Lycée Godefroy de Bouillon

Rapport de stage fin d'année BTS 2e année

Dorian GIRAUD

Stage effectué du 5 janvier au 13 février 2026

Tuteur de stage : M. David RIGAUDIE - PDG

Superviseur académique : M. Bastien DUFOUR

Formation : BTS Services Informatiques aux Organisations

Établissement : Lycée Godefroy de Bouillon

Entreprise d'accueil : ITarverne, 3 Route de Moulet les goulots à Volvic 63530

Lieu du stage : 38 Avenue de Clermont à Sayat 65350

SOMMAIRE

Introduction

Si le **cursus académique** permet d'aborder des **grands principes** de la programmation ainsi que des technologies. Il est important d'observer le monde professionnel pour comprendre la réalité du terrain.

Pour mon stage de 2e année de BTS Services Informatiques aux Organisations, j'ai eu la chance d'être accueilli chez **Itarverne** domiciliée au 3 Route de Moulet les goulots à Volvic 63530. Cette immersion a duré 6 semaines, du 5 janvier au 13 février 2026. Elle m'a permis de me familiariser au React ainsi qu'à l'utilisation du Python dans le web dans un environnement Docker, technologie avec laquelle la pratique me manquait.

Mon maître de stage, Monsieur David Rigaudie, **m'a formé** en insistant sur les bonnes pratiques à adopter. Les différentes tâches que j'ai effectuées étaient sur son Blog cela m'a permis de voir et de travailler sur des technologies que je ne connaissais pas et de m'intéresser aux structures des anciens standards.

Ainsi, comment ce stage m'a permis d'explorer et de consolider mes acquis ?

Après avoir présenté l'entreprise et mon environnement de travail, je dévoilerai les différentes tâches accomplies ainsi que les problématiques que j'ai dû faire face avant de conclure.

PARTIE 1 : Entreprise et mon environnement de travail

I - Entreprise

1) Entité

Itarverne est une TPE fondée en 2017 sous le statut de SAS, société par actions simplifiée, par **David Rigaudie**. Les locaux de cette entreprise du web sont situés dans un **espace coworking** nommé « Espace Comodo » au 38 Avenue de Clermont 65350 Sayat tandis que le siège social est au 3 Routes de Moulet les Goulots, 63530 Volvic.

2) Histoire

Après 10 ans dans le privé avec l'ambition de devenir entrepreneur, David Rigaudie s'est lancé dans la création de son entreprise. Pendant ses huit ans, l'entreprise a vécu **trois phases**. La première a duré 3 ans, la société répondait à des **missions d'entreprise** en tant que consultant. Puis, pendant 3 ans il a développé et maintenu une **application**. À ce moment, la société était en phase de **recrutement**. Pour finir, depuis 2 ans, l'entreprise revient sur ses pas en acceptant des **missions diverses de longue durée** dans des entreprises, souvent parisiennes.

3) Activités et marché

Les activités actuelles de l'entreprise sont diverses. Elle se regroupe ses missions dans différents pôles.

Le premier est le **conseil** et la **formation**. Elle propose une analyse sur l'ensemble du projet en incitant sur l'évolutivité, la cybersécurité et les bonnes pratiques. Il est étonnant d'apprendre les aberrations techniques qui existent dans certaines entreprises de grosses tailles ou de secteurs critiques. La société propose notamment des formations techniques tournées vers le web : Python, Js, CSS, etc.

Le second est le statut de **consultant externe**. Pendant le stage, le maître de stage était en mission dans une équipe de développement dans une banque parisienne.

II - Environnement de travail

1) Cadre organisationnel

Les semaines de stage étaient découpées entre période en **présentiel** et période en **distanciel**. Généralement, j'étais dans les bureaux le lundi, mardi et le vendredi. Et toutes mes journées commençaient à 8h30. Il y avait une pause entre 12h et 13h pour finir à 16h30. Mon bureau était en face de mon maître de stage. J'avais à disposition un **double écran** et une **souris verticale** ainsi que les clés de l'entreprise pour avoir la possibilité de travailler dans les locaux pendant l'absence du maître de stage.

