2013/2014

M2 – Informatique - SSI

GULDNER Geoffrey
HURIER Médéric
LACAVE Valentin
SOIHILI Mouchtali

[Cahier des charges techniques]

SOMMAIRE

I. Modifications

Dans cette partie, nous avons décidé de regrouper toutes les modifications par rapport à l'ancienne version du cahier des charges. Toutes les autres informations seront des nouveautés ou des résumés qui viendront compléter notre précédente analyse. Il n'y a aucun copier-coller entre les deux cahiers des charges !

A. Fonctionnelle

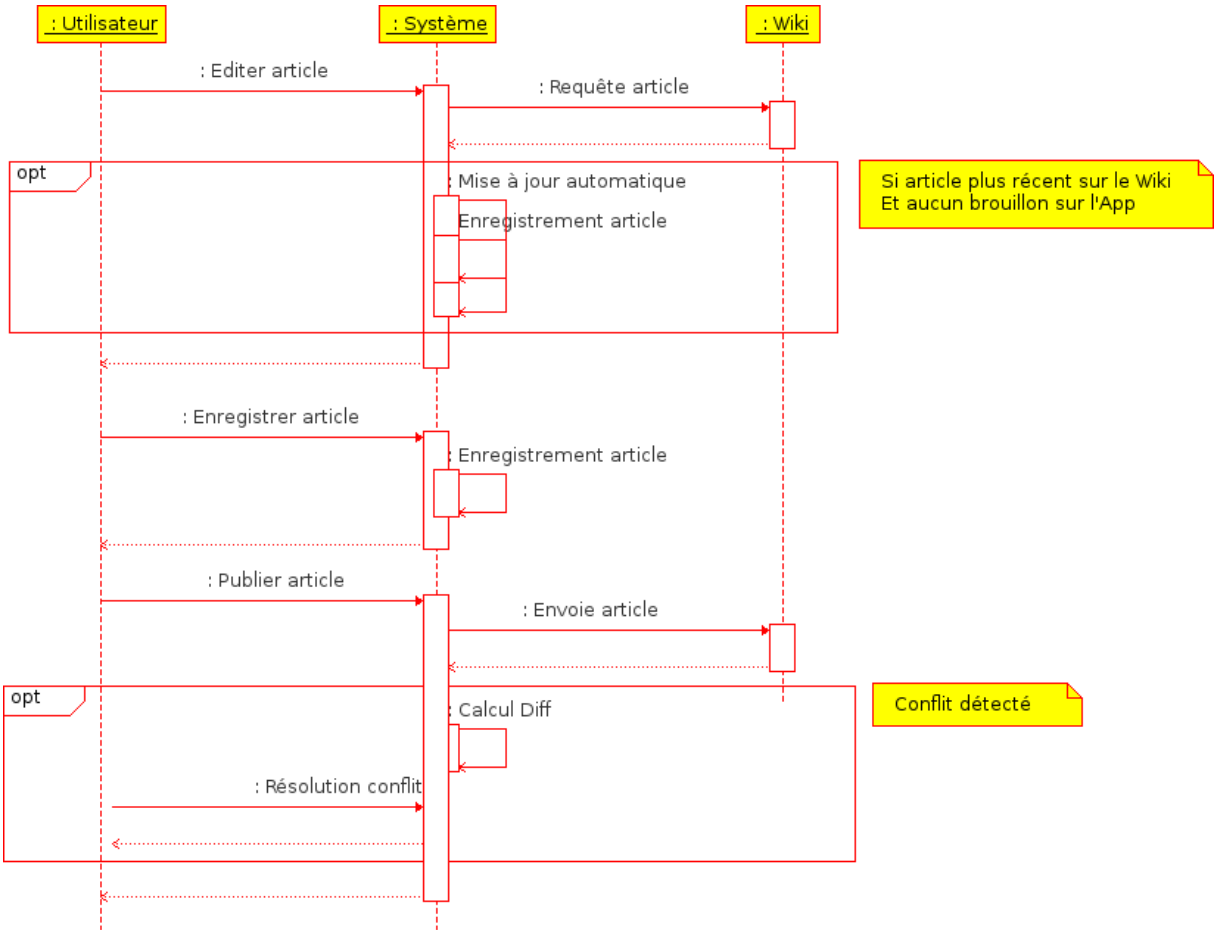
Lors de la réunion du jeudi 24 octobre 2013, plusieurs requêtes d'ordre fonctionnel nous ont été adressées. Nous en avons tenu compte dans nos nouvelles spécifications:

- l'utilisateur ne pourra pas supprimer un article du Wiki depuis notre application.
  - Il pourra toujours retirer l'article de notre application et le rajouter par la suite.
- notre application ne gèrera pas les images internes ou externes d'un article.
  - nous envisagerons de gérer les liens internes en fonction de notre avancement.
- la gestion des versions d'article sera accessible depuis la page édition de l'interface.

B. Logicielle

L'une des plus importantes fonctionnalités du projet est la synchronisation des articles. Dans notre précédente version, nous avons envisagé de gérer l'état de modification/suppression des articles grâce à un batch système. Avec le recul, cette solution nous semble gourmande en ressource pour des articles qui ne sont pas destinés à changer souvent.

