{LOGO PROJET}

CAHIER DES CHARGES

Étude et définition du projet FSP

Fansub Site Project

Sazaju HITOKAGE
+33 (0)6 27 45 61 38
sazaju@gmail.com

L'École du Fansub
http://zerofansub.net/archives/ecole/

db0
zero.fansub@gmail.com

Table des matières

N° VERSION		MODIFICATIONS	DATE
Ver	Rév		
0	1	Insertion du contenu déjà rédigé sur le forum et formatage	10/06/10
	2	Rédaction d'une partie du chapitre III. Solution proposée	10/06/10
	3	Ajout de la partie III.C. Définitions dans la partie III. Solution proposée	12/06/10
	4	Quelques corrections ont été apportées La partie III.C. Définitions a été réécrites avec quelques changements La partie III.D. Fonctionnalités a été partiellement complétée	13/06/10
	5	Légère réécriture et ajouts dans la partie III.D.2. Fonctionnalités membre Réécriture de ce tableau avec les liens directs vers les parties Complétion du chapitre III.D.1. Fonctionnalités équipe Complétion du chapitre III.C.3. L'organisation d'une équipe de Fansub	14/06/10
	6	Réécriture de III.D.1.a. Gestion des tâches Complétion de III.D.2.a. Gestion des fichiers Restructuration de IV. Implémentation	15/06/10
	7	Réécriture de III.C.3. L'organisation d'une équipe de Fansub, principalement la description des rapports Réécriture de quelques points de III.D. Fonctionnalités relativement à la réécriture précédente Début de rédaction de III.D.1.c. Gestion des droits Ajout de la structure du chapitre III.E. Interface graphique	16/06/10
	8	Réécriture de III.D.2.a. Gestion des fichiers et III.D.4. Fonctionnalités connecté Retrait d'un point inutile dans III.D.1.b. Gestion des projets (affichage de l'avancement, déjà géré par les tâches)	19/06/10
	9	Réorganisation de III. Solution proposée (passage de l'interface graphique avant les fonctionnalités) Début de rédaction de III.D. Interface graphique et réflexion sur l'agencement de l'interface (hors document)	26/06/10
	10	Rédaction de III.D.1. Interface Détails des pages de l'interface (III.D.2. Pages publiques, III.D.3. Pages privées et III.D.4. Pages réservées)	27/06/10
	11	Relecture, légères modifications (tournures de phrases)	29/09/10
	12	Légère réorganisation de III.C. Définitions Création de III.C.4. Les conversations : discussions et débats (reprise des conversations de III.E.4. Fonctionnalités connecté)	01/10/10
	13	Réorganisation de III.D. Fonctionnalités avec réécriture de certaines parties	02/10/10

Introduction

De nombreux sites Web voient le jour depuis quelques années, d'envergure plus ou moins grande, et beaucoup de personnes s'essayent à faire **leur propre site**. Ces derniers recouvrent d'ailleurs de nombreuses formes (pages personnelles, blog, forum, ...) et vont des plus **personnalisés** (généralement fait presque intégralement à la main) aux **modèles configurables** tels que phpBB. Plusieurs sites professionnels permettent d'ailleurs de faire, par exemple, son propre forum sans s'occuper de sa mise en place, et cela gratuitement (tels que xooit ou forumactif). Ces solutions permettent aux moins aguerris de faire leurs premiers pas dans la création de sites Web.

À côté de l'essor de ces sites personnels, et plus globalement du **partage via Internet**, de nombreuses technologies se sont vues prises d'assaut par des programmeurs de tous niveaux. Que ce soit pour du simple **affichage textuel**, avec des pages (X)HTML et des flux RSS, ou du partage de **contenu binaire**, sur réseaux Emule et BitTorrent ou via FTP, voire des solutions **hybrides** tels que les serveurs IRC, initialement prévus pour le chat massif mais intégrant aujourd'hui des fonctionnalités d'échanges de fichiers. Il est important aussi de comprendre que la **présentation** de ces données prend une part cruciale dans le développement de ces applications, notamment les sites Web, avec le formatage par feuilles de styles CSS, et la génération dynamique de données, via des scripts (côté serveur ou client). La mode actuelle n'est plus à la programmation pour le programmeur, mais à la création de **systèmes universels**, utilisables par le plus grand nombre.