2) Méthode de travail

La méthode de travail de l'entreprise était détaillée. Elle avait un **Gitlab interne** qui intégrait le code et les tâches à effectuer. J'ai eu accès au Slack de l'entreprise, un réseau social similaire à WhatsApp pour professionnel, un canal qui servait à communiquer notamment pendant les périodes à distanciel.

À la fin **de chaque journée**, **je notais les tâches** que j'avais réussi à réaliser, les tâches en cours et enfin les points de blocage ou les attentes d'action de quelqu'un. Chaque tiret devait être précédé d'un **rond vert, jaune ou bien rouge**. Pour chaque élément à réaliser, David Rigaudie ajoutait une « issue » donc une tâche qu'il m'assignait sur Gitlab. Au total, il y en aura eu une **vingtaine** qu'il me rajoutait petit à petit.

Nous mettions différents labels sur ces issues. Pour définir la **priorité** « P1 », « P2 » et « P3 », puis « TO_REVIEW » lorsque je demandais une consultation de Monsieur Rigaudie et enfin « TO_DEPLOY » pour la mise en production. Le Gitlab contenait deux branches : Master et Dev. Lors du développement d'une fonctionnalité, je créais une branche à partir de dev avec un nom parlant comme feat/color-syntax-gist-block. J'avais en moyenne 3 ou 4 MR [Merge Request], des demandes pour faire valider le nouveau code, en simultanées.

Avant de fusionner les branches, il fallait répondre à **7 prérequis** :

- Avoir **mis à jour** sa branche
- Avoir résolu les **conflits**
- Avoir lancé un **linter** (programme qui lit et signale quand une mauvaise pratique est repérée et qui reforme correctement le code)
- Avoir supprimé les **affichages consoles**
- Avoir réalisés des **tests**
- Avoir écrit la **documentation** du code s'il y a le besoin
- Avoir **fait lire le code** par son supérieur (ici seul Monsieur Rigaudie pouvait fusionner les branches)

Enfin, les « commits », les points de sauvegarde, devaient avoir un message qui commençait par un mot-clé puis « : » comme « fix : error 500 Api Get /summary », c'était une **convention** initiée par Angular.

https://www.conventionalcommits.org/fr/v1.0.0/

PARTIE 2 : Les missions

Les missions que j'ai effectuées étaient sur le **blog de l'entreprise** rigaudie.fr, un site qui sert aussi bien pour David Rigaudie que l'entreprise elle-même dû à sa position de freelance. Sur ce blog, il écrit des **articles techniques** sur la programmation. Il existe plein de catégories, elles sont principalement nommées par le nom d'un langage. Le blog est disponible grâce quatre serveurs sur **Docker** :

- Le serveur front en React avec le framework Gastby, un serveur optimisé pour le SSG [Static Site Generator], le préchargement des pages web.
- Le serveur d'édition et d'API sur Wagtail, une surcouche de Django, un Framework Python
- La base de données SQL Postgree
- Le serveur Nginx où on utilisait son reverse proxy. Un reverse proxy est un serveur de façade qui redirige les flux vers les serveurs souhaités.

I – Semaine n°1

Le premier jour a servi à **l'installation** de Docker sur ma machine, l'installation du projet, l'explication des serveurs, la création des comptes Gitlab et Slack ainsi que **la prise en main du projet** en suivant le README. J'ai aussi reçu trois tâches à faire pour me familiariser avec le projet. Je devais :

- Ajouter une balise canonique dans la page article.
- Uniformiser la taille des images dans la liste déroulante des catégories.
- Mettre responsive la page d'accueil et supprimer un décalage lors de l'ouverture de la barre de navigation au format téléphone

Étant donné que pour valider les MR il fallait passer un **linter**, ma première mission fût de **le configurer**. Lors de la première journée, j'ai terminé cette configuration puis j'ai créé la balise HTML canonique. La **balise canonique** est une balise de métadonnées dans le Head de la page web utilisée pour le **SEO**. Elle s'inscrit de cette manière : « <meta rel='url-

de-la-page' ». J'ai pour cela utilisé le composant « **Helmet** » qui permettait d'ajouter des balises dans la section « **header** » de la page. Sans ce dispositif, à chaque fois que l'on revenait sur la page web avec un retour en arrière du navigateur, **la balise se dupliquait**.