Nous vous proposons une nouvelle version plus souple qui met à jour automatiquement l'article lorsque l'utilisateur décide de l'éditer, et qui vérifie les conflits de manière paresseuse (lazy) lorsque l'utilisateur décide de le publier.



Cette modification nous permet de regrouper 4 traitements en un seul. Nous espérons ainsi gagner en robustesse, en sécurité et en performances.

C. Technique

D'un point de vue technique, nous avons décidé d'utiliser un système d'exploitation **Ubuntu Server** plutôt que Debian. Les deux systèmes sont très similaires, mais nous avons trouvé de nouveaux arguments en faveur d'Ubuntu:

- possibilité d'installer des méta-paquets (outil tasksel)
- utilisation par défaut de super-utilisateur (sudo) plutôt que du compte root
- version plus récentes des paquets (similaire à Debian testing)

Tous nos autres choix techniques sont validés et détaillés dans la partie II.A de ce document.

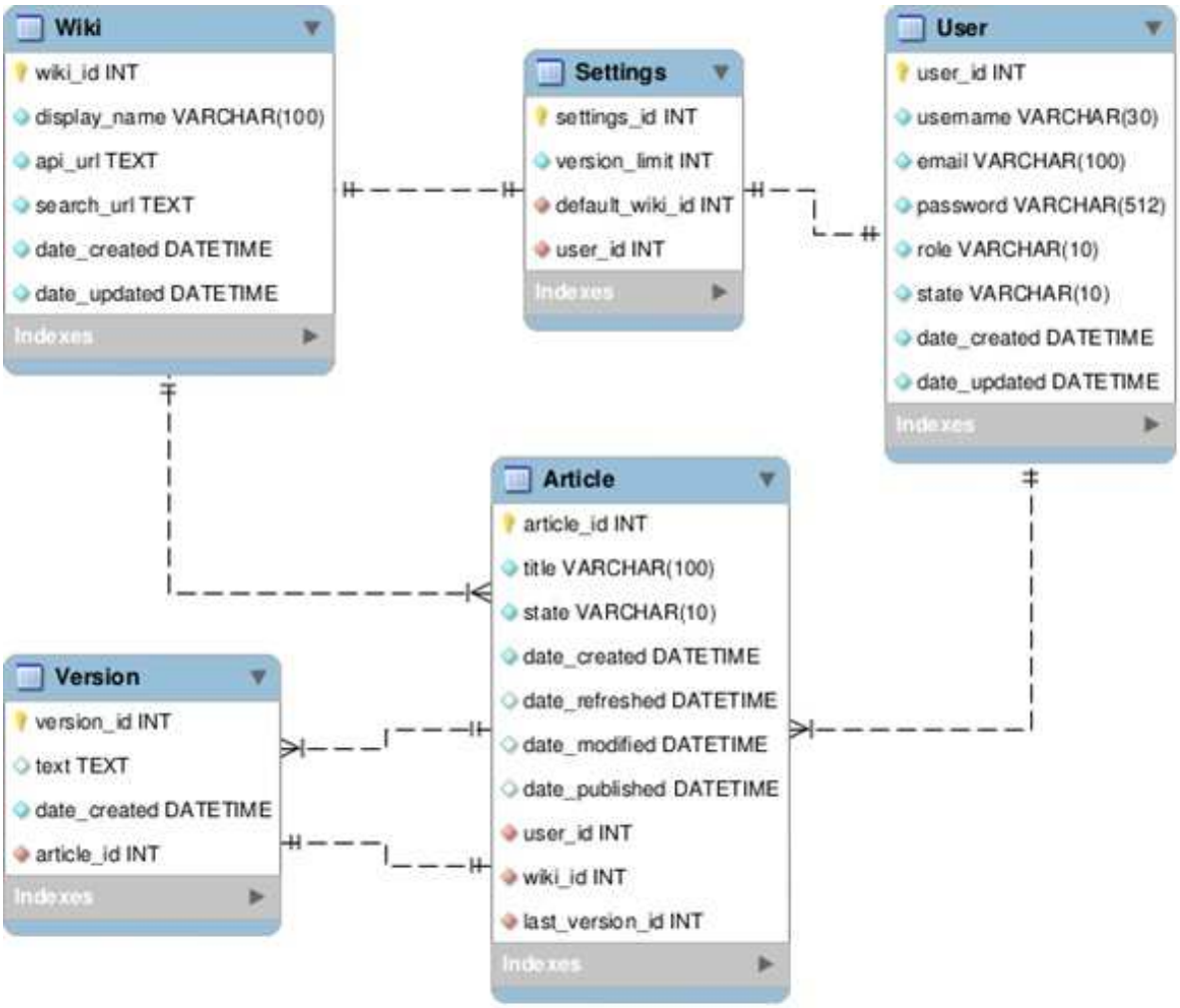
II. Conception

A. Modélisation des données

1) Schéma de la base de données

Le diagramme suivant reprend la structure physique de la base de données que nous avons détaillée dans notre précédente spécification. Pour aller plus loin, nous avons indiqué dans un tableau le nombre estimé d'enregistrement que nous attendons une fois le site déployé, et le niveau de confidentialité des données de la table.

L'échelle de confidentialité va de 1 (faible) à 4 (fort).