Profitant de la diversification des procédés de partage, le **Fansub** (de *Fan Subbing*, sous-titrage d'*animes* et *dramas* principalement) a lui aussi prit du terrain : sites d'équipes, sites de téléchargement, sites de référencement et canaux de discussion sont le lot quotidien de nos fans de sous-titres. Les **cours de langues** abondent sur le net pour que tout un chacun puisse se lancer dans ces apprentissages longs mais pleins d'ouvertures, les logiciels de **traitement vidéo** ont eux aussi suffisamment évolué pour permettre à nos amateurs de proposer des sorties dignes de versions DVD. Le prochain objectif est maintenant d'arriver à mettre en place une structure collective facilitant **l'organisation, la communication et la participation** de chacun des membres d'une équipe de Fansub.

C'est dans l'esprit de formation des élèves de l'**École du Fansub** que ce projet a vu le jour, aussi l'objectif du projet **Fansub Site Project** (FSP) est double :

- tout d'abord de profiter de cet essor et de l'expérience acquise par la communauté des développeurs, de façon à **créer un site Web pluridisciplinaire**, orienté sur les besoins d'une **équipe de Fansub**, grâce à des technologies et langages qui ont fait leur preuve et offrant des possibilités de développement riches et puissantes,

- ensuite de permettre aux élèves programmeurs de l'École du Fansub de disposer d'un projet concret sur lequel ils peuvent se casser les dents, de façon à acquérir une **expérience riche et passionnante**, à la fois pour leur propre passion et pour leurs compétences futures.

Synchronisation des tâches, communication ciblée et procédures optimisées : la prochaine génération du monde du Fansub est en marche, faites place aux pros !

I. Problématique

Ce projet reste en soi un projet "classique" de développement Web, peu innovant sur le principe, même si cela n'a pas encore été fait (tout du moins pas au niveau escompté ici). N'ayant pas de contraintes de **temps** ou de **financement** particulières, et disposant de ressources Open Source gratuites suffisamment riches, le projet ne devrait pas subir de problèmes techniques particuliers. Le point sur lequel l'attention est portée est avant tout la **fusion des besoins** écoliers et utilisateurs finaux.

I.A. Point de vue école

Nous cherchons à **former des élèves**, qui doivent donc arriver à tirer une expérience qu'ils peuvent utiliser par la suite. En particulier si celle-ci peut s'inscrire dans leur CV pour justifier à un emploi, c'est tout ce qu'on peut leur souhaiter. Par conséquent, les technologies ne sont pas choisies uniquement sur les besoins **utilisateurs**, mais aussi ceux des **développeurs** ! À ce titre, il faut s'assurer que les technologies utilisées offrent un **maximum de possibilités**, de façon à répondre à la multitude des besoins utilisateurs, et qu'elles soient **porteuses sur l'avenir**, pour que l'expérience acquise par nos élèves soit utilisable dans de nombreux domaines.

Bien entendu, il n'est pas question de partir tête baissée dans un domaine non maîtrisé, aussi les technologies choisies sont **prises en charge** par au moins un des membres du corps enseignant de l'École du Fansub, de façon à ce que les élèves perdus puissent se rattacher à quelqu'un ayant des connaissances solides et pouvant les aider à sortir la tête de l'eau. Si une technologie ne peut être **suivie efficacement** par aucun des enseignants, elle ne peut pas être utilisée au sein du projet, à moins qu'au moins un enseignant décide de s'y former.

