests en prenant l'exemple de trois slug différents : john-johnson-jr/ john-johnson-jr-1/ john-johnson-jr-2

- 1 : Si le slug « john-johnson » n'existe pas, on s'attend à avoir le slug « john-johnson »
- 2 : Si le slug « john » existe, on s'attend à avoir « john-1 »
- 3 : Si le slug « john-johnson-jr » existe plusieurs fois, on attend « john-johnson-2 »

On exécute la commande « npm run test » et voici le résultat :

```

$ npm run test

> site-mlva@1.0.0 test C:\Users\epaud\Desktop\DEV\Projets\Mission Locale NodeJS
> jest --coverage

PASS test/slugify.test.js
  slugify
    ✓ no duplication (5ms)
    ✓ one duplication (1ms)
    ✓ multiple duplications (1ms)

-----|-----|-----|-----|-----|-----|
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files |    100 |    100   |    100   |    100   |                   |
 slugify.js |    100 |    100   |    100   |    100   |                   |
-----|-----|-----|-----|-----|-----|

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        3.644s
Ran all test suites.

```

Les trois tests sont concluants, la fonction marche correctement.

MISE EN PRODUCTION

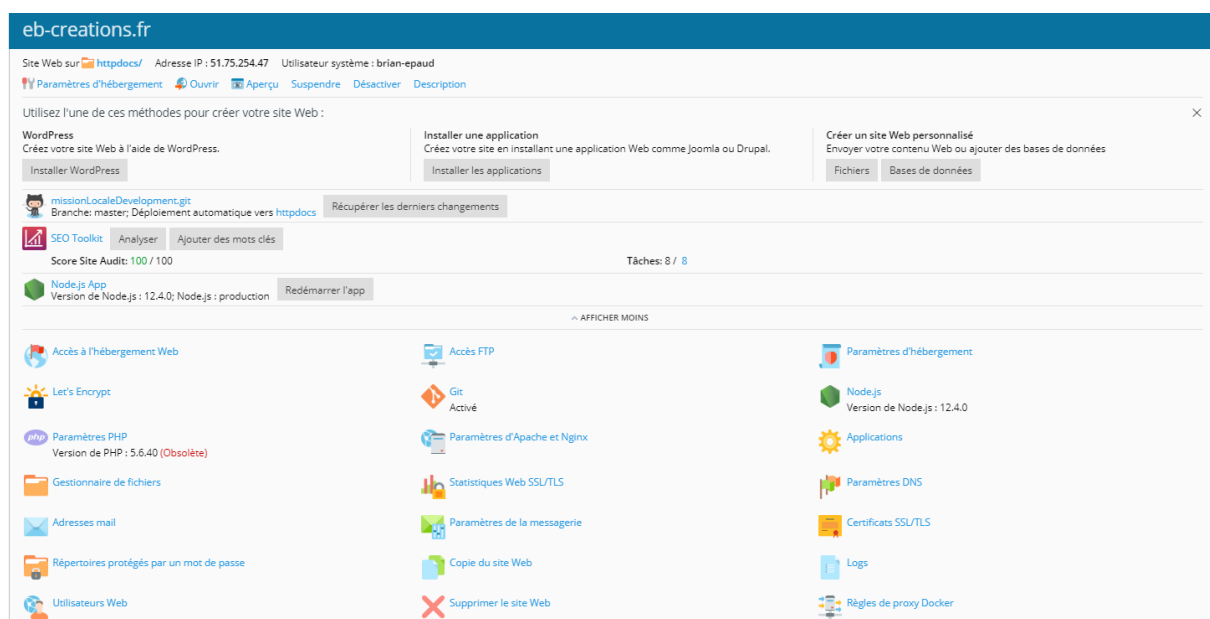
Compte tenu que le site est toujours en développement et pour les besoins de l'examen, j'ai choisi d'héberger le site sur l'un de mes serveurs personnels avec mon propre nom de domaine : **eb-creations.fr**

HEBERGEMENT

Le serveur utilisé est un VPS (Virtual Personal Server) hébergé chez Ionos :



Pour faciliter mes opérations, j'ai opté pour l'utilisation de PLESK afin de gérer plus facilement le contenu du serveur, les mises à jour, la consultation des erreurs, etc...