Nom	*Nature des informations*	*Estimation nb enregistrements*	*Confidentialité*
Wiki	nom et URL pour accéder à un Wiki	1 à 3	1
User	**authentification** et permissions	12	**4**
Settings	données interne à l'application	12 (= nombre User)	2
Article	dates et états d'un article	60 (~5 par User)	2
Version	texte d'un article	300 (~5 par article)	2

Les informations les plus critiques sont contenues dans la table User. Nous devons donc apporter une attention toute particulière aux secrets des données de cette table. Le niveau de confidentialité des autres données n'est pas très important car beaucoup sont publiques.

2) Contraintes d'intégrité

Les contraintes d'intégrité permettent d'assurer la cohérence des données insérées dans la table. Pour chaque table, nous avons identifié les contraintes suivantes:

Table Wiki:
- display_name: UNIQUE et format alphanumérique (pas de caractères spéciaux)
- api_url: format URL
- search_url: format URL
- aucun champ ne peut être NULL

Table User:
- username: format alphanumérique
- email: UNIQUE et format email
- password: donnée chiffrée !!
- role: {USER, ADMIN}
- state: {INSCRIT, BLOQUE, OK, INACTIF}
- aucun champ ne peut être NULL

Table Settings:
- version_limit: entre 0 et 10
- user_id: UNIQUE
- aucun champ ne peut être NULL

Table Article:
- title: format alphanumérique
- state: {OK}, {BROUILLON}, {EDITION EN COURS}, {CONFLIT}
- date_refreshed, date_modified, date_published, last_version peuvent être NULL

Table Version:

- aucun champ ne peut être NULL

Contraintes supplémentaires:

- l'encodage par défaut est UTF-8
- les modifications et suppressions seront répercutées en cascade
- le moteur de base de données doit supporter les clés étrangères (ex: InnoDB)

B. Traitement de l'information

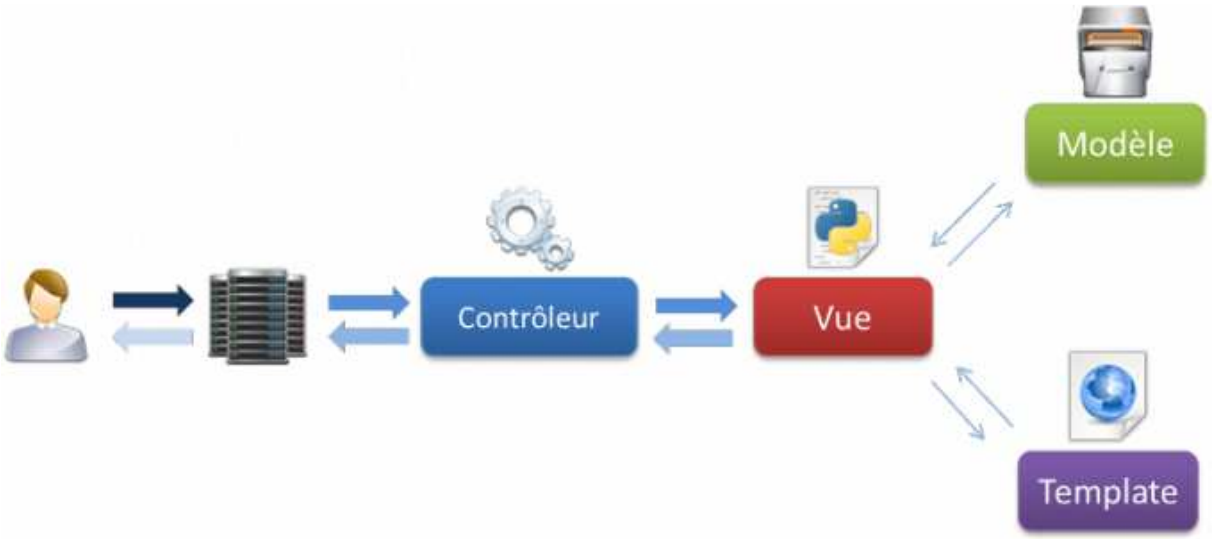
1) Patrons de conception

Pour traiter l'information au sein de notre application, nous allons nous appuyer sur un patron de conception éprouvé: le modèle MVC. Cette méthode d'organisation du code permet de diminuer au maximum les dépendances et de simplifier le processus de traitement.

C'est un critère important, car **des études montrent que 50% des problèmes liés à la sécurité ont pour cause la complexité des systèmes**:

- http://www.darkreading.com/vulnerability/survey-complexity-causes-security-incide/240009098

Le schéma suivant montre comment ce patron est implémenté au sein du framework que nous allons utiliser.



Les **contrôles,** tel que l'appel des traitements en fonction des requêtes (dispatch) ou les mécanismes d'authentification, sont gérés directement au sein du framework. Nous sommes en charge de l'implémentation des vues, du modèle et des templates.

Les **vues** sont des fonctions Python qui manipulent les données et les templates. Elles effectuent des vérifications sur la requête et renvoie une réponse à l'utilisateur (affichages, redirections, erreurs …).

Le **modèle** est une abstraction de la base de données (DAL: Database Abstraction Layer) codée également en Python. Des sécurités supplémentaires peuvent être mises à ce niveau, comme la vérification des formats (email, URL, alphanumérique …).