I.B. Point de vue utilisateur

Avec l'évolution des technologies et l'expérience acquise dans le développement Web, il est désormais possible de faire des sites Web **ayant une personnalisation poussée** (génération dynamique côté serveur) et **très réactifs** (scripts côté client). L'objectif de ce projet est donc de développer un site Web permettant de profiter de ces possibilités pour offrir un outil efficace aux **équipes de Fansub**, en particulier pour :

- **organiser leur travail**
  - définition des tâches (recherche de RAW, traduction, karaoké, encodage, distribution, ...)
  - allocation des tâches (prise en charge seul ou à plusieurs)
  - dynamisme des tâches (création, annulation, organisation, ...)
  - identification des membres (rôles, responsabilités, travaux, ...)
- **permettre une gestion aisée des données**
  - mise à disposition et droits d'accès
  - mise à jour et comparaisons
  - archivage et suppression
  - vérification des disponibilités
  - automatisation des tâches répétitives

- **faciliter la communication**
  - ergonomie claire et pratique
  - présentation de l'équipe
  - prise en compte de l'origine multiple des utilisateurs (différents navigateurs, téléphone portable, personnes handicapés, ...)
  - messages personnels & messages de masse
  - news & newsletter
  - commentaires, sondages, ...
  - référencement
  - ...

II. Technologies existantes

Il convient tout d'abord de séparer **technologies et *frameworks*** (littéralement *cadres de travail*). Une **technologie** est la description (définition, cas d'utilisation, ...) d'un ensemble d'outils de base, pouvant avoir plusieurs implémentations équivalente : le **langage HTML** dispose d'une définition standardisée et est implémenté spécifiquement pour chaque navigateur (MicroSoft/Windows Internet Explorer, Mozilla Firefox, NetScape Navigator, Opera, Safari, ...), bien que certaines fonctionnalités puissent avoir des effets différents selon le respect des normes. Un ***framework*** est un ensemble d'outils spécialisés, utilisant une(des) technologie(s) particulière(s) : **Hibernate** offre un ensemble de classes Java (technologie) permettant de faciliter la persistance des données (spécialité), il n'est donc pas adapté pour la création graphique.

Comme nous souhaitons garder un maximum de **marge de manœuvre**, et éviter de spécialiser nos élèves, il va de soi que les *frameworks* sont à **éviter**. En particulier, même si industriellement le point de vue est inversé pour gagner du temps, apprendre à utiliser des outils sans en **connaître et maîtriser les bases** est peu recommandé en matière de pédagogie (le risque étant de limiter la vision de l'élève à ce qu'il **sait** faire, au lieu de l'élargir à ce qu'il **peut** faire, et ainsi limiter ou empêcher l'innovation).

II.A. Technologies retenues

Les technologies citées ci-après sont considérées comme devant être **maîtrisées** : elles sont grandement utilisées au sein du projet et doivent donc faire l'objet d'une certaine rigueur. Chacune développe des avantages qui ont fait qu'elles se sont retrouvées choisies pour ce projet, mais il est évident que ce n'est pas une limitation **stricte**, d'autres peuvent être utilisées pour étendre les fonctionnalités ou répondre à certaines problématiques bien spécifiques, néanmoins les technologies citées ici seront préférées à tout autre, tant que cela reste productif.

II.A.1. (X)HTML

La notation (X)HTML est souvent utilisée pour parler indifféremment de deux langages différents, mais similaires : le **HTML** (HyperText Markup Language, basé sur SGML) et le **XHTML** (eXtensible HyperText Markup Language, basé sur XML). Ce sont deux langages à balises qui permettent de décrire le contenu d'un document (le fond comme la forme, bien que ce ne soit que le premier qui nous intéresse ici). Ce n'est pas du *code source* à proprement parler, vu qu'il ne sera pas exécuté : le (X)HTML est un langage de **description**.

Le HTML est plus permissif que le XHTML, entre-autres :

- certaines balises n'ont pas besoin d'être fermées (... au lieu de ...)
- les balises sans contenu n'ont pas besoin d'être auto-fermantes (
 au lieu de
)
- les balises peuvent se croiser (.........)

Le XHTML prend les même contraintes que le **XML**, ce qui l'oblige à fermer toutes ses balises (incluant l'auto-fermeture) et à décrire une **structure arborescente** (des balises dans des balises, interdisant les croisements). C'est donc le XHTML qui est utilisé dans ce projet, en particulier car il oblige à **expliciter** (rien n'est laissé au hasard). De plus, c'est la tendance naturelle à l'heure actuelle (le HTML ayant laissé des différences d'interprétation entres les différents navigateurs, impliquant des rendus différents, le XHTML permet de ranger tout ça grâce à des règles strictes et claires que tout le monde tente aujourd'hui de satisfaire).