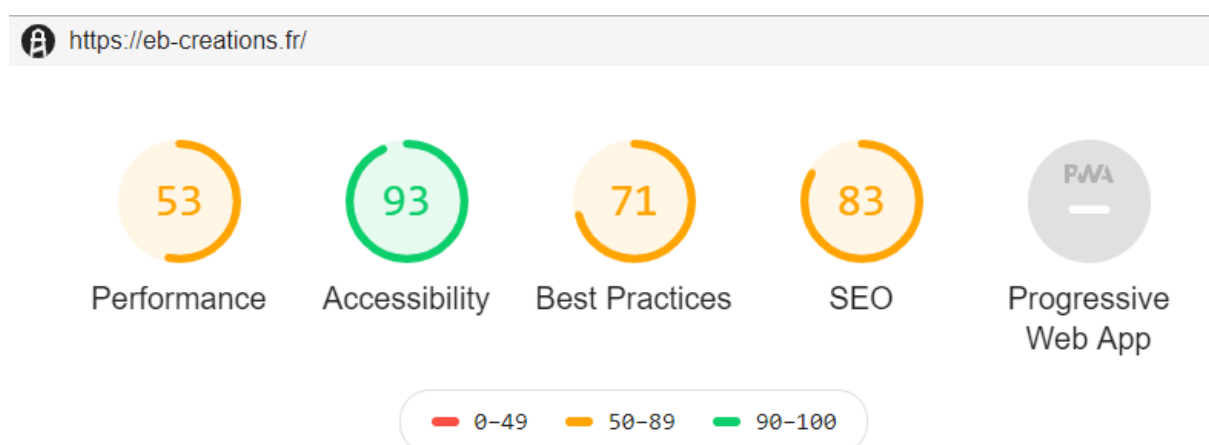
J'ai ensuite importé mon dépôt enregistré sur GitHub dans le dossier « httpdocs » de mon serveur Linux. Puis installé **Node** et **NPM**. Enfin, j'ai redémarré mon serveur afin que tout soit actualisé. Après cette étape, le site était enfin en production !

OPTIMISATION DES PERFORMANCES ET REFERENCEMENT

Le site mis en ligne, j'ai commencé par évaluer différentes caractéristiques de la page d'accueil :

- Performances de la page
- L'accessibilité de la page
- Les bonnes pratiques de programmation
- Le référencement (SEO)

Pour cela, j'ai utilisé l'extension de **Google Chrome** « **LightHouse** ». Cette extension permet d'établir une évaluation de ces 4 items. Voici mon score à la mise en ligne du site :



REFERENCEMENT (SEO)

J'ai commencé par améliorer le référencement (SEO). Je devais notamment ajouter une méta-description dans l'entête de mon site. J'ai mis en place une méta description dynamique pour les pages que je voulais mieux référencer :

```
{{#if descriptionPage}}
<meta name="description" content="{{descriptionPage}}">
{{else}}
<meta name="description"
  content="La Mission Locale Vendée Atlantique est la meilleure solution
  d'information et d'accès à l'emploi pour les jeunes de 16 à 25 ans">
{{/if}}
```

De cette manière, la méta-description est définie par défaut et pour les pages ciblées, il y a une méta-description particulière.

J'ai ensuite paramétré un **robots.txt**, que j'ai ajouté à la racine du site – Pour m'aider à son implémentation j'ai suivi une documentation (<http://robots-txt.com/>). La

configuration du robots.txt empêche les robots d'exploration (web crawler) d'accéder aux pages que j'ai déterminées :

```
robots.txt
1  User-agent: *
2  Disallow: /admin/
3  Disallow: /login/
4  Disallow: /contacte-nous/
5  Disallow: /mlva/presentation
6  Disallow: /mlva/services
7  Disallow: /mlva/equipe
8  Disallow: /contacte-nous/
9  Disallow: /error404/
10 Disallow: /newsletter/
11 Disallow: /mentions-legales/
```

La mention « **User-agent=*** » permet de spécifier que tous les robots d'exploration sont concernés par le robot.txt.

La mention « **Disallow** » permet de notifier les différentes pages que je souhaite bloquer aux robots d'exploration et pour ne pas référencer ces pages.

Pour parfaire le référencement naturel, j'ai ajouté des balises **Open Graph**, dans l'entête du site. En effet, sachant que plusieurs éléments sont partageables sur les réseaux sociaux, il est important de configurer ces balises pour afficher le maximum d'informations possibles :

```
<meta property="og:title" content="{{titlePage}}" />
<meta property="og:type" content="website" />
<meta property="og:url" content="http://eb-creations.fr" />
<meta property="og:image" content="http://eb-
creations.fr/images/logo/logo.jpg" />
<meta property="og:description"
  content="La Mission Locale Vendée Atlantique est la meilleure solution
d'information et d'accès à l'emploi pour les jeunes de 16 à 25 ans" />
```

Enfin, j'ai ajouté une balise « *canonical* » afin d'indiquer aux moteurs de recherche de prendre en compte les résultats de l'URL indiqués dans le cas où il y aurait du contenu identique :

```
<link rel="canonical" href="http://www.eb-creations.fr" />
```

BONNES PRATIQUES

Puis, compte tenu du peu d'élément à modifier, je me suis attaché à améliorer les bonnes pratiques.

En effet, j'ai simplement dû mettre à jour la version de jQuery utilisé par le site et j'ai dû, pour des raisons de sécurité, ajouter l'attribut « *rel=noopener* » à mes liens qui possédaient l'attribut « *target=_blank* ». Cet attribut rendra nulle la valeur de *window.opener* (interdisant donc tout changement d'URL sur la page appelante).