Enfin, les **templates** sont des fichiers HTML incorporant un langage à balise pour insérer le contenu des vues. Ils doivent filtrer les variables d'affichage pour éviter que des codes malveillants ne soient insérés (code HTML ou Javascript).

D'autres patrons et bonnes pratiques de programmation seront amenés grâce à l'utilisation du framework. Nous espérons ainsi nous reposer sur une base solide afin de renforcer la sécurité et d'assurer notre productivité.

2) Détails

La représentation schématique des traitements a été détaillée dans la spécification fonctionnelle. Pour résumer ces informations, nous allons les présenter sous forme de tableau avec les données en entrée, en sortie et les contrôles qui seront effectués.

Nom	*Entrée*	*Conséquences*	*Contrôles*
index	¤	¤	¤
home	¤	¤	¤
configuration [GET]	¤	¤	¤
profil [GET]	¤	¤	¤
créer article [GET]	¤	¤	¤
éditer article [GET]	id article	article => EDITION mise à jour auto article	vérification dernière version Wiki
inscription classique	adresse email	ajout nouvel utilisateur utilisateur => INSCRIT	adresse email unique
confirmation email	lien avec code	utilisateur => OK redirection profil	code valide et non expiré
inscription SSO	retour API SSO	association API SSO	¤
authentification	email mot de passe	création session redirection page home	authentification utilisateur utilisateur INSCRIT ou OK
déconnexion	¤	destruction session	¤
configuration [POST]	formulaire conf	mise à jour interface mise à jour Version	vérification formulaire
profil [POST]	formulaire profil	mise à jour User	vérification formulaire confirmation mot de passe
recherche	nom article nom Wiki	requête vers le Wiki	¤
suivre article	titre article nom Wiki	ajout nouvel article article => OK	l'article unique pour le User/Wiki
créer article [POST]	titre article texte	ajout nouvel article article => OK	l'article unique pour le User/Wiki
voir sur Wiki	titre article	redirection vers Wiki	¤
éditer article [POST]	id article texte article	modification article article => brouillon ajout nouvel version suppression anciennes versions	retrait balises <script>
arrêter suivi	id de l'article	suppression article suppression versions	demande confirmation

publier article	id de l'article	envoie sur le wiki article => OK OU article => conflit	pas de conflit avec le Wiki
résoudre conflit	id article texte article	article => OK OU article => conflit	pas de conflit avec le Wiki

C. Communication interservices

1) Authentication unique

Nous avons eu l'occasion de tester le mécanisme d'authentification unique (SSO) du point de vue d'un utilisateur en nous basant sur l'implémentation d'un autre site (senscritique.com). Les échanges entre l'utilisateur et les sites ont été documentés dans notre précédente spécification.

Cependant, nous n'avons pas eu le temps de développer un prototype pour vérifier en détail ce qui se passera du point de vue de notre application.

Nous avons sélectionné les outils suivants pour notre implémentation:

- extension de Mediawiki **SocialLogin**:
  - https://www.mediawiki.org/wiki/Extension:SocialLogin
- extension Python et Django **Social-Auth**:
  - https://github.com/omab/python-social-auth

2) Interface de programmation du Wiki

Mediawiki met à disposition **une API HTTP Rest** (Representational State Transfer) accessible publiquement sur tous les Wikis utilisant ce logiciel. Elle est activée sur le site Wikipedia.

Nous allons utiliser cette interface de programmation pour échanger les articles entre notre application et les logiciels de Wiki. Il nous suffira de récupérer le champ api_url de notre table Wiki pour publier l'article sur le site souhaité.

L'API se base sur des requêtes HTTP et nous renvoie un contenu **au format XML**. Pour gérer ces fonctions, nous allons créer une bibliothèque Python qui sera incluse à notre site.
Python est un très bon langage de manipulation de requête HTTP et de format XML, notamment grâce à ses bibliothèques **http.client** et **ElementTree**.
La documentation de l'API du Wiki est disponible à cette page:

- https://www.mediawiki.org/wiki/API:Main_page

III. Architecture

A. Environnement

Nos choix techniques ont été détaillés dans notre précédente spécification. Cette partie a pour but de les résumer et de regrouper les arguments qui ont motivé nos décisions.

1) Système

Nom	Catégorie	Description	Arguments
Ubuntu Server	Système d'exploitation	Système Linux basé sur Debian Testing	libre, gratuit, nombreux outils, facile à administrer
OpenSSH	Accès à distance	Shell sécurisé en ligne de commande	authentification par mot de passé et clé asymétrique
MySQL	Serveur de Base de données	Base relationnelle en mode client/serveur	populaire, bien documenté, performant à faible volume
Nginx	Serveur Web	Sert du contenu HTTP avec modèle asynchrone	performances très élevées modulaire, CK10 problem

2) Développement

Nom	Catégorie	Description	Arguments
Python	Langage de programmation	Langage haut-niveau, objet et dynamique	Utilisé par de grands groupes (Google, Nasa)
Django	Framework web	Outil libre et complet basé sur MVC	Nombreux outils de validation et productivité
Mediawiki	Logiciel de Wiki	Gestion du savoir moteur de Wikipédia	Spécification client
PhpMyAdmin	Administration SQL	Accès depuis le Web écrit en HTML/PHP	Simple, complet, évite d'ouvrir un port MySQL
Mercurial	Gestionnaire de version	Historique et collaboration logiciel	Rapide, décentralisé, multiplateforme
Redmine	Gestionnaire de projet	Gère les demandes et l'avancement	Déjà utilisé dans un cadre personnel
Kate	Éditeur texte	Editeur simple de l'environnement KDE	Choix personnel