II.A.2. CSS

Les feuilles de styles CSS (Cascaded Style Sheet) sont une évolution apportée pour répondre à un soucis de **simplification** et d'**unification** : anciennement dans la description (X)HTML, le rendu se mélangeait à la structure des données (ont définissait le rôle d'un contenu, comme une image ou un paragraphe, en même temps que son affichage, les bordures et couleurs par exemple). L'extraction du **rendu** dans un fichier à part (feuille de styles CSS) permet de se concentrer sur la **structure** du document au niveau du (X)HTML. Cela apporte notamment plusieurs avantages majeurs comme :

- **la centralisation** : possibilité de définir un style pour un ensemble de balises (inutile de répéter le style pour chacune d'entre elles)

- **l'héritage** : possibilité de définir des styles dépendants des balises parentes (notamment avec la propriété *inherited*) et de définir des balises génériques et spécialisées (redéfinissant certains paramètres des balises génériques)

- **la personnalisation** : possibilité d'adapter le style en fonction de l'utilisateur (différents médias -PC, téléphone, TV, braille, ...- et différents rendus -choisit par l'utilisateur-)

- **la clarté** : on ne se soucis plus de savoir quel style appliquer au sein de la description (X)HTML, on donne un rôle clair au contenu via un balise spécifique (lien URL, paragraphe, classes de balises, ...) et le style est clairement définit dans une feuille de styles (couleur, largeur, bordures, ...) en fonction de son emplacement, de sa classe, ...

II.A.3. Javascript/AJAX

Le JavaScript (à ne pas confondre avec Java) est un langage de programmation (au même titre que le C/C++, Java, Python, Camel et bien d'autres). Il fait partie de la catégorie des langages **interprétés** (non compilés). Il s'intègre à même le code (X)HTML ou via l'import d'un fichier JavaScript (JS). Il sert notamment à apporter un certain **dynamisme** aux pages Web, de façon à offrir une interaction en temps réel avec l'utilisateur (plutôt que d'envoyer une requête à un serveur et d'attendre une réponse qui peut être calculée localement) ou de d'éviter la génération de pages majoritairement identiques (en ne modifiant que la partie dynamique).

En particulier, son utilisation avec **AJAX** permet de réduire le temps de chargement des pages en réduisant les communications client-serveur : une requête envoyée à un serveur génère une page Web complète, généralement avec une **partie commune** (les menus, entêtes et pieds de pages, styles, ...) et une **partie dynamique** (le contenu demandé par l'utilisateur) qui doit ensuite être envoyée complète au client et traitée dans son intégralité par le navigateur. L'utilisation du JavaScript selon les principes AJAX permet de se limiter à la génération des parties dynamiques, économisant ainsi les ressources nécessaires (serveur, client et bande passante).

Il convient cependant de noter que, du fait que ce code soit interprété par le navigateur (côté client) et donc non contrôlé par le serveur, c'est une **mauvaise** source de contrôle des données : toute requête envoyée par un client peut être générée par le code JavaScript ou par un tout autre moyen, le contrôle des données doit s'effectuer avant tout **côté serveur** pour éviter les problèmes. De plus le JavaScript peut être **désactivé** sur certains navigateurs, il faut donc pouvoir s'en passer pour les fonctionnalités de base. Malgré ces contraintes, le JavaScript n'en reste pas moins un langage fort attrayant pour améliorer l'interface d'un site Web, tout en restant relativement léger (comparé à d'autres solutions comme le Flash).

AJAX, contrairement à une idée répandue, n'est pas une évolution du JavaScript, mais un **ensemble de techniques** alliant différents langages :

- JavaScript et DOM pour les interactions homme-machine (interaction avec la page Web elle-même)
- (X)HTML et CSS pour la structure et le rendu de la page (contenu et rendu)
- XML, JSON, ...

L'idée est d'obtenir une interface Web adaptée à l'utilisateur, tout en évitant les écueils générés par les différentes implémentations des navigateurs et les failles de sécurité classiques, en utilisant des techniques d'implémentation spécifiques.