L'ACCESSIBILITE

De la même manière, il manquait peu de choses pour améliorer l'accessibilité de la page. J'ai complété les différents attributs « alt= » qui étaient restés vides et j'ai ajouté l'attribut « aria-label » sur mes liens <a> et les champs des formulaires. J'ai aussi modifié quelques couleurs d'écritures pour que le ratio entre la couleur du background et du texte soit satisfaisant.

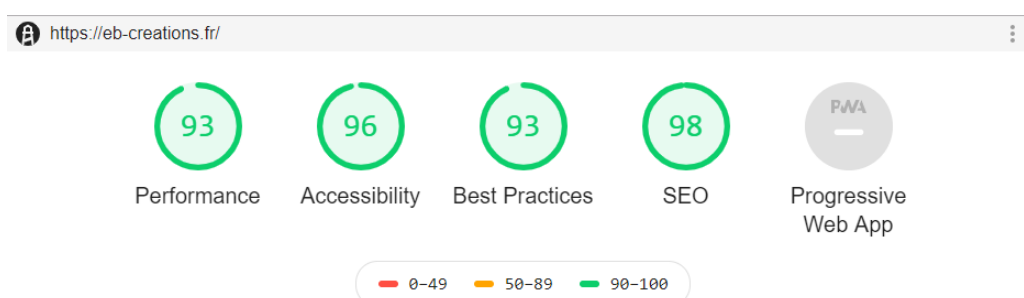
PERFORMANCES

Cette partie fût la plus longue. J'ai commencé par reprendre chaque image pour les compresser et les redimensionner à l'aide du site « I LOVE IMG (<https://www.iloveimg.com/fr>) » afin d'alléger le poids de la page.

Puis, en suivant les conseils d'un autre développeur, j'ai mis en place du Lazy Loading afin que les images se chargent au fur et à mesure du scroll de la page. Pour cela, j'ai utilisé le module « **Lazyload** » (<https://github.com/verlok/lazyload>).

J'ai ensuite sélectionné des versions minifiées de mes différentes bibliothèques Javascript.

Après avoir suivi la plupart des recommandations, voici le score obtenu :