B. Déploiement

La particularité de ce projet est de faire communiquer 2 applications web intégrées sur un seul système. Ce type de fonctionnement mutualisé est commun à beaucoup d'entreprises et d'hébergeurs en ligne mais induit de nombreuses questions:

- comment cloisonner les applications pour limiter l'impact des vulnérabilités ?
- comment partager les instances de logiciels (ex: base de données) ?
- comment gérer la multiplicité des outils (ex: langage de programmation, framework) ?
- comment permettre aux applications de personnaliser leur environnement ?
- comment allouer les ressources systèmes ?

Dans cette partie, nous allons décrire les applications avec leurs composants et dépendances.

1) Identification des applications

La première application est le logiciel de Wiki **Mediawiki**. A l'instar de Wikipédia, il doit être accessible à tous les utilisateurs, qu'ils soient connectés ou non à l'application. Depuis ce site, les utilisateurs peuvent voir et rédiger des articles et les administrateurs configurer son fonctionnement.
Ce logiciel a 3 dépendances:

- un serveur Web
- le langage de programmation PHP
- un serveur de base de données

Le détail est listé sur cette page:

- https://www.mediawiki.org/wiki/Manual:Installation_requirements

La seconde application est une application cliente à Mediawiki que nous sommes chargés de développer. Nous l'avons nommé **WikiWikiWeb**, en hommage au premier site de Wiki mis en ligne. Grâce à ce site, des utilisateurs moins expérimentés peuvent récupérer et modifier des articles en toute simplicité. L'inscription est cependant obligatoire pour y accéder !

Le site aura pour dépendance:

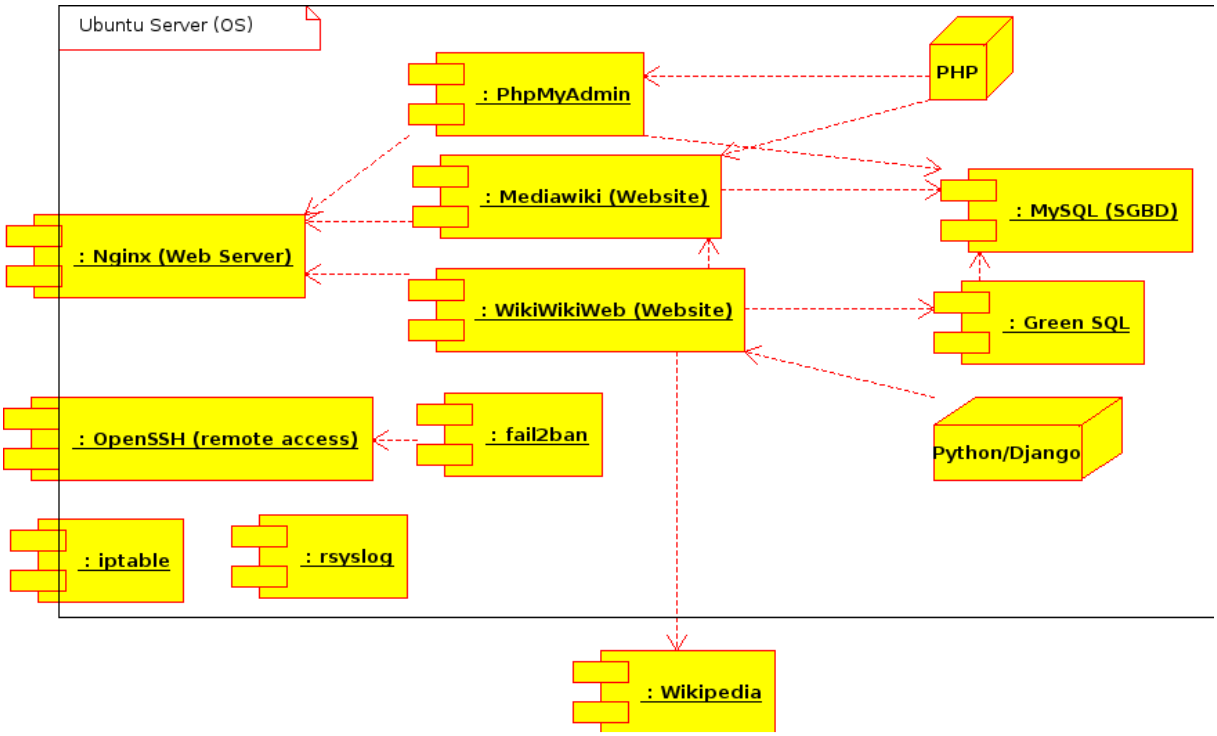
- un serveur web
- le framework Django et le langage de programmation associé Python
- un serveur de base de données

Nous sommes conscients qu'ajouter des composants à un système amène de nouvelles vulnérabilités. Aussi, nous avons fait le choix de faire cohabiter ces deux applications sur le même environnement.