Le lendemain, **les serveurs ne se sont pas redémarrés**. Cela m'a pris, mercredi, jeudi et vendredi de la première semaine pour réussir à les **relancer** car je me suis confronté à des problèmes lors des installations de la librairie **Eslint**, l'analyseur de code. Malgré les suppressions des containers, cela ne suffisait pas à résoudre le problème. J'ai finalement résolu ma problématique en supprimant, les containers, les images ainsi que le volume du Docker. C'était ce dernier que j'avais oublié de **supprimer avant de reconstruire le projet** avec la commande « make build » inscrit dans le fichier **Makefile** à la racine du projet, un fichier qui agglomère des suites de commande en une seule. À ce moment, j'étais assez **déçu** de ma performance car en une semaine je n'avais finalement fait qu'une concaténation simpliste et la configuration du linter. Mon maître de stage m'a **rassuré** en m'indiquant que pour tous ses stagiaires ou alternants ont fait face à ce genre d'impasse quand il travaillait sur Docker avec Windows.

II – Semaine n°2

1) Tâche d'uniformiser les images

Pour la tâche d'uniformiser les images, il n'y a eu **aucune difficulté**. J'ai d'ailleurs eu le plaisir d'utiliser pour la première fois un fichier en « .scss », je ne connaissais que de nom alors la pratique était **intéressante**. La dernière tâche en revanche fût problématique, la barre de navigation sous mobile (« sidebar »).

Habituellement, la barre de navigation sur mobile est une surcouche qui passe par-dessus la page. Ici, **la barre latérale élargissait la taille du document**. Le tier de la page sur mobile était pris par la barre mais il était possible de scroll à l'horizontal pour réafficher la page du blog. Ainsi, on se retrouvait avec **document faisait 130 vw au lieu de 100vw unités de large**. C'était un vœu de Monsieur Rigaudie et l'avait développé comme cela. Cependant, lorsque l'on fermait le menu burger en réappuyant sur le bouton qui l'avait fait apparaître, **une barre vide de 30 vw restait**, cette fois à droite de l'écran. Cet écart était rerempli quand le menu burger était présent, mais une fois disparût, la marque de son passage **subsistait**. Il m'a donc fallu redonner aux éléments « body » et « html » le 100vw de large lors du clic du bouton de fermeture de la barre latérale pour **appliquer la correction**. Pour terminer cette issue, j'ai réduit la barre de recherche à l'intérieur de la barre latérale pour éviter qu'elle soit tronquée. Cependant, il existait toujours des **problématiques** sur la barre latérale que j'ai tenté de résoudre à la **6e semaine**.

2) Balise méta dupliquée

Le site web utilise des cookies qui servent à établir des bases statistiques sur les comportements des utilisateurs comme indiqué sur le formulaire des cookies. À la validation, des balises **scripts Google Analytics** entre dans le DOM (l'arbre de la page Internet) seulement, à chaque visite sur la page d'accueil du site, les deux balises se rajoutés de nouveau. Pour empêcher cela, j'ai écrit une condition qui empêche la création si les balises existent avant de les ajouter.