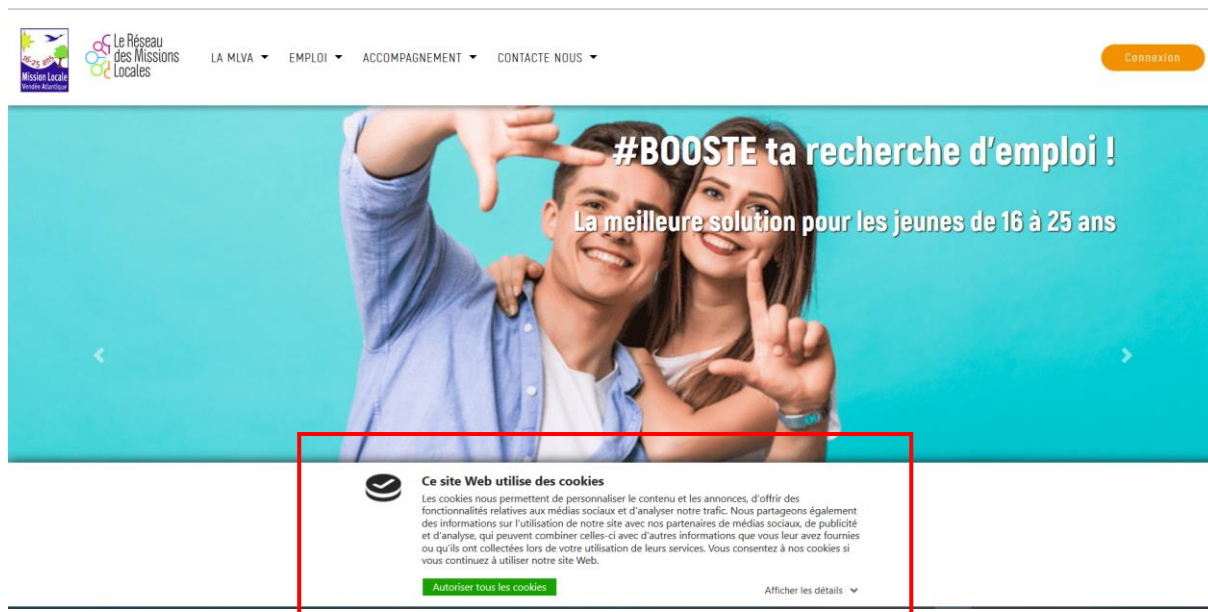


COOKIES / RGPD

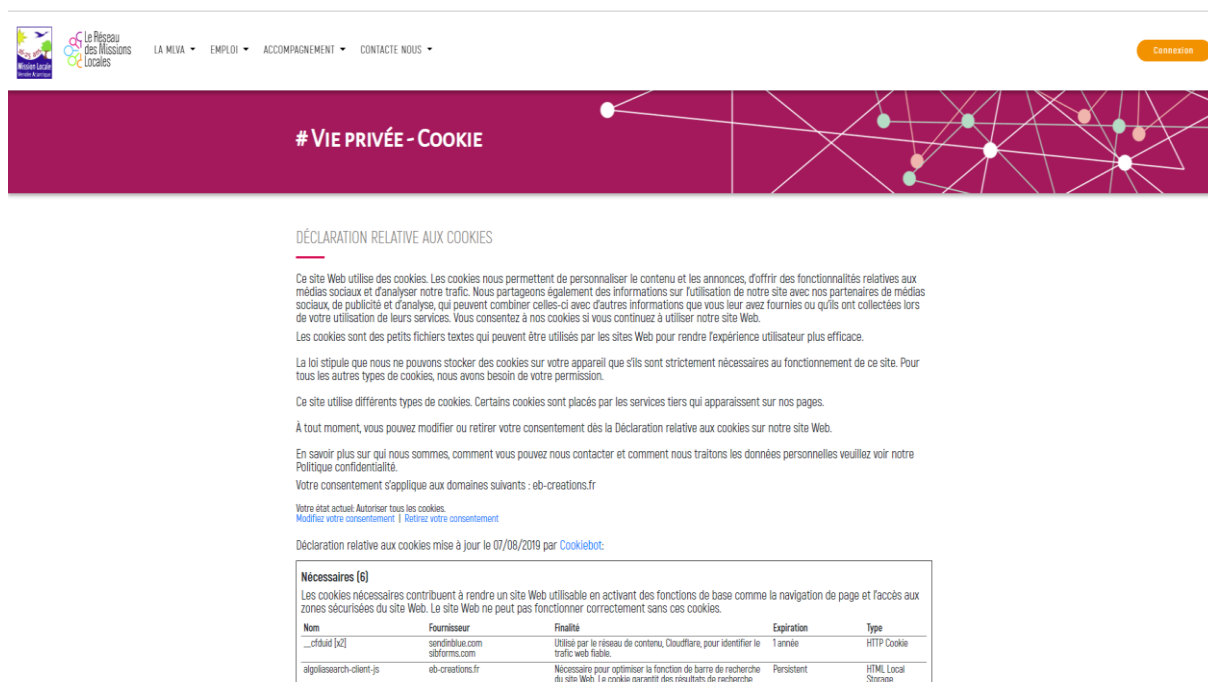
Concernant la RGPD, je suis resté vigilant à l'utilisation des cookies générés par le site. Pour évaluer cela j'ai utilisé le site **CookieBot** (<https://www.cookiebot.com/fr/>). Il a effectué une analyse complète du site :

The figure displays the Cookiebot logo and a screenshot of the 'Rapport d'analyse de cookies' (Cookie Analysis Report). The report includes a 'Récapitulatif' (Summary) section with the following details: Date d'analyse: 07/08/2019, Nom de domaine: eb-creations.fr, Emplacement du serveur: France, and Nombre de cookies au total: 9. The 'Résultat de l'analyse' (Analysis Result) section states: 9 cookies ont été identifiés, 1 cookies ne sont pas classifiés et ont besoin d'un classement manuel et d'une description d'usage.

J'ai donc configuré une bannière de consentement pour les utilisateurs afin qu'ils acceptent l'utilisation des cookies à travers le site.



J'ai ajouté aussi dans le footer l'accès à la page de modification de consentement des cookies afin que chaque utilisateur puisse modifier leur choix.



ANALYSE DES DONNEES

Pour finir ce fabuleux voyage dans la création de ce beau projet, j'ai ajouté deux balises **<script>** dans mon code afin d'étudier les données recueillies à travers le site. Via notamment **Google Analytics** et **Google Search Console** :