Les deux applications utiliseront le serveur web Nginx et la base de données MySQL pour fonctionner. Nous devrons cependant cloisonner les applications aux seins de ces composants.

2) Diagramme de déploiement

Le diagramme suivant reprend le schéma de dépendance que nous avons détaillé dans la sous-partie précédente. Nous avons également ajouté les composants de sécurité que nous détaillerons dans la partie 3 de ce document.



La brique commune à tout le système est le système d'exploitation **Ubuntu Server**. Il héberge de nombreux services, dont certains sont exposés et d'autres non.

Les services exposés sont le serveur web **Nginx** et le logiciel d'accès à distance **OpenSSH**. Un service de pare-feu **iptable** permet de filtrer toutes les communications entrantes et sortantes. Tous les autres services devront être cloisonnés à l'intérieur du système pour limiter le risque d'exploitation.

A l'intérieur du système, on retrouve le serveur de base de données **MySQL** et le système de log **rsyslog**. Pour administrer les bases de données, une application spécifique **PhpMyAdmin** sera également installée.

Nous retrouvons nos deux applications **WikiWikiWeb** et **Mediawiki** avec leurs dépendances. Reposant sur un composant exposé, elles devront également être sécurisées.

Des mécanismes de sécurité comme **fail2ban** et **Green SQL** viennent s'interfacer à tous ses composants pour garantir une protection supplémentaire.

C. Instances applicatives

Tout projet informatique passe par une phase de **développement**, de **test** et enfin de **production**. Pour chaque phase, le logiciel va être déployé selon des critères précis:

- **développement**: un programmeur va coder et vérifier ses fonctions sur un environnement qui n'est accessible qu'à lui. Les ressources sont limitées et dépendent souvent d'un système local.
- **test**: plusieurs testeurs de la maîtrise d'œuvre et d'ouvrage vont effectuer des tests fonctionnels et de performance sur un environnement identique à la production.
- **production**: le grand public doit pouvoir utiliser le site dans les meilleures conditions possibles. Les tests doivent être limités pour ne pas nuire aux performances.

Dans le cadre de notre projet, nous avons fait le choix de nous passer de l'environnement de test. Déployer une instance spécifique à cette phase requiert une maintenance supplémentaire et une vigilance contre les menaces. De plus, ces instances sont souvent aussi accessibles mais moins sécurisées que le site en production.

Nous allons donc nous limiter à deux types d'instance, l'une en production sur le serveur mis à notre disposition, et l'autre en local sur nos propres machines.

1) Instance en production

Cette instance doit rester disponible, intègre et performante 24h/24 et 7j/7. Nos efforts de sécurisation vont se concentrer sur cet environnement, et des mécanismes de protection supplémentaires seront mis en place pour répondre à ces attentes.

L'infrastructure est la même que celle que nous avons décrit dans la partie II.B.2) de ce document. Avant que le site ne soit rendu public, nous allons réaliser des tests de performance et de vulnérabilité sur ce site.

La configuration matérielle est la suivante:

- Ordinateur tout-en-un HP Compaq
- Elite 8300 (unité de base)
- FreeDOS 2.0
- Intel Core i5-3470 3.2G 6M HD 2500 CPU B2T56AV
- 8GB DDR3-1600 SODIMM (2x4GB) RAM B2N40AV
- 500GB 7200 RPM 3.5 Hard Drive
- HP USB Keyboard - Win
- HP USB Mouse
- Slim DVD-ROM Optical Disc Drive

2) Instance en local

L'instance locale est un environnement plus souple et avec moins de mécanisme de sécurité. L'enjeu ici est de garantir la confidentialité du code source et des configurations. Aucun des services hébergés ne doit être exposé !

Certains logiciels, comme les bases de données ou les serveurs web, seront déclinés dans des versions plus facile et portable à utiliser. Par exemple, nous pourrons remplacer le serveur MySQL par SQLite ou le serveur Nginx par le serveur interne du framework Django.

Il est important de noter que compte tenu de ces différences, certains problèmes pourront avoir lieu lors du passage en production. **Ces problèmes sont un risque qui doit être anticipé et planifié en conséquence**. Nous avons réservé une semaine sur notre planning pour nous assurer que tout fonctionne comme prévu.

IV. Mécanismes de protection

A. Défense périphérique

1) Accès à distance

L'accès complet à un système par un attaquant est l'un des pires scénarios en termes de sécurité informatique. Il n'est alors plus possible de garantir la confidentialité, l'intégrité et la disponibilité de ses composants. Cependant, les administrateurs doivent avoir ce contrôle pour fournir les fonctionnalités attendues et utilise pour cela un logiciel d'accès à distance.

Nous avons fait le choix d'utiliser **OpenSSH** qui est un logiciel populaire et très sécurisé. Il établit une session distance en ligne de commande, identique à celle du terminal local, que l'utilisateur peut utiliser à condition de connaître le mot de passe ou d'avoir une clé asymétrique. Pour maximiser la sécurité de cette partie critique, nous avons fait le choix d'utiliser ces deux méthodes combinées en plus d'une configuration restrictive.

De plus, nous utiliserons OpenSSH en combinaison avec **Fail2ban**, un détecteur d'intrusion qui surveille les logs de différents services (dont SSH) pour bloquer les tentatives d'attaque par dictionnaire ou force brute.

