JUNIA ISEN

Étude et Interopérabilité des CMS PHP et de la Technologie AJAX

Elèves : Ilyes ACHAQ, Louis CHARPENTIER, Mohamed-Walid NASSIR ELHAK, Orel AGUIAR, Othniel ARYEE, Leandra CHEKAM NJOKO, Kawtar BELKHETAB

Tuteur de projet : Vincent LEFEVRE

Année 2024 - 2025

Contents

Part I

Etude Bibliographique

1 Introduction

Au sein du contexte actuel de développement du web, la mise en place de systèmes d'information performants et évolutifs doit être assurée par l'intégration des technologies les plus modernes. Les systèmes de gestion de contenu (CMS : Content Management System) tels que WordPress, Joomla ou WebSiteBaker, tous très largement utilisés pour créer et gérer le contenu web (site ou blog), sont une solution efficace pour la gestion des informations. Mais leur architecture traditionnelle, souvent très centrée sur le binôme back-end et front-end, amenuise leur flexibilité face aux exigences croissantes de la personnalisation des interfaces utilisateur et de leur dynamisme. [1]

D'autre part, l'avènement des technologies telles qu'AJAX (Asynchronous JavaScript and XML) [2] [3] ou des applications monopages (SPA : Single Page Application) [4] a révolutionné les interactions entre systèmes web et utilisateurs. En permettant d'organiser des échanges de données avec le serveur sans avoir à recharger complètement la page, la mise en œuvre de ces technologies favorise une séparation plus marquée des préoccupations entre front end et back end, correspondant mieux aux besoins des interfaces modernes. [5]

Le projet a donc pour but de croiser ces deux paradigmes technologiques sur comment utiliser les CMS comme un back-end centralisateur de données tout en intégrant AJAX et les concepts de SPA pour concevoir des interfaces modernes de type front-end, d'une part, interopérables, d'autre part. Nous tenterons, à partir d'un état de l'art bibliographique et de Proofs of Concept (POC), d'étudier comment opérer cette combinaison pour améliorer la flexibilité, la performance ou la sécurité des systèmes d'information.

Ce projet se consacre à démontrer à la fois la faisabilité technique de cette approche et à donner des éléments de recommandation pour bâtir des modèles de conception universels assurant une interopérabilité maximale entre différents CMS, le tout respectant les contraintes de performance, de sécurité et de compatibilité. À plus long terme ce projet s'inscrit dans une volonté de moderniser le recours aux CMS pour le développement web, en les amenant à s'intégrer à part entière aux technologies front-end de nouvelle génération.

1.1 Etat de l'Art

1.1.1 Qu'est ce qu'un CMS ?

Un *Content Management System* (CMS) [1] est une application logicielle qui facilite la création, la gestion et la publication de contenu sans difficulté, souvent utilisée pour les sites Web type blog ou encore E-commerce. Ces systèmes sont connus pour leur capacité à maintenir le contenu facilement ordonné à l'aide de leur interface utilisateur et ils sont destinés à être utilisés par des non-professionnels de l'informatique. Dans la suite de ce document, nous nous pencherons uniquement sur les liaisons AJAX avec les CMS : Joomla, WordPress et WebSiteBaker, ces CMS étant les plus courants du marchés et les plus utilisés (pour WebSiteBaker, le choix a été fait en raison de sa particularité à être un CMS concu pour des projets courts) [6].

Figure 1: Utilisation d'un CMS

Joomla [7] se caractérise par une structure hiérarchique bien définie des contenus, organisée en catégories et sous-catégories. Depuis la version 4, le CMS intègre nativement une API REST, permettant de l'utiliser comme un

système "headless", où les données peuvent alimenter un front-end moderne basé sur des frameworks comme React ou Vue.js. Apprécié notamment pour sa gestion multilingue intégrée et ses mécanismes de sécurité avancés, tels que l'authentification à deux facteurs et une gestion précise des permissions via les *Access Control Lists* (ACL), Joomla peut malheureusement sembler assez complexe à mettre en place pour les utilisateurs non-initiés, surtout à cause de son architecture MVC (Model-View-Controller) et de la configuration des ACLs [8]. De plus, bien que sa bibliothèque d'extensions soit riche, elle est moins fournie que celle de WordPress.

Concernant WordPress [9], sa facilité d'utilisation et l'usage du système ont déjà été mentionnés. De nombreuses parties tierces prennent en charge le CMS, qui possède un écosystème riche avec plus de 60 000 plugins et des milliers de thèmes. L'éditeur Gutenberg est basé sur React.js. De ce fait, il est possible de créer des pages visuelles à l'aide de blocs glissables dans une interface intuitive. De plus, malgré le manque d'interopérabilité, WordPress propose une API REST. En outre, il peut être facilement étendu via des plugins, dont un composant pour faciliter l'obtention de données, WPGraphQL, permettant une requête plus conviviale et flexible via le réseau.

Enfin, WebSiteBaker [10] est un CMS assez léger, prévu pour des projets de petite envergure devant être réalisés rapidement. Sa structure de fichiers simpliste et son interface intuitive rendent ce CMS adapté pour les utilisateurs non techniques ou pour des sites simples. Toutefois, WebSiteBaker ne permet pas de faire des intégrations complexes et son utilisation de l'AJAX nécessite des scripts personnalisés, ce qui est insuffisant pour un projet requérant une interactivité dynamique.

1.1.2 Qu'est-ce que la technologie AJAX et sa liaison avec les SPA ?

AJAX [2], acronyme de *Asynchronous JavaScript and XML*, est une technologie qui permet de mettre à jour dynamiquement des contenus web sans nécessiter le rechargement complet de la page. Il s'agit d'un ensemble de technologies combinées, comprenant JavaScript pour manipuler le *DOM* (Document Object Model), des requêtes HTTP asynchrones pour interagir avec le serveur, et des formats d'échange de données comme JSON. Contrairement aux approches traditionnelles où chaque action utilisateur provoquait un rechargement de la page, AJAX permet de n'actualiser que les parties nécessaires, offrant une expérience utilisateur plus rapide et fluide.

Le fonctionnement de cette technologie repose sur plusieurs étapes : une interaction utilisateur déclenche une requête `XMLHttpRequest` ou `Fetch API` [11], envoyée de manière asynchrone au serveur. Le serveur traite la requête et renvoie une réponse, souvent au format JSON, qui est ensuite utilisée pour mettre à jour le contenu de la page via le DOM, sans interrompre le contenu que regarde l'utilisateur.

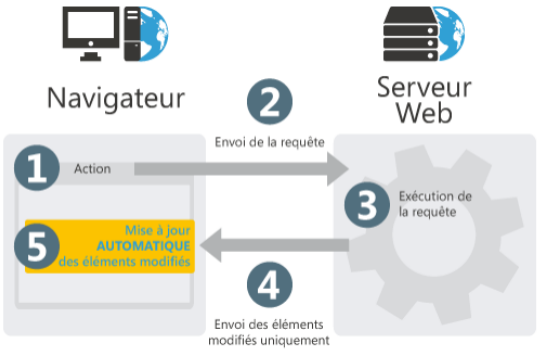


Figure 2: Interaction AJAX

L'intégration d'AJAX dans les CMS, comme WordPress ou Joomla, permet donc d'améliorer les fonctionnalités comme le chargement dynamique de contenus, la validation de formulaires en temps réel, mais aussi la gestion de notifications.