3) Formulaire des cookies

En parlant du formulaire de cookie, j'ai **reformaté la bannière**. J'ai entre autres repris les couleurs de Bootstrap pour le bouton d'acceptation et de déclinement des cookies de manière responsive. Cette petite activité m'a permis d'appendre plusieurs choses :

- BEM, [Block Element Modifier]

Après un retour de Monsieur David Rigaudie, il m'a présenté la **convention BEM**. Elle s'insère dans le **choix des noms de classe**. Le [**Block**] est la **localisation de l'élément**, c'est donc la classe du parent. Si le nom de la classe est une suite de mot, chaque mot est séparé par un tiret « - ». Entre le nom de la classe parent et le nom de l'**élément** on met deux tirets du bas « _ _». Le [**Modifier**] est un groupe de mot que l'on met à la fin pour désigner si cette classe est utilisée dans des **cas spécifiques** : mobile, js, focus, restricted, overlap ou pour préciser l'usage de la classe de manière générale ex : h3—home, les h3 dans la page home. Ainsi, entre le nom de l'élément et le [Modifier] on écrit deux tirets « -- ».

- Il faut **bannir** les propriétés z-index, les indicateurs !important et les id dans le css et faire attention **Specificity**.

Pour définir quelle ligne de css sera assignée à un document, chaque ligne est soumise à une **note**. Si deux lignes sont contradictoires, ce sera propriété avec la note la plus élevée qui sera choisie. Comprendre le barème de notation permet d'esquiver les !important.

- Pour finir j'ai appris que toutes les unités de longueurs (px, rem, em, %) étaient compatibles avec **toutes les propriétés qui utilisaient des unités de longueur** comme font-size, par exemple.

III – Semaine n°3

1) Coloration des blocs syntaxiques

Le blog traite des **sujets techniques** sur le **développement**, il est courant d'avoir des **blocs de code**. Il n'y avait pas de couleurs syntaxiques, tous les mots étaient blancs sur fond foncé. Monsieur David Rigaudie m'a demandé de résoudre ce souci esthétique qui complique la lisibilité du code en utilisant une librairie nommée **Pygment**. Pygment est une **librairie Python** qui vise à ajouter des balises span entre les mots pour définir des couleurs différentes. Ces couleurs sont décidés en fonction du langage inscrit dans le paramètre de la fonction et des couleurs des classes html choisies dans le site. Pygment est côté back-end au côté du serveur d'édition des pages.

Le serveur back-end est sur **Wagtail**, une **surcouche de Django** qui est spécialisé pour les blogs. Il contient une page d'édition des articles. À l'intérieur, on peut choisir **d'ajouter des formulaires**. Les résultats seront stockés dans la base de données dans la table api_blogpage à la colonne body avec un type jsonb. J'ai donc modifié le formulaire **GistBlock** qui était en lien avec les blocs de code pour y ajouter une liste défilante de langage allant de PowerShell à YAML. Je notais le résultat de ce formulaire dans la clé « language » et le résultat du formulaire initial dans « code ». Grâce à ses deux informations, j'ai pu retourner une chaîne de caractère comprenant le contenu du code segmenté par les balises span.

Du côté du front, j'ai ajouté dans la balise parent un dangerouslySetInnerHTML pour que les spans soit considérés comme des balises et non comme un mot standard. Ensuite, pour ajouter les numéros des lignes, un paramètre dans Pygment générait la chaine de caractère avec des balises tr et td pour former un tableau afin de séparer le code et les numéros pour faciliter le **bouton de copier-coller** du code et avoir tous les numéros de ligne correctement alignés.

2) Intégration de la coloration sur le serveur de production

Avant la mise en place de la fonctionnalité, la valeur du bloc de code était dans la base de données dans la chaine de caractère associée à la clé « value » dans chaque objet de type « gist ». Maintenant, « value » était un dictionnaire qui contenait une clé « language » et une clé « code ». Ainsi, lorsque Pygment tentait de rendre la chaine de caractère, il piochait des informations qui n'existait pas et retournait une **chaine de caractère vide**. Pour régler ce souci, il fallait mettre la valeur stockée dans « value » dans la valeur de « code » et ajouter la clé « language ».