Enfin, nous allons **brouiller au maximum les pistes de nos attaquants** en utilisant des noms d'utilisateurs plus compliqués (ggeoffreyy au lieu de geoffrey par exemple) et en utilisant des ports de service non standard:

- port 2244 pour SSH au lieu du port 22
- port 8899 pour PhpMyAdmin au lieu du port 80

2) Règles de filtrage

Les règles de filtrage comprennent des règles par défaut, des règles spécifiques et des contre-mesures:

- Refuser toutes les entrées
- Refuser toutes les sorties
- Autoriser le port TCP SSH en entrée
- Autoriser le port TCP Web en entrée / sortie
- Ignorer les requêtes ECHO ICMP
- Ignorer les requêtes/réponses REDIRECT ICMP
- Ignorer les paquets SYN invalides et les attaques SYN FLOOD
- Ignorer les paquets XMAS et NULL mal formés

D'autres règles de protection pourront être ajoutées en fonction des attaques que nous découvrirons. Nous utiliserons le pare-feu **Iptables** de Linux pour implémenter ces règles.

De plus, nous combinerons ces règles avec le logiciel **Snort**, un IPS (Intrusion Prevention System) populaire sur Linux. Il permet entre autre d'éviter le scan des ports et la prise d'empreinte du système.

B. Défense applicative

1) Validation des données

Au sein de notre application, **le framework Django** fournit de nombreuses fonctions pour valider les données. Nous nous reposerons sur ces fonctions pour nous assurer que les données en entrée et en sortie soient valides:

- validation des formulaires:
  - https://docs.djangoproject.com/en/dev/ref/forms/validation/
- validation des modèles:
  - https://docs.djangoproject.com/en/dev/ref/models/instances/
- validation des templates:
  - https://docs.djangoproject.com/en/dev/ref/templates/builtins/
- création de validateur:
  - https://docs.djangoproject.com/en/dev/ref/validators/

Nous sommes également conscient qu'une des failles de sécurité les plus importantes pour un site web est **l'injection SQL**. Elle est listée comme faille numéro 1 dans le projet TOP 10 OWASP (A1). Pour répondre à ce problème, nous allons employer le pare-feu SQL **Green SQL** afin de filtrer les tentatives d'injection. Plus d'information à cette page:

- http://www.greensql.com/article/protect-yourself-sqli-attacks-create-backdoor-web-server-using-mysql

Django fournit également des mécanismes intégrés de protection contre les injections SQL.

2) Configuration des sites

Une autre faille de sécurité importante pour notre projet est **la mauvaise configuration des applications ou de leurs composants**. Cela comprend les parties A5 (Security Misconfiguration) et A10 (Using Components with Known Vulnerabilities) du projet TOP 10 d'OWASP.

Il est important que nous passions en revu toutes les recommandations de sécurité des éditeurs et que nous suivions les nouvelles failles. Pour nos deux applications, nous avons trouvé des ressources sur le site de l'éditeur:

- Security in Django: https://docs.djangoproject.com/en/dev/topics/security/
- Security in Mediawiki: https://www.mediawiki.org/wiki/Manual:Security

Nous avons également d'autres recommandations sur les dépendances:

- Security in Nginx:
  - http://www.cyberciti.biz/tips/linux-unix-bsd-nginx-webserver-security.html
- ModSecurity:
  - http://www.modsecurity.org/projects/modsecurity/nginx/
- Security in MySQL:
  - http://www.greensql.com/content/mysql-security-best-practices-hardening-mysql-tips
- Security in PHP:
  - http://www.cyberciti.biz/tips/php-security-best-practices-tutorial.html

C. Durcissement du système

1) Cloisonnement

Après avoir étudié la possibilité d'utiliser un cloisonnement léger de type **LXC** ou **OpenVZ**, nous avons conclu que **ces solutions ne sont pas adaptées à notre projet**. Le cloisonnement des deux applications demandera un travail similaire sur la base de données, et notre temps d'implémentation est limité. De plus, nous n'avons pas les compétences sur ce genre d'outils.

Toutefois, nous voulons isoler le plus possible ces deux systèmes. Chaque application sera installée dans un site virtuel qui lui est propre sur le serveur web, et un utilisateur du serveur de base de données sera associé à chaque application.

2) Chiffrement

Le chiffrement garantit la confidentialité, l'intégrité et l'authentificité d'un composant.

Dans le cadre de notre projet, nous avons listé tous les éléments qui nécessitent d'être chiffré:

- chiffrement des mots de passe de la base de données (PBKDF2 + SHA256)
  - https://docs.djangoproject.com/en/dev/topics/auth/passwords/
- chiffrement des sites web (RSA 2048)
  - http://wiki.nginx.org/HttpSslModule
- chiffrement de l'accès à distance via SSH (DSA)
  - http://www.employees.org/~satch/ssh/faq/ssh-faq-1.html

Les menaces physiques n'étant pas gérés par notre équipe, nous n'allons chiffrer les partitions du système. Nous pouvons espérer ainsi gagner en performance.

V. Politique de sécurité

A. Communication des évènements

Les rapports de sécurité sont un des éléments les plus confidentiels d'un système d'information. Il est important de les communiquer de manière discrète et professionnelle aux personnes concernées. Dans cette partie, nous vous proposons de visualiser le template de nos rapports.