Les *Single Page Applications* (SPA) [12] représentent une évolution naturelle dans l'utilisation d'AJAX. Contrairement aux sites web classiques qui nécessitent le chargement de nouvelles pages a chaque fois qu'on en ouvre une, une SPA charge toutes les ressources nécessaires lors de l'accès initial. Ensuite, AJAX est utilisé pour récupérer les données à afficher, tandis que des frameworks comme React.js, Vue.js ou Angular manipulent dynamiquement le DOM pour

actualiser les sections pertinentes. l'utilisation des SPA en parallèle d'AJAX réduit les interactions réseau et donc améliore la fluidité de l'expérience utilisateur en réduisant le temps de chargement de chaque élément.

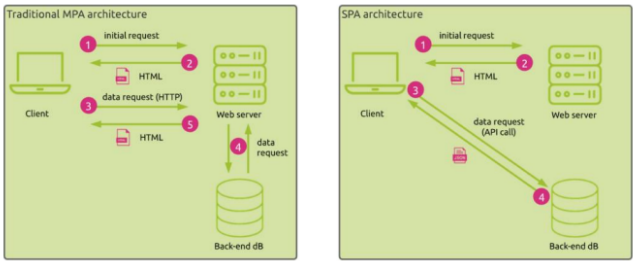


Figure 3: Interaction SPA

Malgré tout, cette utilisation comporte des défis importants. Sur le plan technique, il est nécessaire de gérer des problématiques de compatibilité entre navigateurs, de sécurité (par exemple contre les attaques XSS[13] ou CSRF[14]), et d'accessibilité. De plus, le chargement initial des SPAs peut être plus long que le chargement du CMS sans implementation de SPA, et leur architecture requiert une gestion fine de l'historique de navigation pour conserver une expérience utilisateur intuitive. Enfin, le référencement des contenus dynamiques peut nécessiter des ajustements, comme le rendu côté serveur (SSR). [15]

1.2 Analyse Approfondie des CMS

L'intégration d'AJAX (Par exemple ici sur WordPress [16])dans un CMS permet de créer des interactions dynamiques sans recharger entièrement la page, améliorant ainsi significativement l'expérience utilisateur. Bien que les trois principaux CMS étudiés (WordPress, WebsiteBaker et Joomla) offrent des approches légèrement différentes, le processus général reste similaire :

Étapes Générales

- **Choix de la méthode d'intégration :** Cela inclut l'utilisation directe de JavaScript pour une grande flexibilité, ou de bibliothèques comme jQuery ou Axios pour simplifier le développement. Les plugins dédiés permettent une mise en œuvre rapide pour les fonctionnalités courantes[17].
- **Création d'un point d'entrée PHP :** Le fichier PHP reçoit et traite les requêtes AJAX pour retourner les données nécessaires[18].
- **Envoi de requêtes AJAX :** Ces requêtes permettent une communication asynchrone entre le front-end et le serveur[19].
- **Mise à jour du DOM :** Les données retournées sont utilisées pour mettre à jour dynamiquement l'interface utilisateur[20].

Chaque CMS présente des spécificités dans la manière dont ces étapes sont réalisées, influençant la performance, la sécurité et la flexibilité de l'intégration.

1.3 Synthèse de l'Opérabilité CMS-AJAX

La combinaison d'AJAX avec des CMS comme WordPress, WebsiteBaker et Joomla optimise leur interopérabilité avec des Single Page Applications (SPA). Voici un résumé des avantages et défis de chaque CMS :

Critères	WordPress	WebsiteBaker	Joomla
Flexibilité	Très élevée, grâce à un écosystème riche en plugins et API REST	Moyenne, moins de plugins dédiés et développement personnalisé souvent requis	Élevée, mais nécessite souvent une configuration plus avancée
Communauté	Très large, avec des ressources abondantes	Plus petite, mais active	Grande, moins concentrée sur AJAX que WordPress
Intégration AJAX	Facile grâce à la REST API et jQuery, soutenue par de nombreuses extensions	Nécessite du développement manuel, intégration d'AJAX plus limitée	Facile grâce à MooTools et aux extensions dédiées
Performance	Bonne en général, mais peut être affectée par les nombreux plugins	Optimisée pour des projets de petite à moyenne taille	Optimisable pour les grandes applications avec une configuration soignée
Sécurité	Vigilance nécessaire sur les plugins tiers pour éviter les failles	Moins problématique mais dépend fortement de la configuration	Sécurité robuste, grâce à des fonctionnalités natives comme ACL
Interopérabilité SPA	Idéal pour des SPAs avec React, Vue.js ou Angular grâce à la REST API	Possible, mais moins adapté pour des projets complexes	Bien adapté, mais nécessite une configuration spécifique

Table 1: Comparaison des CMS pour une intégration AJAX et SPA.
[21] [22]

1.4 Exemple d'Intégration AJAX avec Joomla

a) Envoi d'une requête AJAX

Depuis le front-end, une Single Page Application (SPA) peut envoyer des requêtes HTTP via des bibliothèques comme `fetch` ou `Axios`. Voici un exemple simple en JavaScript pour interroger l'API REST de Joomla[23] [24] :

```
fetch('https://monsite/api/index.php?option=com_content&task=getArticles', {
    method: 'GET',
    headers: {
        'Authorization': 'Bearer <token>',
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => console.log(data));
```

Listing 1: Exemple de requête AJAX avec fetch

URL : Il s'agit du point d'accès à l'API REST de Joomla.[23]
Headers : Contiennent souvent un token d'authentification, comme un JWT.[25]
Payload : Utilisé pour envoyer des données supplémentaires avec des requêtes POST.

b) Traitement côté serveur (Joomla)

Côté serveur, Joomla traite la requête via son composant REST (comme `com_api` ou l'API native). Voici les étapes détaillées :

- **Authentification :** Vérification du token JWT ou de la session utilisateur[25].
- **Vérification des droits :** Joomla utilise les ACL (*Access Control List*) pour contrôler les accès. On peut demander à notre système d'information de les intégrer[26].
- **Traitement de la requête :** Extraction des articles ou mise à jour des données demandées[23].

Le serveur renvoie ensuite une réponse en JSON :

```
{
    "status": "success",
    "data": [
        {"id": 1, "title": "Article 1"},
        {"id": 2, "title": "Article 2"}
    ]
}
```

Listing 2: Exemple de réponse JSON

2 Analyse de faisabilité

2.1 Définition des cas d'utilisation réalistes généraux

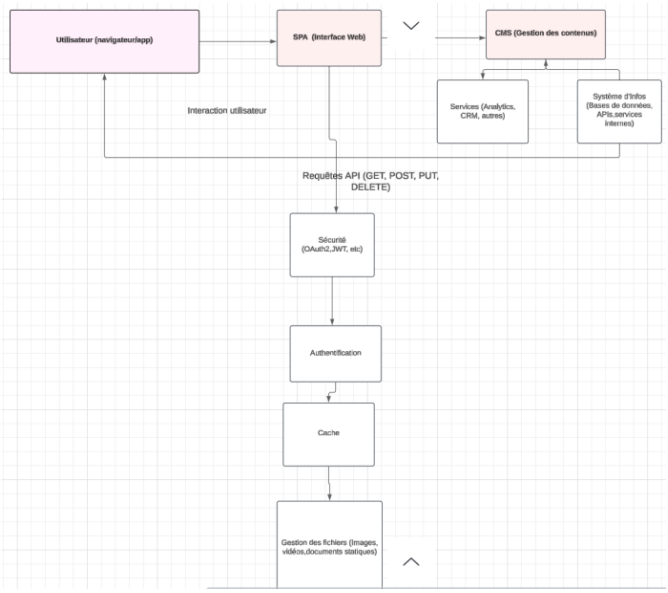


Figure 4: Schéma génral de l'integration AJAX avec des CMS

Le schéma présente une idée générale du fonctionnement d'une application web moderne de type Single Page Application (SPA). Quand un utilisateur se connecte, il saisit ses identifiants dans un formulaire. Ces informations sont envoyées au serveur via une requête **HTTPS** sécurisée[27]. Le serveur vérifie si tout est correct et renvoie un **jeton d'authentification** (comme un *JSON Web Token*, ou JWT)[25]. Ce jeton contient des informations sur l'utilisateur et est stocké côté client (dans le *localStorage*, le *sessionStorage* ou dans des cookies sécurisés)[28]. Ensuite, toutes les requêtes API suivantes incluent ce jeton pour garder la session sécurisée[25].