Ainsi, j'ai créé un script qui se connectait à la base de données, qui récupérait le body des articles pour y **ajouter les deux clés** dans chaque clé « value » des json. Pour le langage par défaut dans ce script nous avons choisi Python car il représentait la majorité des blocs de code. Ce script a été réalisé en Python et utilisait la librairie psycopg2 pour la connexion avec la base de données Postgree. Ce script a été ajouté au dossier « scripts » à la racine du projet. J'ai ici appris la différence de syntaxe entre le dictionnaire

Python et le json. Python utilise des guillemets simples ‘ ’ tandis que le json se sert des guillemets doubles « ».

IV – Semaine n°4

1) Prévisualisation

À l’intérieur du serveur Wagtail, qui sert à la fois **de serveur API et de serveur d’édition des pages**, il y a un module intégré de **prévisualisation**. La page envoie attaque route API avec la méthode GET à son propre serveur. **Deux paramètres** sont récupérés : « content_type » et « token ». « Content_Type » sert à définir quel est le type de page qui est en train d’être modifier. Ici, la valeur était « BlogPage » ce qui correspondait à une page d’un article. Le second paramètre contenant quant à lui tous les champs complétés à l’intérieur des formulaires qui étaient sur la page (et non sur la base de données). Ainsi, cela servait à appeler la page en cours en donnant en paramètre les informations à afficher.

Seulement, Wagtail a deux modes, un mode **Monolithique** (Traditionnel) et un mode « **Headless** » (API). Le mode « Headless » où nous étions est comparable au serveur Symfony quand on n’initialise pas le serveur avec le paramètre « --webapp ». Ici, la prévisualisation ne fonctionne pas nativement car le serveur s’appelle soi-même mais **n’est pas en mesure d’afficher** le résultat. Ainsi, ma mission était de faire en sorte qu’il appelle le bon serveur. Pour cela, j’ai fait hériter « BlogPage » qui affichait la page d’édition des pages d’article un objet « HeadlessMixin ». **J’ai ajouté une route API** qui renvoie les données « content_type » et le « token », j’ai créé un routeur qui renvoie les informations au template de la bonne page en fonction de « content_type ». Dans le template, je vérifiais s’il recevait des valeurs de la prévisualisation, si oui alors il n’appelait pas la route standard. La route standard qui servait à récupérer les informations de la base de données.

Dans le « token », **il n’y avait pas le slug de la catégorie** qui était nécessaire pour l’affichage de la page, j’ai donc fait **deux requêtes api**, l’un pour récupérer les données de la **prévisualisation** et l’autre pour récupérer le **slug de la catégorie** en comparant avec l’identifiant de la catégorie inscrit sur le json renvoyé par le premier.

V – Semaine n°5

1) Récupérer le slug de la catégorie

Pour éviter le second appel de l'API, il fallait que le premier appel revoie le slug de la catégorie. Dans l'objet de BlogPage, il était noté la valeur du slug, j'ai dû faire une **sérialisation du champ catégorie** pour ajouter à l'intérieur le nouveau champ slug. Je n'ai pas su pourquoi par défaut il n'était pas renvoyé.

2) Extrait de code inline

David Rigaudie a l'habitude d'écrire les articles en ajoutant une ligne ou quelques mots dans un bloc de code. Il fallait que je mette la **coloration syntaxique** sur ces éléments également. Pour cela, j'ai utilisé un regex pour remplacer les éléments entre les balises « code » par le résultat de la fonction de Pygment. J'en ai également profité pour mettre cette fonctionnalité également sur les **formulaires RawHTML**. Les formulaires RawHTML sont, de la même manière que les blocs de code, insérés en DangerousSetInnerHTML pour ajouter des balises alors j'ai fait un regex dans ces blocs qui vérifie les itérations des balises code avec un attribut « language » pour appliquer les changements.