1) Rapport d'attaque

Voir en Annexes

2) Rapport de défense

Voir en Annexes

B. Plan de sauvegarde

1) Classification des informations

La sécurité d'un système se paye toujours en temps et en budget. Il est impossible de tout sécuriser de la meilleure manière possible, et des choix doivent être faits constamment par la direction et le RSSI.

Au-delà de la vision informatique, c'est l'information et le service rendu qui donnent la valeur d'un système. Hiérarchiser ces éléments permet de donner une priorité lors de l'amélioration continue des processus de sécurité.

Dans le tableau suivant, nous avons listé les informations de notre système qui ont le plus de valeur par ordre décroissant:

Nom	*Portée*	*Impact*	*Méthode d'obtention*
Authentification des administrateurs	Tout	contrôle total du système	attaque sur mot de passe, trahison, espionnage
Authentification des utilisateurs	Utilisateur	usurpation d'identité et vol de données	accès à la base de données, interception des communications
Code d'accès à la base de données	Base de données	vol de données, fausses informations	interception des communications, accès aux configurations du système
Code source des applications	Applications	Injection de code déni de services	aspiration de site web accès au système de fichier
Code d'accès au serveur de mail	Serveur mail	Lecture d'email confidentiel, spams	accès aux configurations
Clés de chiffrement	Données	Vol d'informations	accès au système de fichier
Règles de filtrage	Système	Facilite l'accès au système	accès aux configurations
Configuration des services	Services	Facilite l'accès au système	accès au système de fichier

Schéma de la base de données	Base de données	Facilite l'accès au système	espionnage accès à la base de données
Nom et version des logiciels utilisés	Système	Facilite l'accès au système	scan des ports prise d'empreinte

Ce tableau illustre la variété offerte aux attaquants pour nuire à un système. Espionnage, écoute des canaux de communication, exploits … toutes ces méthodes permettent de récupérer des informations plus ou moins critiques qui aideront à en récupérer davantage.

2) Méthode de sauvegarde

Nous pouvons utiliser facilement un script **rsync et cron** pour sauvegarder les fichiers de configurations, les logs et les dumps de la base de données à intervalle régulier. Le plus difficile est ensuite de conserver les sauvegardes de manière sûre et de ne pas nuire aux performances du système lors de la sauvegarde !

Pour sécuriser les sauvegardes, nous allons créer un utilisateur dédié qui aura accès uniquement aux informations nécessaire. Nous pouvons pour cela utiliser les **ACL** de Linux pour gérer finement les permissions d'accès.

Nous comptons réaliser des sauvegardes complètes de manière hebdomadaire. Jusqu'à 3 sauvegardes pourront être conservées, plus une pour la configuration initiale du serveur après la mise en production. **Elles seront archivées au format ZIP avec un mot de passe et exportées chaque semaine sur un espace de stockage externe**.

C. Plan de reprise d'activité

1) Prioritisation des services

Si les informations sont le sang d'un système, les services en sont le cœur. Nous avons établi la hiérarchie des services en partant du plus important au moins important

Nom	*Catégorie*	*Impact*
Accès à distance	Connexion à distance	impossible d'administrer les machines, de prévenir ou de contrer les attaques
Accès à la base de données	Base de données	impossible d'accéder aux sites web
Accès à Médiawiki	Application web	impossible d'accéder aux informations publiques, atteinte à l'image du groupe
Accès à WikiWikiWeb	Application web	impossible à nos utilisateurs novices de créer de nouveaux articles
Log système	Log	impossible de tracer les attaques et les accès
Filtrage des tentatives d'intrusion (snort)	Sécurité	facilite l'entrée sur le système
Filtrage du pare-feu (iptable)	Sécurité	facilite l'entrée sur le système
Filtrage SQL (Green SQL)	Sécurité	facilite l'injection de requête SQL
Filtrage des tentatives d'accès (fail2ban)	Sécurité	facilite l'entrée sur le système

2) Reconstruction en cas de sinistre

L'analyse précédente montre encore une fois l'importance de sécuriser **l'accès distant (OpenSSH)**. C'est le premier service qui devra être remis en place en cas d'incident, et il doit être très difficile pour un attaquant d'en prendre le contrôle.

Vient ensuite **la base de données** (**MySQL**), qui est la dépendance principale des deux applications web.

Puis **les deux applications web (Mediawiki et WikiWikiweb)** doivent être remises en marche, car c'est ce qui fait notre valeur ajoutée. Si ces deux applications tombent en panne, l'utilisateur n'a plus aucun intérêt à consulter notre site.

Le service de log (**rsyslog**) devra également être remis en marche rapidement pour nous permettre de vérifier les attaques contre le système et le bon fonctionnement des services.

Enfin, **les services de sécurité (Snort, iptable, Green SQL, fail2ban)** seront remis en marche à la fin. Ces derniers servent surtout à rendre plus difficile l'attaque sur notre système, mais n'ont pas de valeur métier.

Pour remettre en marche le système, **nous utiliserons les sauvegardes** que nous avons détaillées dans la partie précédente. Si nous disposons d'assez de stockage externe, nous pourrons également créer une image du disque.

Pour terminer, nous procéderons à un test de toutes les fonctionnalités essentielles de notre système.

VI. Annexes