Côté serveur, il y a un **CMS** (Content Management System) qui permet de gérer facilement les contenus comme des textes, images ou vidéos[29]. Les administrateurs peuvent modifier ces contenus directement depuis une interface sans avoir à coder. Ces données sont accessibles via des **APIs**[30], ce qui permet à l'application de les afficher en temps réel. Pour éviter de trop solliciter le CMS, une couche de **cache** est utilisée pour gagner en performances[31].

Enfin, des services externes (comme des outils d'analyse ou des CRM) et des bases de données complètent le tout[32]. Les fichiers statiques (CSS, JavaScript, images) sont également bien gérés pour que l'application reste fluide, rapide et sécurisée pour l'utilisateur.[33]

2.2 Etude de faisabilité par CMS

2.2.1 WordPress

Le schéma (cliquez ici pour voir le schéma) montre comment intégrer **AJAX** dans WordPress pour une **SPA** connectée à un **Système d'Information (SI)**.

AJAXPress et API REST WordPress : Contrairement au schéma général où les requêtes API sont standardisées, cette intégration se concentre spécifiquement sur WordPress à travers **AJAXPress** et l'**API REST**.

- **AJAXPress** : permet l'envoi de requêtes dynamiques sans nécessiter le rechargement de la page[34].
- **API REST** : favorise un échange structuré de données entre le front-end (SPA) et le back-end de WordPress[35].

Base de Données WordPress : Le système de gestion de contenu WordPress gère directement une base de données dédiée (**BDD WordPress**) pour le stockage et la récupération de contenus (articles, médias, etc.). Les opérations de lecture et d'écriture s'effectuent par le biais de requêtes **AJAX** qui passent par le serveur WordPress[36].

Système d'Information (SI) et Base de Données Externe : Dans ce modèle, le SI fonctionne de manière autonome par rapport à WordPress et possède sa propre base de données. Le serveur WordPress joue le rôle d'intermédiaire pour transmettre les requêtes API du front-end (**SPA**) vers le SI externe :

1. Les requêtes **AJAX** sont émises par la SPA pour interroger le serveur WordPress.
2. Le serveur appelle l'API du SI, reçoit une réponse et peut éventuellement mettre en cache les données.
3. Le SI effectue des opérations de requêtes et d'écritures dans sa propre base de données.

Mise en Cache : Une couche de mise en cache est intégrée pour améliorer les performances en évitant de solliciter systématiquement la base de données WordPress ou le SI externe. Cela permet de réduire les temps de réponse des requêtes API[37].

Transmission des Données JSON/XML : Les données échangées entre les différents éléments sont formatées en **JSON** ou **XML**, ce qui facilite leur interprétation par le front-end SPA pour une mise à jour dynamique du **DOM**[38].

2.2.2 Joomla

Architecture 1 : Ce schéma montre comment **Joomla**, en tant que **back-end** de la SPA, agit comme un **proxy** entre la SPA et le **Système d'Information (SI)**. Joomla gère principalement les pages statiques, tandis que le SI prend en charge les processus métiers dynamiques, tels que la gestion des utilisateurs, des commandes, et d'autres logiques métier complexes. **Joomla** n'interagit pas directement avec la SPA, mais agit uniquement comme un intermédiaire entre la SPA et le SI[39].

Rôle de Joomla en tant que Proxy

Joomla est utilisé ici pour servir les pages statiques (articles, blogs, etc.) et pour traiter les requêtes **AJAX** envoyées par la SPA. Cependant, au lieu d'interroger directement la base de données de la SPA ou d'effectuer des traitements métiers, Joomla agit comme un **proxy**, en transférant les requêtes de la SPA vers le SI, et en renvoyant la réponse du SI à la SPA[39].

Joomla reçoit les requêtes de la SPA, les traite si nécessaire (par exemple, gestion des permissions avec les **ACL**), puis les redirige vers le SI via des appels **API REST** ou d'autres méthodes de communication comme des requêtes HTTP. Garantissant que la SPA n'accède pas directement aux données du SI[40].

[41] [42].

Architecture 2 : Dans cette architecture, la SPA peut interagir directement avec Joomla (CMS) et le Système d'Information (SI) via des API distinctes. Chaque entité (SPA, CMS, SI) peut fonctionner indépendamment[40].

Gestion de l'authentification :

La SPA s'authentifie auprès de Joomla en envoyant une requête à un endpoint dédié (exemple : /api/auth/login). Joomla génère un JWT (JSON Web Token) signé et le retourne à la SPA. Ce JWT est ensuite utilisé pour authentifier les requêtes suivantes.

Joomla envoie le JWT au SI afin qu'il puisse vérifier l'authenticité de la SPA. (La SPA communique son JWT par requete)[43].

2.2.3 WebSiteBaker

Le schéma est retrouvable ici

AJAX et SPA (Application Monopage) L'interface SPA (*Single Page Application*) envoie des requêtes asynchrones AJAX aux points de terminaison personnalisés de WebsiteBaker (par exemple, `ajax_handler.php`) pour récupérer ou mettre à jour des données sans recharger la page. Cela permet une communication en temps réel entre la SPA et le backend de WebsiteBaker[44].

Points de Terminaison AJAX Personnalisés dans WebsiteBaker WebsiteBaker utilise des scripts PHP pour gérer les requêtes AJAX. Les requêtes sont validées à l'aide de **JWT** (*JSON Web Tokens*) ou de tokens **CSRF** pour garantir une communication sécurisée. Le backend détermine si les données proviennent de la base de données interne ou nécessitent un appel API vers le Système d'Information (SI)[45][46].

Interaction avec la Base de Données Interne Pour les contenus internes (pages, articles, médias), le backend :

- Effectue des requêtes sécurisées avec des requêtes paramétrées[47].
- Formate les réponses en **JSON**/**XML** pour les renvoyer à la SPA[48].

Intégration avec les APIs et le Système d'Information (SI) Pour les données externes :

- Le backend effectue un appel API sécurisé vers le SI externe via **HTTPS**[49].
- Le SI interagit avec sa propre base de données et renvoie une réponse.
- Le backend WebsiteBaker formate cette réponse en **JSON**/**XML** pour la SPA.

2.3 Conception d'une architecture technique adaptable selon les différents CMS

Le schéma général précédent (Le schéma est retrouvable ici) représente l'idée générale de la façon dont on peut théoriquement implémenter cette liaison SPA/CMS via AJAX. Cependant selon les CMS utilisés il nous a fallut identifier les technologies qui allaient nous permettre de mettre en place ce système.

Nous avons, suite à nos réalisation par CMS, adopté un schéma généraliste pour couvrir les différents cas de communication entre une SPA et un CMS, en tenant compte des variations de fonctionnalités des CMS. Ce schéma intègre une étape facultative dédiée à un script ou une API customisé lorsque le CMS ne prend pas nativement en charge les requêtes AJAX. Cette approche permet de s'adapter aux CMS modernes disposant d'API REST natives (ex. Joomla) tout en incluant des solutions pour des CMS plus limités comme WebsiteBaker[50].