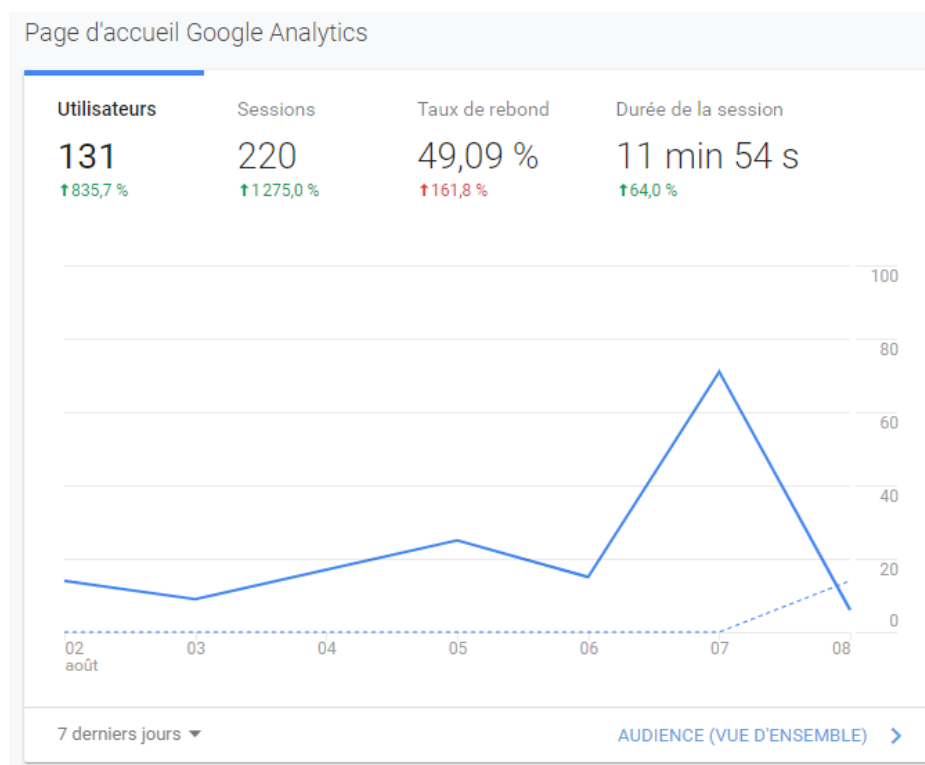
GOOGLE ANALYTICS

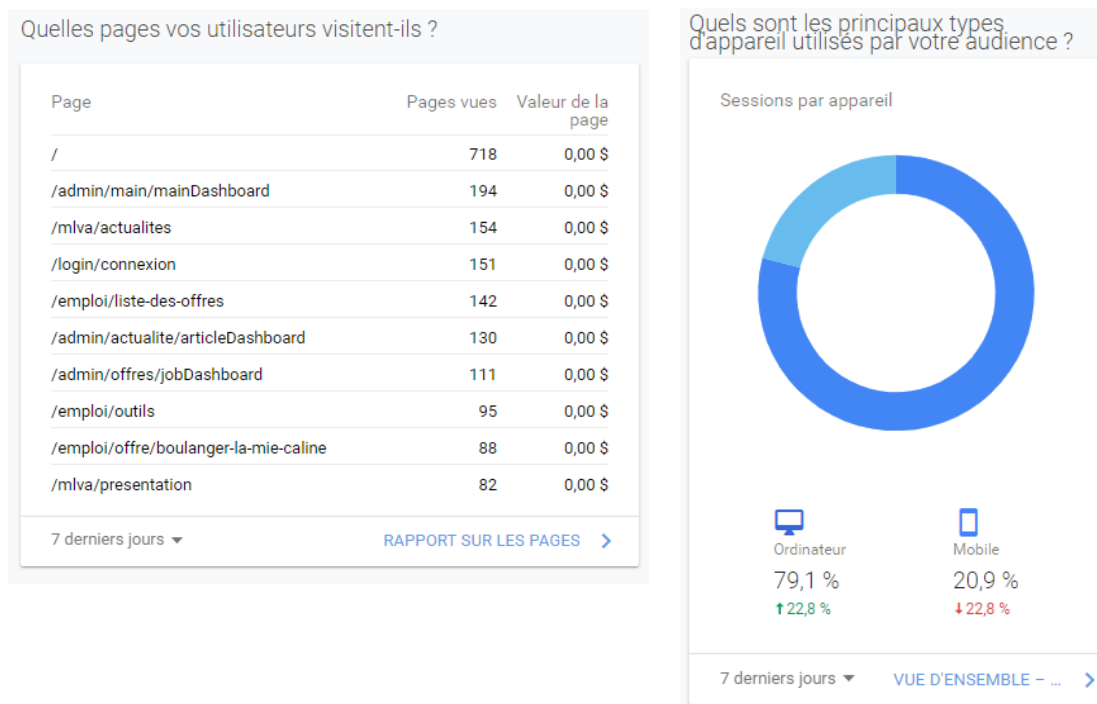
L'ajout de Google Analytics me permet de consulter les données recueillies par la navigation des utilisateurs.

Le nom de domaine n'étant pas le définitif, les résultats n'ont pas véritablement de valeur, mais je compte utiliser à moyen terme ces données pour affiner à la fois la structure du site et augmenter/diminuer le contenu en fonction des pages les plus utilisées.

Parallèlement, Je resterai vigilant sur la répartition de l'utilisation du support utilisé pour consulter le site (ordinateur ou smartphone) afin d'optimiser si nécessaire l'ergonomie du design.