3) Retour à la ligne des blocs de code

Tous les résultats de la fonction de Pygment débordaient de la page web. Ainsi, dans les blocs de code « classique », j'ai ajouté **un scroll horizontal** et dans les petits bouts de code j'appliquais un **retour à la ligne**.

4) Petites retouches de CSS

Les différentes cartes dans les pages avaient des tailles différentes, disproportionnaient, pas droites, les titres dépassaient de la page. J'ai tout mis au propre jusqu'au début de la 6e semaine.

VI – Semaine n°6

1) Correction Orthographique

Dans la page d'édition, David Rigaudie voulait que je fasse une **correction de texte automatique**. J'ai donc fait des appels API de TextGears, un site qui propose ce service. Je suis arrivé à un début de fonctionnalité concluant mais nous avons décidé d'utiliser l'extension Chrome Grammarly, elle a donc été avortée.

2) Retravail sur la sidebar

La barre latérale faisait une hauteur de 100vh, il fallait que je fasse descendre la barre jusqu'en bas de la page de manière dynamique et idéalement faire descendre les boutons. J'ai essayé avec une propriété sticky top mais la barre de navigation passait sous le contenu de la page. J'ai repéré qu'il y avait deux balises « header » impriquées, j'en ai donc supprimé une. Avec une flexbox, j'ai pu faire descendre la barre jusqu'en bas de la page. **L'animation d'ouverture et de fermeture** de la barre latérale a était produite par une transition d'une marge gauche de -30vw à 0. C'est-à-dire que par défaut, la page glisse sur la barre et revient à l'état initial quand on clique sur le bouton pour afficher la barre de navigation.

La dernière journée a servi à finaliser des petits éléments sans importance.

Conclusion

Pour conclure, ce stage m'a permis d'explorer des technologies qui m'étaient totalement inconnu telles que Gatsby et Wagtail. Je ne pense pas réutiliser Wagtail car un framework est censé faire gagner du temps au développeur, ce n'est pas mon expérience. Il m'a fait intégrer des automatismes sur le css, avec BEM notamment et ainsi que sur les messages des commits Git et sur Git en général. Grâce à Monsieur David Rigaudie j'ai consolidé mes acquis sur Python : le mot-clé yield, les conventions sur les variables et les noms des fonctions, convention sur le tuple, etc. J'ai également développé mon vocabulaire : SSG, SSR, Linter. Je prendrai soin d'appliquer les éléments appris à ce stage dans mes autres projets, aussi bien scolaire, professionnel que personnel.

Cordialement,

Giraud Dorian

Compétences travaillées selon le référentiel du BTS SIO :

- B1.5.2 : Déploiement d'un service
- B1.2.2 : Installation et configuration des éléments nécessaires pour assurer la continuité des services
- B1.2.3 : Rédaction ou mise à jour de la documentation technique et utilisateur d'une solution d'infrastructure
- B2.1.4 : Exploitation des ressources du cadre applicatif (framework)
- B2.1.5 : Identification, développement, utilisation ou adaptation de composants logiciels
- B2.1.6 : Exploitation des technologies Web et mobile pour mettre en œuvre les échanges entre applications
- B2.1.7 : Utilisation de composants d'accès aux données
- B2.1.9 : Réalisation des tests nécessaires à la validation ou à la mise en production d'éléments adaptés ou développés
- B2.1.11 : Exploitation des fonctionnalités d'un environnement de développement et de tests
- B2.2.3 : Analyse et correction d'un dysfonctionnement
- B2.2.4 : Mise à jour de documentations technique et d'utilisation d'une solution applicative
- B2.2.5 : Élaboration et réalisation de tests des éléments mis à jour
- B2.3.1 : Exploitation de données à l'aide d'un langage de requêtes
- B2.3.3 : Conception ou adaptation d'une base de données
- B3.1.3 : Application de la réglementation en matière de collecte, de traitement et de conservation des données à caractère personnel
- B3.5.1 : Vérification des éléments contribuant à la qualité d'un développement informatique