Approche Choisie

- **CMS avec support AJAX :** Les requêtes AJAX de la SPA sont transmises directement à l'API REST du CMS, en intégrant des mécanismes de sécurité comme les tokens JWT et une gestion fine des droits via les ACL[51].

– **CMS sans support AJAX :** Une API ou un script personnalisé est mis en place pour transformer les requêtes AJAX en requêtes GET/POST, et pour convertir les réponses *HTML/XML* en JSON[50].

Processus de Communication

Étape 1: La SPA envoie une requête AJAX avec un token JWT[52].

Étape 2: Le CMS ou le script personnalisé traite la requête et interroge la base de données ou le système d'information (SI)[53].

Étape 3: Le CMS renvoie une réponse en JSON à la SPA, permettant une mise à jour sécurisée et dynamique.[53]

Avantages

– **Flexibilité :** Compatible avec une large gamme de CMS.

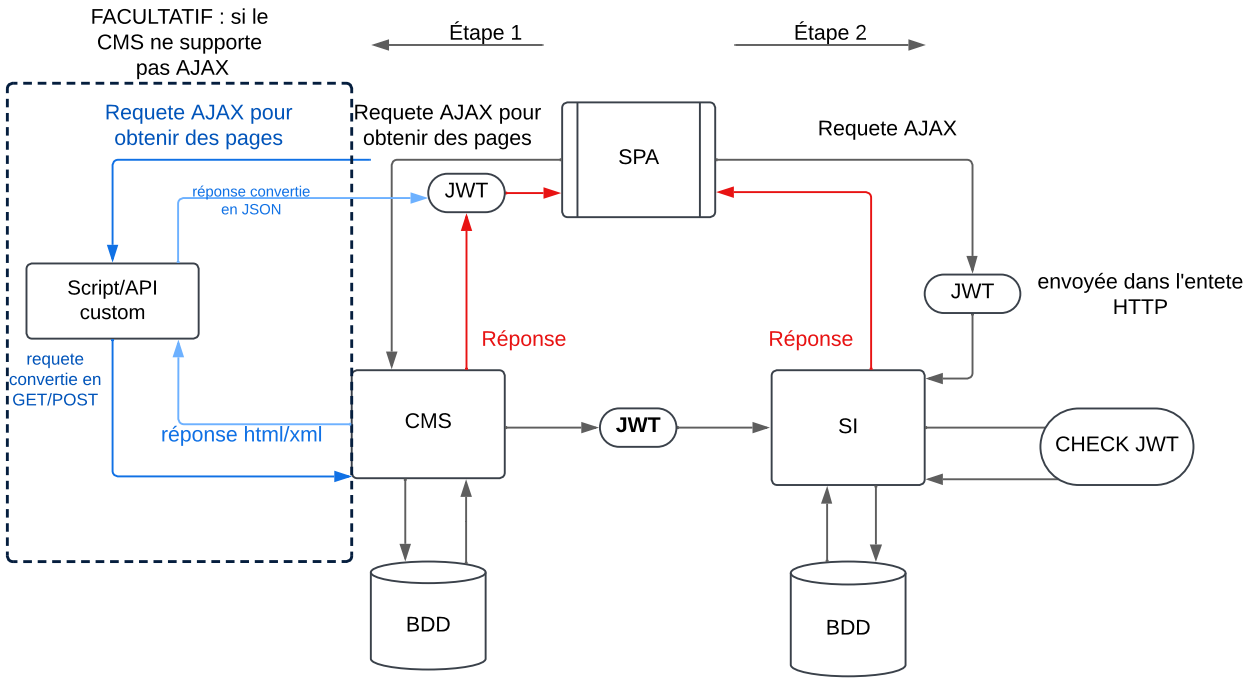
– **Interopérabilité :** Les échanges sont uniformisés via JSON.

– **Sécurité :** Les tokens JWT assurent des échanges protégés.

– **Réutilisabilité :** Une architecture adaptable pour d'autres projets.

Ce mode de traitement garantit une communication efficace entre la SPA et les CMS, tout en s'adaptant aux spécificités techniques des CMS, qu'ils soient modernes ou plus limités.

Le schéma reste applicable à une large gamme de projets, indépendamment du CMS utilisé :



3 Cahier des Charges du Second Semestre

3.1 Thématique Proposée

Pour le prochain semestre, nous nous pencherons sur le cas d'usage d'un site d'hotel de reservation. Les actualités de l'hotel seraient gérées par des billets blogs contenu dans les CMS, le système de réservation, lui, sera géré par le système d'information. L'objectif principal sera de démontrer le fonctionnement de notre architecture présenté ci-dessus

3.2 Objectifs du Semestre

Objectifs

- **Objectif 1 :** Démontrer de façon pratique l'existence d'un système général permettant la décorélation entre CMS, SI et SPA via des liaisons AJAX.
- **Objectif 2 :** Mettre en place un système virtualisant les différents CMS pour prouver les concepts.

3.3 Thématiques Abordées

- **Thématique 1 :** Cybersécurité : sécurisation des tokens de connexion.
- **Thématique 2 :** Développement Web : développement des solutions.
- **Thématique 3 :** Réseaux : virtualisation des CMS.

3.4 Contraintes et Exigences

- **Contraintes techniques :** Utiliser la structure définie dans ce dossier en POC.
- **Contraintes temporelles :** 1 mois de réalisation.
- **Exigences spécifiques :** Les POCs doivent être présentables lors de la soutenance finale.

3.5 Planning Prévisionnel

Cliquez ici pour voir le planning prévisionnel.

3.6 Livrables Attendus

Livrables

- **Livrable 1 :** Présentation des POCs.
- **Livrable 2 :** Code source et documentation technique.

3.7 Critères de Réussite

- Démonstration de l'interopérabilité de chaque système.
- Fonctionnalité des environnements CMS virtualisés et démontrables.
- Respect des bonnes pratiques en matière de sécurité et d'architecture.

References

[1] Wikipedia, "Cms," *Wikipedia*, 2016. [Online]. Available: https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_contenu

[2] ——, "Ajax," *Wikipedia*, 2024. [Online]. Available: https://fr.wikipedia.org/wiki/Ajax_(informatique)

[3] Mozilla, "Using xmlhttprequest," *MDN Web Docs*, n.d., [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[4] B. Rogier, "Qu'est-ce qu'une single page application (spa)," *kadiska*, 2021. [Online]. Available: https://kadiska.com/fr/blog-quest-ce-quune-single-page-application-spa/

[5] N. Li and B. Zhang, "The research on single page application frontend development based on vue," *Journal of Physics*, 2021. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1883/1/012030/pdf

[6] E. Team, "Top 10 des cms les plus populaires par part de marché (pour créer un site web)," *isitwp*, 2024. [Online]. Available: https://www.isitwp.com/fr/popular-cms-market-share/

[7] O. S. Matters, "Joomla," *Joomla*, 2005. [Online]. Available: https://www.joomla.org/4/fr/

[8] J. Documentation, "Développement d'un composant mvc- ajout d'acl," *Joomla Documentation*, 2005. [Online]. Available: https://docs.joomla.org/J3.x:Developing_an_MVC_Component/Adding_ACL/fr

[9] Wordpress, "Wordpress," *Wordpress*, –. [Online]. Available: https://wordpress.com/fr/

[10] WebsiteBaker, "Websitebaker," *WebsiteBaker*, –. [Online]. Available: https://websitebaker.org/en/home/

[11] A. T. Holdener, *AJAX The Definitive Guide*. O'reilly, 2008.