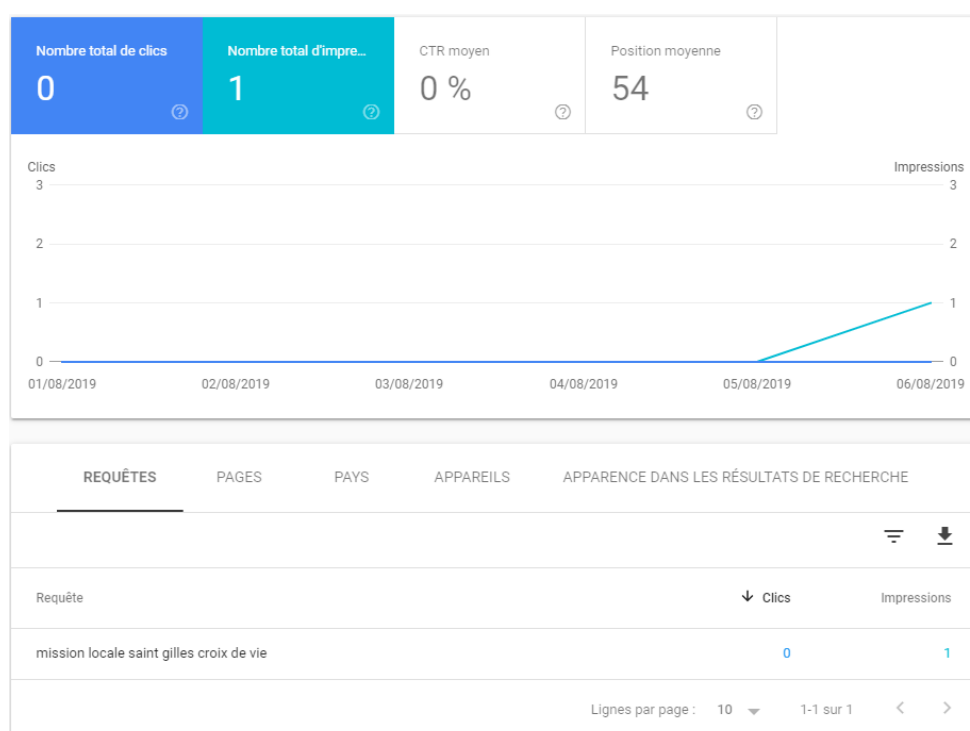
Données relevées au 09/08/2019 :





GOOGLE SEARCH CONSOLE

Cet outil me sera à l'avenir très utile pour augmenter les performances et la visibilité du site dans les résultats de recherche des moteurs de recherche. Comme pour Google Analytics, les données ne sont pas viables avec ce nom de domaine, mais j'ai pu me rendre compte des différents items à consulter :