[12] None, "A guide to single page applications and headless content management," *Functionandform*, None. [Online]. Available: https://www.functionandform.co.uk/blog/what-is-a-single-page-application

[13] OWASP, "Cross-site scripting (xss)," *OWASP*, n.d. [Online]. Available: https://owasp.org/www-community/attacks/xss

[14] Imperva, "What is csrf (cross-site request forgery)?" *Imperva*, n.d., [Online]. Available: https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/. [Online]. Available: https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

[15] Adobe, "Spa et rendu côté serveur (ssr)," *Adobe*, 2024. [Online]. Available: https://experienceleague.adobe.com/fr/docs/experience-manager-65/content/implementing/developing/spas/spa-ssr

[16] OpenClassroom, "afficher du contenu dynamiquement," *openclassroom*, 2024. [Online]. Available: https://openclassrooms.com/fr/courses/8069121-perfectionnez-vous-sur-wordpress/8236097-affichez-du-contenu-dynamiquement

[17] IONOS, "What is jquery ajax?" *IONOS*, n.d. [Online]. Available: https://www.ionos.fr/digitalguide/sites-internet/developpement-web/jquery-ajax/

[18] J. Documentation, "Using joomla ajax interface," *Joomla Documentation*, n.d. [Online]. Available: https://docs.joomla.org/Using_Joomla_Ajax_Interface/fr

[19] C. WP, "Wordpress ajax basics," *Capitaine WP*, n.d. [Online]. Available: https://capitainewp.io/formations/developper-theme-wordpress/wordpress-ajax/

[20] D'Artagnan, "Ajax and dom manipulation with jquery," *D'Artagnan*, n.d. [Online]. Available: https://dartagnan.cg.helmo.be/~p150107/tutoriels/js-jquery-ajax/

[21] IONOS, "Wordpress vs joomla cms comparison," *IONOS*, n.d. [Online]. Available: https://www.ionos.fr/digitalguide/hebergement/cms/wordpress-vs-joomla/

[22] W. Documentation, "Websitebaker features overview," *WebsiteBaker Documentation*, n.d. [Online]. Available: https://websitebaker.org/en/home/

[23] J. Documentation, "Joomla rest api introduction," *Joomla! Documentation*, n.d. [Online]. Available: https://docs.joomla.org/J4.x:Joomla_Core_APIs

[24] mozilla. (2024) Using the fetch api. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

[25] Auth0, "What is json web token?" *Auth0*, n.d. [Online]. Available: https://auth0.com/learn/json-web-tokens/

[26] J. Documentation, "Access control list (acl)," *Joomla Documentation*, n.d. [Online]. Available: https://docs.joomla.org/Access_Control_List

[27] M. W. Docs, "Https - secure communication," *MDN Web Docs*, n.d. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[28] ——, "Window.localstorage," *MDN Web Docs*, n.d. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[29] W3Techs, "What is a cms?" *W3Techs*, n.d. [Online]. Available: https://w3techs.com/technologies/overview/content_management

[30] Postman, "What are apis and how do they work?" *Postman*, n.d. [Online]. Available: https://learning.postman.com/docs/getting-started/apis-introduction/

[31] Cloudflare, "What is caching?" *Cloudflare*, n.d. [Online]. Available: https://www.cloudflare.com/learning/cdn/what-is-caching/

[32] HubSpot, "Crm basics: What is crm?" *HubSpot*, n.d. [Online]. Available: https://blog.hubspot.com/marketing/what-is-crm

[33] G. Developers, "Static files in web development," *Google Developers*, n.d. [Online]. Available: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/

[34] WordPress.org, "Ajaxpress | transform your website into single page application," *WordPress.org*, n.d. [Online]. Available: https://wordpress.org/plugins/ajaxpress/

[35] W. D. Resources, "Rest api handbook," *developer.wordpress.org*, n.d. [Online]. Available: https://developer.wordpress.org/rest-api/

[36] C. WP, "Créer une requête ajax avec wordpress," *Capitaine WP*, n.d. [Online]. Available: https://capitainewp.io/formations/developper-theme-wordpress/wordpress-ajax/

[37] D. Brains, "Handling ajax requests in wordpress: Wp rest api vs admin-ajax.php vs must-use plugin," *Delicious Brains*, 2023. [Online]. Available: https://deliciousbrains.com/comparing-wordpress-rest-api-performance-admin-ajax-php/

[38] W. P. les Nuls, "Comment le ajax est-il utilisé dans wordpress?" *WP Pour les Nuls*, n.d. [Online]. Available: https://www.wppourlesnuls.com/glossaire/ajax/

[39] J. Forum, "Joomla acting as a proxy..." *Joomla! Forum*, 2008. [Online]. Available: https://forum.joomla.org/viewtopic.php?t=300954

[40] J. P. Documentation, "Web services," *Joomla! Programmers Documentation*, n.d. [Online]. Available: https://manual.joomla.org/docs/general-concepts/webservices/

[41] Joomla, "Spécification api joomla," 2021. [Online]. Available: https://docs.joomla.org/Joomla_Api_Specification/fr

[42] codingmall, "How can i integrate a custom rest api with existing joomla components," 2020. [Online]. Available: https://codingmall.com/knowledge-base/25-global/1876-how-can-i-integrate-a-custom-rest-api-with-existing-joomla

[43] JoomDev, "Token based authentication – how to use it in joomla 4," *JoomDev*, 2020. [Online]. Available: https://www.joomdev.com/token-based-authentication/

[44] WebsiteBaker, "Websitebaker home," *WebsiteBaker*, n.d. [Online]. Available: https://websitebaker.org/en/home/

[45] Auth0, "What is json web token?" *Auth0*, n.d. [Online]. Available: https://auth0.com/learn/json-web-tokens/

[46] OWASP, "Cross-site request forgery (csrf)," *OWASP*, n.d. [Online]. Available: https://owasp.org/www-community/attacks/csrf

[47] PHP.net, "Prepared statements," *PHP.net*, n.d. [Online]. Available: https://www.php.net/manual/en/pdo.prepared-statements.php

[48] W3Schools, "Json introduction," *W3Schools*, n.d. [Online]. Available: https://www.w3schools.com/js/js_json_intro.asp

[49] M. W. Docs, "Overview of http," *MDN Web Docs*, n.d. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[50] M. CMS, "Single-page applications," *Magnolia CMS Documentation*, n.d. [Online]. Available: https://docs.magnolia-cms.com/product-docs/6.3/features/single-page-applications/

[51] Auth0, "What is json web token?" *Auth0*, n.d. [Online]. Available: https://auth0.com/learn/json-web-tokens/

[52] ——, "Authentication in spa (reactjs and vuejs) the right way - part 1," *jcbaey.com*, 2018. [Online]. Available: https://jcbaey.com/authentication-in-spa-reactjs-and-vuejs-the-right-way/

[53] CodeOpinion, "Smarter single page application with a rest api," *CodeOpinion*, 2019. [Online]. Available: https://codeopinion.com/smarter-single-page-application-with-a-rest-api/