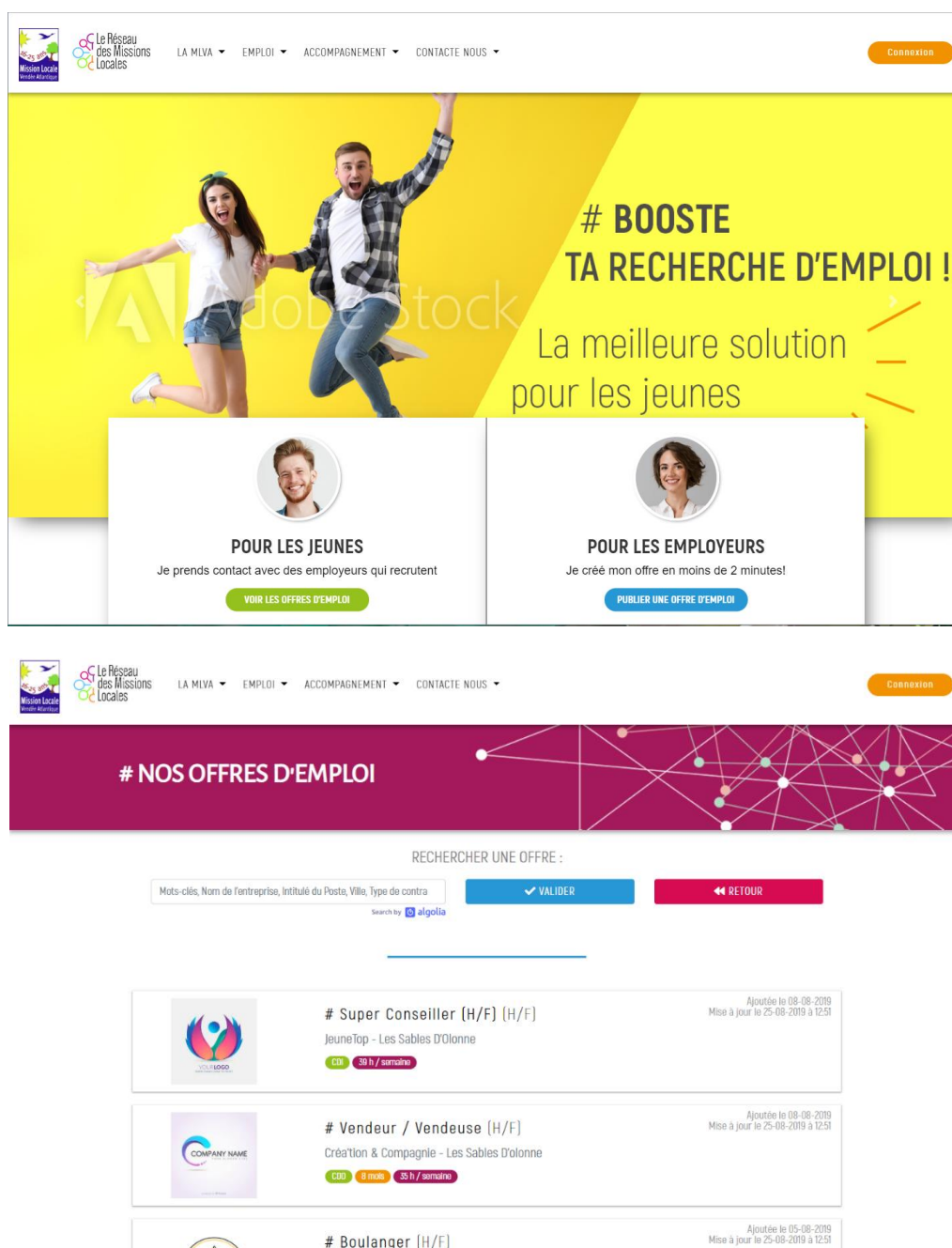



Sur cet exemple, on constate qu'en cherchant « mission locale saint gilles croix de vie » dans Google, le site (eb-creations.fr) se retrouve à la 54^{ème} position. L'objectif futur sera donc de le positionner à la première position.

Je m'attacherai notamment à ajouter un compte **Google My Business** pour promouvoir l'association.

RENDU FINAL

Voici quelques exemples du rendu final du site (annexe n°8) :






Le Réseau
des Missions
Locales

LA MLVA ▾ EMPLOI ▾ ACCOMPAGNEMENT ▾ CONTACTE NOUS ▾

Connexion

SUPER CONSEILLER (H/F) (H/F)



Ajoutée le 08-08-2019
 Mise à jour le 25-08-2019
 Référence de l'offre: mlva-oe
 Je partage sur:
 Facebook Twitter LinkedIn Email

SUPER CONSEILLER (H/F) (H/F) - LES SABLES D'OLONNE

JeuneTop
CDD 39 h / semaine

Description du poste:

Tu sais :

- Faire un CV en 5 minutes
- Faire une lettre de motivation les yeux fermés
- Ecrire un FAJ plus vite que ton ombre
- Remplir i-milo comme personne !

Profil recherché:


Tu es patient, organisé, polyvalent. Tu adores l'animation d'ateliers, l'administratif et le contact avec les jeunes. Alors rejoins-nous l'équipe des supers conseillers de JeuneTop !

Modalités et Avantages:

👉 Salaire: 2000 €

🕒 Date de démarrage: 25-08-2019

⬅️ Retour
 Imprimer
 Postuler à cette offre



Le Réseau
des Missions
Locales

LA MLVA ▾ EMPLOI ▾ ACCOMPAGNEMENT ▾ CONTACTE NOUS ▾

Connexion

NOUS TROUVER

NOS HORAIRES D'OUVERTURE


	Matin	Après-midi	Jour de permanence
Lundi	8h45-12h30	13h30-17h30	Nolmoutier Talmont St Hilaire
Mardi	8h45-12h30	13h30-17h30	Moutiers-les-Mauxfaits Bouvair sur Mer
Mercredi	8h45-12h30	13h30-17h30	
Jeudi	8h45-12h30	13h30-17h30	Clos d'Yeu Talmont Saint Hilaire Saint Jean de Monts
Vendredi	8h45-12h30	13h30-16h	
Samedi / Dimanche	Fermé	Fermé	

NOS QUATRE ANTENNES ET NOS CINQ PERMANENCES

LES SABLES D'OLONNE


24 rue de l'Hôtel de Ville
85100 Les Sables d'Olonne

☎ 02.51.23.36.87 📍 Suivre le GPS !



SAINT GILLES CROIX DE VIE

17 route de l'Aiguillon
85800 Saint Gilles Croix de Vie



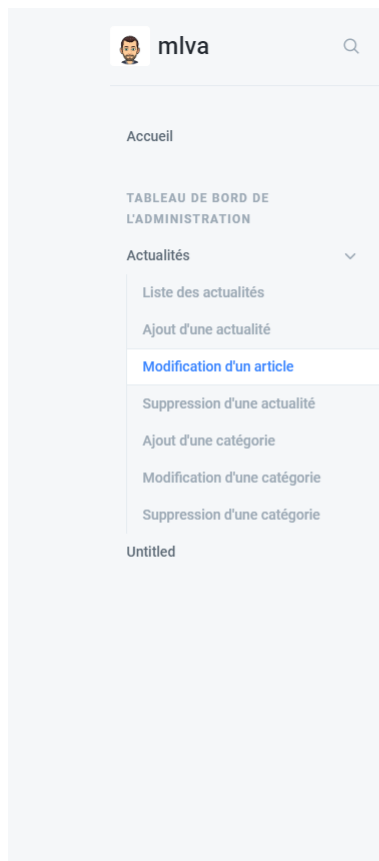
L'une des étapes finales de la réalisation du site de la Mission Locale Vendée Atlantique est de documenter l'API. En effet, si le site et par conséquent le code sont amenés à changer ultérieurement, il est important d'avoir des éléments de repères, que ce soit pour moi, ou pour un autre développeur travaillent sur le site.