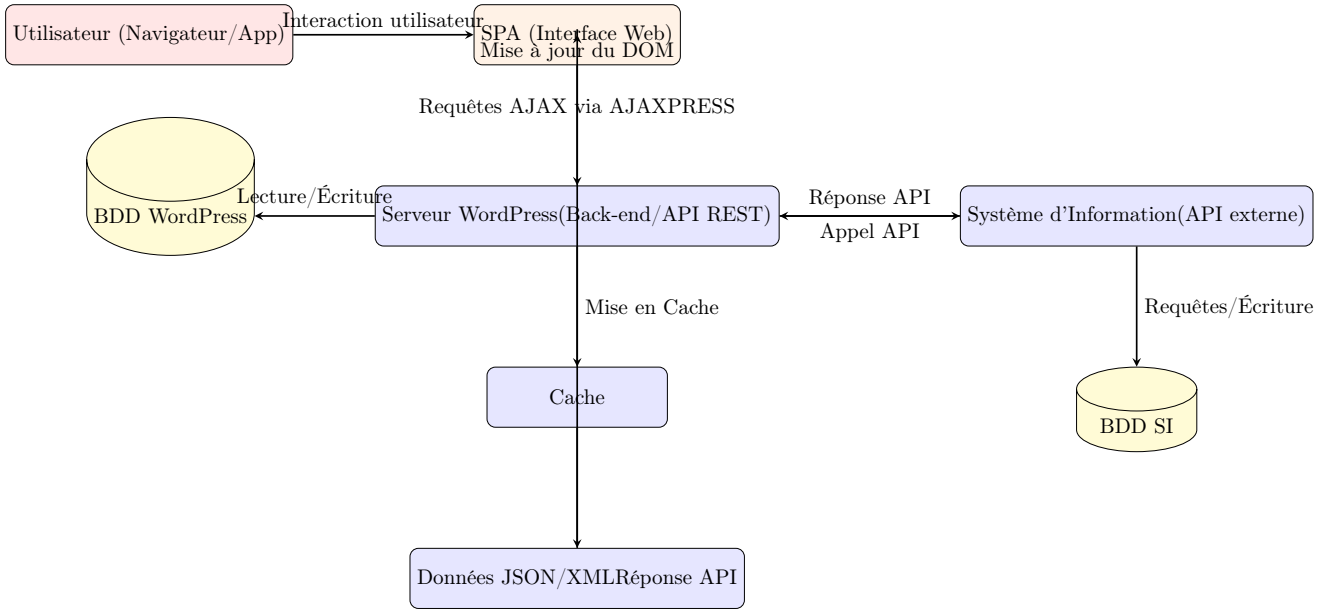
Part II

Annexes

Annexes

4 Schémas de réalisation pour les différents CMS

4.1 Wordpress



4.2 Joomla

Architecture 1:

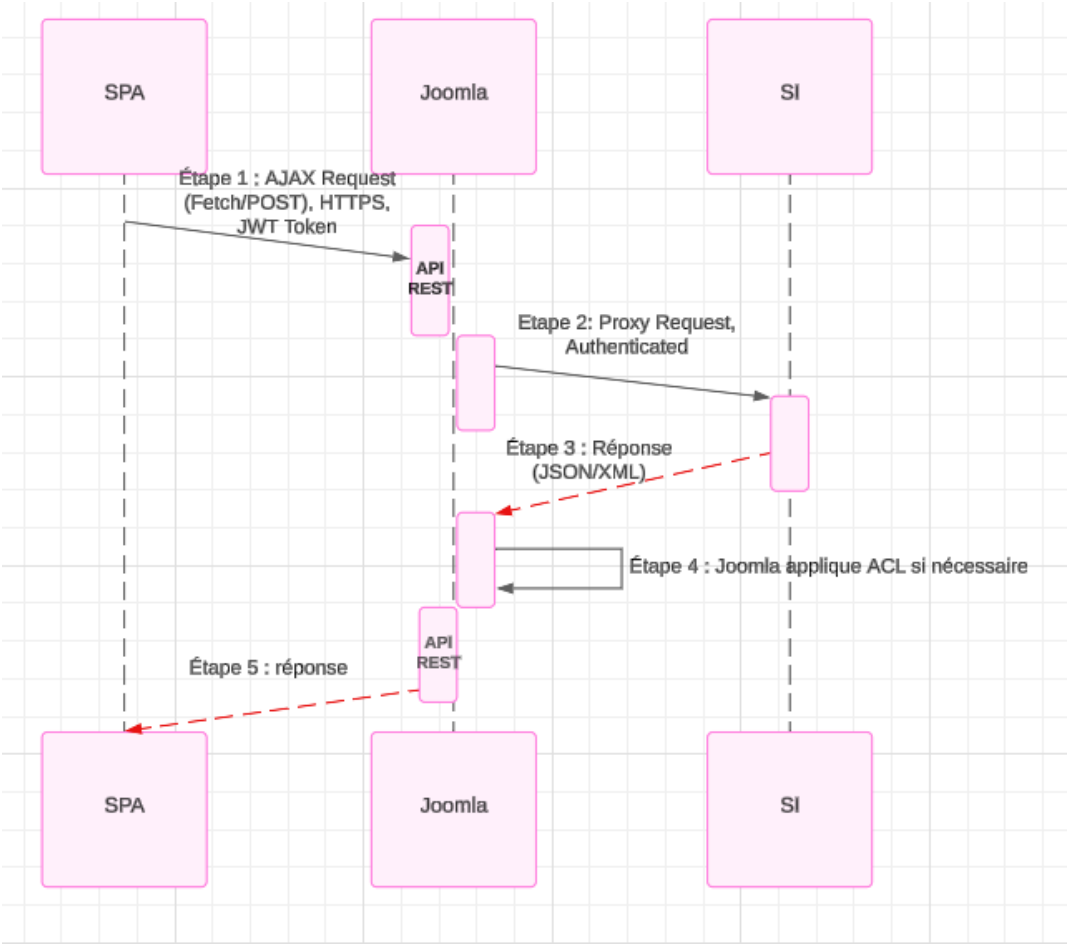


Figure 5: Schéma de l'architecture avec Joomla en tant que Proxy

Architecture 2 :

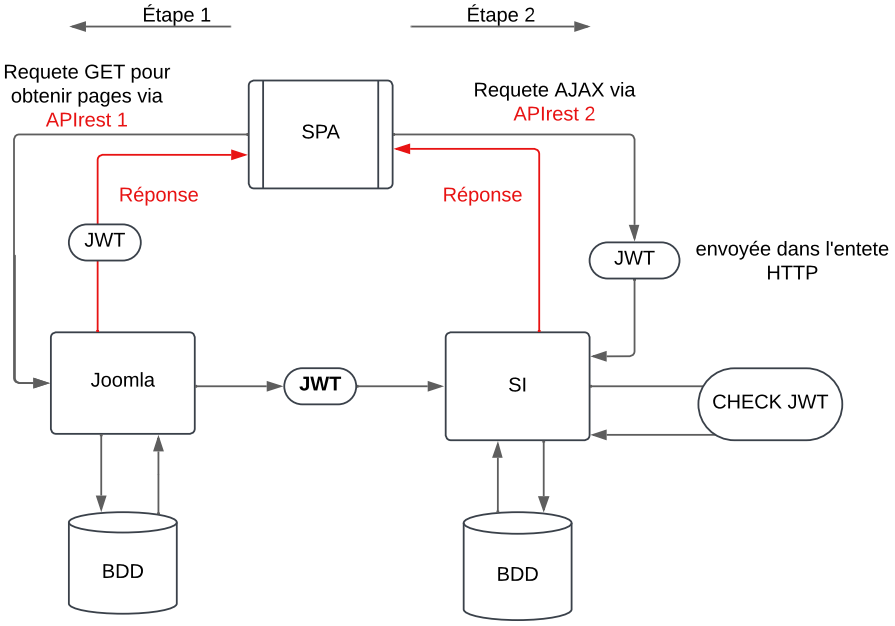
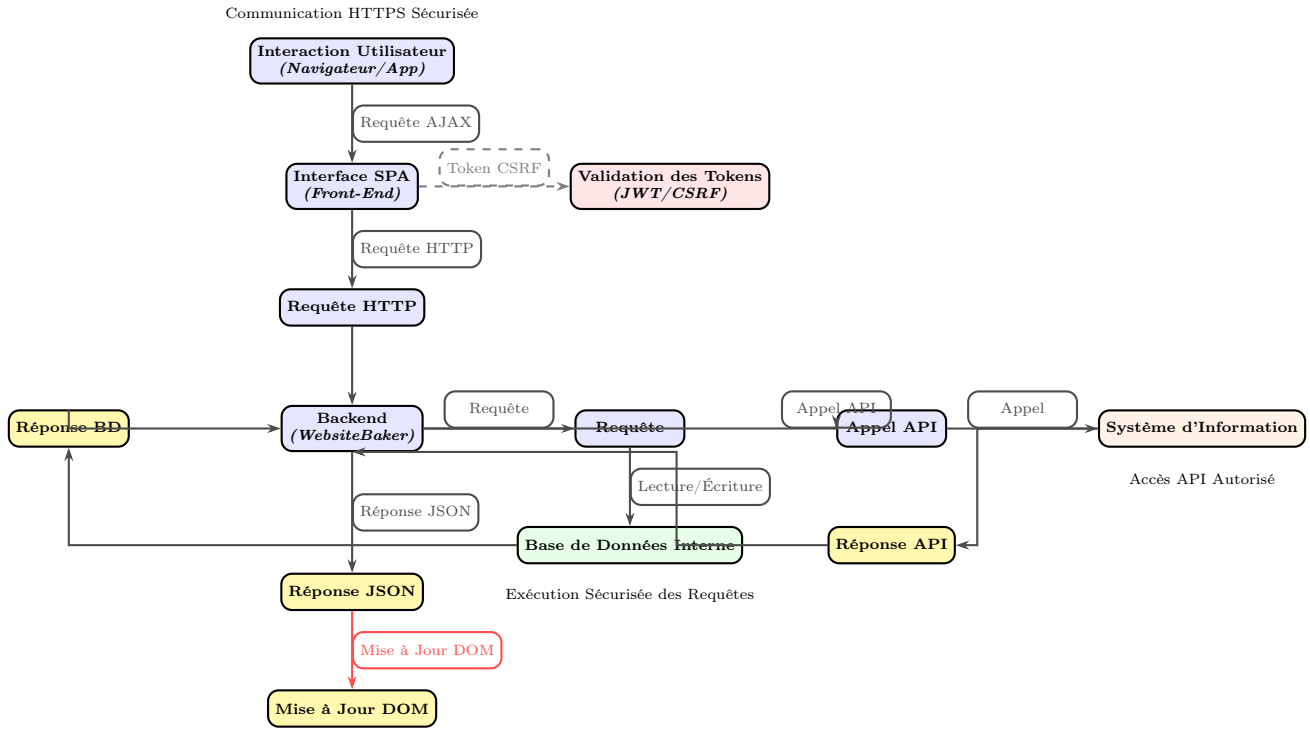


Figure 6: Schéma fonctionnel

4.3 WebsiteBaker



PLANNING PREVISIONNEL:		CMS-AJAX	Chefs de Projet	Louis CHARPENTIER										
			Membres	Orel AGUIAR	Ilyes ACHAQ	OthnielARYEE	Leandra CHEKAM NJOKO	Mohamed Walid NASSIR ELHAK	Kewtar BELKHETAB					
			Client	Mr. Lefevere										
Etapes / Taches	Livrable attendus/ Objectifs	Responsable des taches	17/03/25-22/03/25		24/03/25-29/03/25		31/03/25-05/04/25		07/04/25-12/04/25		14/04/25-19/04/25		21/04/25-25/05/25	
Jalons			Mise en place du projet fini		CMS configurés				Intégration AJAX par CMS fini interopérabilité et sécurité vérifiées.				Synthèse du projet terminé, diapositive de soutenance préparée	
Mise en place du Projet														
Installer LAMP sur les VM	Installer Apache, MySQL/MariaDB et PHP sur chaque machine virtuelle. Tester le fonctionnement de LAMP pour s'assurer qu'il est opérationnel sur chaque instance.	Orel AGUIAR												
Configuration des VMs	Créer trois instances séparées pour les trois CMS. Attribuer des ressources (CPU, RAM, disque) en fonction des besoins spécifiques de chaque CMS. Configurer les réseaux pour permettre une communication entre les machines (par exemple, en mode pont ou NAT).	OthnielARYEE												
Configurer les Bases de données	Créer une base de données unique centralisée pour gérer les réservations (partagée entre les CMS). Vérifier que chaque CMS peut se connecter à cette base via le Système d'information (SI).	Leandra CHEKAM NJOKO												
Déploiement des CMS														
Télécharger et installer les CMS	CMS 1 : Télécharger et installer (par exemple, WordPress). CMS 2 : Télécharger et installer (par exemple, Joomla). CMS 3 : Télécharger et installer (par exemple, Drupal ou un autre CMS choisi).	Ilyes ACHAQ												
Configurer les CMS	Configurer les paramètres de base : base de données, URL, utilisateurs administrateurs, etc. Activer les modules ou plugins nécessaires pour supporter les fonctionnalités (blogs, API). Appliquer un thème commun pour la cohérence visuelle	Mohamed Walid NASSIR ELHAK												
Implémentation des fonctionnalités														
Mise en place des réservations via le SI	Créer les API nécessaires sur le SI pour gérer les réservations (CRUD : création, lecture, mise à jour, suppression). Connecter chaque CMS au SI via des appels AJAX pour accéder aux données de réservation.	Louis CHARPENTIER												
Gestion des actualités via les CMS	Ajouter des sections blog dans chaque CMS pour permettre la publication d'articles. Vérifier que les articles sont correctement stockés dans la base de données propre à chaque CMS.	Kewtar BELKHETAB												
Intégration AJAX	Implémenter les appels AJAX dans les CMS pour récupérer ou mettre à jour des données sans rechargement. Tester les échanges de données entre les CMS, le SI et la base de données partagée	Mohamed Walid NASSIR ELHAK												
Configurer les connexions HTTPS	Installer des certificats SSL/TLS sur chaque serveur pour sécuriser les communications.	Ilyes ACHAQ												
Mettre en place la gestion des tokens	Générer et valider des tokens JWT pour sécuriser les échanges entre les CMS et le SI.	OthnielARYEE												
Effectuer des tests de sécurité	Vérifier la vulnérabilité des tokens et des appels API (tests manuels ou outils comme OWASP ZAP)	Leandra CHEKAM NJOKO												
Rédaction et documentation														
Créer une documentation technique	Décrire l'architecture mise en place (diagrammes, explications). Inclure les API créées pour le SI.	Orel AGUIAR												
Préparer les supports pour la soutenance	Créer une présentation claire montrant les étapes de la réalisation, les challenges rencontrés et les résultats obtenus	Leandra CHEKAM NJOKO												
Vérifications finales	Assurer que tous les CMS sont fonctionnels et interconnectés avec le SI. Vérifier que toutes les contraintes initiales du cahier des charges sont respectées.	Ilyes ACHAQ												
Sauvegarde et export des livrables	Sauvegarder l'ensemble des configurations et du code (base de données, fichiers sources, scripts). Préparer les livrables pour la soutenance finale	Kewtar BELKHETAB												

5 Note de clarification

5.1 Objectif du Projet

Le but principal du projet est de **préconiser des bonnes pratiques** pour séparer le **back-end** (le CMS qui gère les données) du **front-end** (l'interface utilisateur), en utilisant **AJAX** et les concepts de **SPA** (*Single Page Application*). L'objectif final est de permettre une **interopérabilité maximale** entre différents CMS et une architecture front-end moderne.