J'ai donc commencé la réalisation de la documentation avec GITBOOK. Cela m'a permis d'obtenir une explication claire de mes différentes requêtes avec une belle interface :

The screenshot shows the GitBook API documentation interface. On the left is a sidebar with a user profile 'mlva' and a search icon. The sidebar menu includes 'Accueil', 'TABLEAU DE BORD DE L'ADMINISTRATION', 'Actualités' (with a dropdown arrow), and 'Modification d'un article' (highlighted in blue). Below 'Actualités' are links for 'Liste des actualités', 'Ajout d'une actualité', 'Suppression d'une actualité', 'Ajout d'une catégorie', 'Modification d'une catégorie', and 'Suppression d'une catégorie'. At the bottom of the sidebar is 'Untitled'. The main content area is titled 'Modification d'un article'. It features a 'GET Modifier un article' header with a blue 'GET' badge. Below this is the endpoint URL 'https://eb-creations.fr/admin/actualite/:id'. A note states 'This endpoint allows you to get free cakes.' Below the note are two tabs: 'Request' (highlighted with a red box) and 'Response'. Under the 'Request' tab, there are two sections: 'Path Parameters' and 'Body Parameters'. 'Path Parameters' includes a table with one row: 'id' (REQUIRED) with type 'number' and description 'ID de l'article'. 'Body Parameters' includes a table with six rows: 'description' (REQUIRED, string, 'contenu de la description pré-remplie de l'article'), 'auteur' (REQUIRED, string, 'nom de l'auteur pré-rempli de l'article'), 'ville' (REQUIRED, string, 'nom de la ville pré-remplie de l'article'), 'catégorie' (REQUIRED, string, 'nom de la catégorie pré-remplie de l'article'), 'image' (REQUIRED, string, 'image pré-remplie de l'article'), and 'titre' (REQUIRED, string, 'nom du titre pré-rempli de l'article').

Grâce à cet outil, chaque méthode est détaillée, à la fois pour les « requests », et pour les « responses » avec le statut des différentes requêtes.



PUT Modifier un article

<https://eb-creations.fr/admin/actualite/:id>

Modification de l'article

Reques

Response

● 200: OK

l'article a bien été modifié

```
1 {  
2   "status": "success",  
3   "result": {  
4     ... // données de l'article  
5   }  
6 }  
7 }
```

● 400: Bad Request

```
1 {  
2   "status": "error",  
3   "message": "Le titre est requis",  
4   "message": "L'image est requise",  
5   "message": "La catégorie est requise",  
6   "message": "La ville est requise",  
7   "message": "L'auteur est requis",  
8   "message": "La description de l'article est requise"  
9 }
```

La rédaction de toute la documentation sera longue et prendra tout son sens lorsque je serai amené à travailler en équipe sur le projet.

CONCLUSION ET PERSPECTIVES

Ce projet a véritablement commencé en mars 2019 avec le recueil des besoins de la cliente, ma Directrice. Cette aventure fut passionnante puisqu'elle m'a permis de découvrir, même superficiellement, la majorité des postes qui opèrent autour d'un projet de création d'un site Web. En effet, j'ai occupé bien plus que le poste de développeur, puisque je me suis tout d'abord mué dans un rôle de chef de projet, puis de designer WEB, pour ensuite être développeur front-end et back-end avant de découvrir finalement les rôles de devs ops et data analyst.

Ce fut passionnant, mais cela a rendu le projet d'autant plus complexe, car le langage n'est pas le même en fonction de son poste. Lorsque je devais expliquer l'avancée du projet à la cliente, je ne le verbalisais pas de la même manière que lors de mes échanges avec mon formateur. Cette gymnastique n'est pas simple, mais elle m'a plu.

Il n'est pas non plus aisé lors une reconversion professionnelle de démarrer par un projet aussi dense, car il n'est pas simple de construire en apprenant. Si je devais reconstruire le site aujourd'hui, je ne l'organiserai pas de la même manière et j'utiliserai sûrement des technologies différentes.

C'est d'ailleurs l'une des prochaines étapes de ce beau projet. D'ici la fin de l'année 2019, la migration du nom de domaine et de l'hébergement sera effective avec quelques fonctionnalités supplémentaires (ajout/suppression des salariés, modifications des coordonnées des différents sites, etc.). Mais courant 2020, je compte développer une nouvelle fois le site en utilisant sûrement le framework **REACT**, afin d'obtenir une application très rapide, et surtout qui puisse se déployer en **Progressive Web App**, ce qui serait encore mieux adapté aux jeunes.

Parallèlement il faut que je sois en mesure de déployer un site qui puisse être utilisé par d'autres Missions Locales. En effet, le site a déjà été présenté à d'autres Directions de Missions Locales en Vendée, et elles seraient intéressées pour acquérir le même outil.

Cela conclut d'une belle manière tous les efforts fournis et le temps consacré à ce projet.