5.2 Déroulement du Projet

5.2.1 Étape 1 : Étude bibliographique (S7)

- **CMS** :
  - Comprendre comment les CMS comme WordPress, Joomla, et WebSiteBaker gèrent leurs données et contenus.
  - Analyser leurs capacités d'interaction via une API ou des extensions pour récupérer ou modifier dynamiquement des données.
- **AJAX et SPA** :
  - Étudier comment AJAX permet d'envoyer et de recevoir des données sans recharger la page.
  - Comprendre comment les SPAs utilisent AJAX pour interagir dynamiquement avec le back-end.
- **Interopérabilité** :
  - Identifier des moyens standardisés pour rendre différents CMS compatibles avec une approche AJAX/SPA.

5.2.2 Étape 2 : Preuves de faisabilité théorique (S7)

- Décrire théoriquement comment séparer le back-end et le front-end d'un CMS :
  - Proposer un scénario où un CMS comme WordPress servirait uniquement à stocker et structurer des données, tandis que le front-end serait entièrement géré par JavaScript (frameworks comme React ou Vue.js) et les données à caractères techniques serait géré par un autre système d'information.
- Identifier les défis techniques :
  - **Performance** : Les requêtes AJAX multiples ralentissent-elles l'application ?
  - **Sécurité** : Comment protéger les données transférées via AJAX ?
  - **Compatibilité** : Tous les CMS supportent-ils cette approche ? Quels sont les freins ?
- Documenter les avantages et les inconvénients de cette méthode :
  - Avantages : Flexibilité accrue pour les développeurs.
  - Inconvénients : Complexité technique et risque de performances réduites.

5.3 Preuves de faisabilité théorique attendues au S7

- **Étude comparative des CMS et d'AJAX** :
  - Identifier les CMS supportant AJAX nativement (API, extensions) parmi WordPress, Joomla, WebSiteBaker.
  - Décrire des cas d'utilisation réalistes pour AJAX dans le contexte de ces CMS.
- **Analyse des défis et recommandations** :
  - Identifier les problèmes techniques liés à la sécurité, aux performances et à la compatibilité.
  - Proposer des bonnes pratiques pour intégrer AJAX avec les CMS.

5.4 Rôle des POC (S8)

Les **Proof of Concept (POC)** à développer au S8 serviront à démontrer pratiquement les concepts étudiés théoriquement au S7.

– Exemple : Créer une page d'accueil alimentée dynamiquement par un CMS via AJAX, en utilisant uniquement une API REST pour obtenir les contenus.

– Tester plusieurs CMS pour vérifier si cette approche est adaptable à des plateformes variées.

5.5 Résumé de l'idée principale

En une phrase, le projet consiste à :

> "Étudier comment transformer un CMS en un système purement back-end gérant seulement des éléments comme des billets de blog, en utilisant AJAX pour gérer l'affichage des données côté front-end, avec des templates modernes et interopérables pour différents CMS et en reliant le tout à un système d'information gérant uniquement des fonctionnalités poussés comme des formulaires ou autres."

6 Cahier des Charges du premier semestre

6.1 Objectifs du Semestre

Objectifs

- **Objectif :** Démontrer de façon théorique l'existence d'un système général permettant la décorélation entre CMS, SI et SPA via des liaisons AJAX

6.2 Thématiques Abordées

- **Thématique 1 :** Cybersécurité : sécurisation des tokens de connexions
- **Thématique 2 :** Bibliographie et Recherche
- **Thématique 3 :** Développement Web : Approche théoriques des CMS, d'AJAX, des SPA et des liaisons entre les différentes entités

6.3 Contraintes et Exigences

- **Contraintes techniques :** La partie "utile" du rapport écrit ne doit pas dépasser 10 pages et d'être au minimum constitué de 8 pages
- **Contraintes temporelles :** 4 semaines de recherche et de synthèses.
- **Exigences spécifiques :** Le client, les membres du projets et les examinateur doivent comprendre l'objectif du projet pour le second semestre et les tenants et aboutissants de ce dernier.

6.4 Planning Prévisionnel

Voir le tableau de gantt : cliquez ici.

6.5 Livrable Attendu

Livrable

Un rapport écrit synthétisant les recherches et expliquant quel architecture sera étudié de façon pratique lors du second semestre et pourquoi. Le document rendu avec une bibliographie complète et des annexes permettant de comprendre les phases de recherche.

PLANNING PREVISIONNEL:		CMS-AJAX	Chefs de Projet	Louis CHARPENTIER						
			Membres	Orel AGUIAR	Ilyes ACHAQ	OthnielARYEE	Leandra CHEKAM NJOKO	Mohamed Walid NASSIR ELHAK	Kawtar BELKHETAB	
			Client	Mr. Lefevere						
Etapes / Taches	Livrable attendus/ Objectifs	Responsable des taches	25/11-30/11		2/12-6/12		9/12-13/12		15/12-20/12	
			26/11/2024 14:30		2/12/2024 14:30		Pas de réunion		17/12/2024 11:30	
Jalons										
Mise en place du Projet										
Analyse du sujet et des livrables attendus.	Document de cadrage du projet : objectifs, tâches, et répartition des responsabilités.									
Lecture approfondie de la documentation fournie (CMS, AJAX, SPA).										
Répartition des rôles pour la collecte d'informations (chaque membre peut se spécialiser sur un CMS ou une technologie spécifique)										
Étude bibliographique des CMS et d'AJAX										
Recherche sur les CMS ciblés : Fonctionnalités principales. Compatibilité avec AJAX et PHP.	Objectifs : identifier les principaux CMS en PHP et analyser leurs caractéristiques.									
Analyse des possibilités offertes par AJAX : Comment AJAX fonctionne dans une SPA avec PHP. Cas d'utilisation possibles avec les CMS.	Étudier les capacités d'AJAX pour interagir avec les CMS.									
	Livrables									
Comparaison des CMS (sous forme de tableau ou synthèse).	Synthèse sur les CMS analysés. Tableau comparatif des CMS (fonctionnalités, compatibilité avec AJAX, sécurité, performances). Documentation sur AJAX et ses cas d'utilisation.									
Analyse de faisabilité										
Identifier les cas d'utilisation réalistes : Exemples : récupération de contenus d'un CMS via AJAX, modification en temps réel.	Objectifs :									
Élaborer un diagramme d'architecture illustrant les interactions (client-serveur-CMS).										
Étudier les défis techniques : Sécurité : intégration de l'authentification et gestion des données sensibles. Performances : temps de réponse, impact sur le serveur. Synthétiser les points faisables et les limites techniques.	Évaluer la faisabilité technique de l'interopérabilité entre CMS et AJAX. Définir des cas d'utilisation pour les POC.									
Rédaction et présentation du rapport final										
Rédaction : Introduction et objectifs du projet. Synthèse sur les CMS et AJAX.	Objectifs : Restituer les résultats de l'étude bibliographique et de la faisabilité dans un document structuré.									
	Livrables									
Mise en forme et relecture collective.	Rapport final complet : Introduction et objectifs. Résultats de l'étude bibliographique. Analyse de faisabilité.									