II.A.4. Java

Le langage Java est un langage de POO (Programmation Orienté Objet), au même titre que C++, Python, Javascript, ... Il est cependant entre deux paradigmes : la **compilation**, visant l'exécution rapide mais dépendante du matériel, et l'**interprétation**, visant le caractère multi-plateformes mais au détriment de la vitesse d'exécution.

Les programmes Java sont **semi-compilés** : la compilation à proprement dite se fait, non pas vers un langage machine, mais vers un langage similaire à l'assembleur (Bytecode), qui ne sera donc pas exécuté par le processeur mais par un programme à part, appelé *machine virtuelle*. C'est la machine virtuelle qui est **compilée** pour l'ordinateur sur lequel tourne le programme. De ce fait, tout ceux qui ont la JVM (Java Virtual Machine) installée, peuvent exécuter n'importe quel programme Java, le Bytecode étant le même pour tout le monde (on dispose d'un programme multi-plateformes). De plus, le Bytecode étant une suite de commandes simples, son exécution se fait relativement vite (même si on est loin de la vitesse processeur).

Un autre point important des programmes Java, ou plutôt de la machine virtuelle, c'est la **sécurité** : tout ce qui est exécuté dans une machine virtuelle reste dans cette machine virtuelle, ou plus simplement le programme s'exécute dans un environnement cloisonné, évitant les interactions non souhaitées avec des programmes trop "intrusifs". Cette sécurité est ce qui fait, après le caractère multi-plateformes, la renommée de Java, bien que cela implique le côté *lourd et lent* que l'on rattache à ce genre d'applications. Néanmoins, ce genre de problèmes peut se retrouver grandement réduit par une programmation **bien structurée** et une **configuration précise** de la JVM.

Plutôt qu'un choix, l'utilisation du langage Java est une conséquence de l'utilisation des JSP. Il n'est cependant en rien limitant car offre de nombreuses fonctionnalités de manière simple (inutile de descendre très bas niveau) et native (pas besoin de télécharger et installer des librairies spécialisées pour un grand nombre de fonctionnalités).

II.A.5. JSP/JSTL

Les JSP (Java Server Pages) sont un équivalent des pages PHP (PHP:Hypertext Preprocessor), au sens ou ce sont généralement des programmes exécutés **côté serveur** pour générer les pages Web de manière **dynamique**. Cependant plusieurs différences sont à noter :

- les JSP sont en **Java** (PHP est un langage à part entière)
- les JSP sont *de préférence* **séparées du code (X)HTML** (contrairement au code PHP intégré au (X)HTML ou vice-versa)
- les JSP sont **compilées** (PHP est interprété)

Le premier point mis à part, ces différences ont une importance cruciale qui ont prévalues sur le choix des JSP et au lieu du PHP (plus rôdé et répandu). En particulier, la JSTL (JSP Standard Tag Libraries) est une **bibliothèque de balises** utilisables dans un contexte JSP, permettant de séparer **intégralement** le code Java des JSP de la description (X)HTML, tout en offrant des fonctionnalités classiques de programmation, telles que les boucles et expressions conditionnelles, au sein de la description (X)HTML (en restant compatible avec XML). Cela permet, tout comme la séparation structure/rendu avec (X)HTML/CSS, de séparer description et logique (la logique étant dans les pages JSP).

Le dernier point offre un critère intéressant pour les projets d'envergure : le PHP étant interprété, il est naturellement plus **lent** qu'un code compilé, ce qui peut se faire sentir sur les projets de grosses tailles ou les serveurs à forte charge (génération de pages longues ou nombreuses). A contrario, le JSP est compilé (ou semi-compilé pour les plus pointilleux), car c'est du code Java qui est exécuté, le rendant plus intéressant pour ces cas là.

L'utilisation de JSP offre donc une implémentation de la "logique métier" (opérations spécialisées, maîtrisées par le serveur) séparée du rendu (affichage chez l'utilisateur), ce qui permet de **paralléliser** les tâches plus simplement. L'utilisation de JSTL est notamment un point clé (par l'ajout de nouvelles balises) permettant cette séparation, inexistante en PHP.

II.B. Technologies non retenues

Les technologies suivantes ne sont pas *rejetées*, il est possible de les utiliser au sein du projet, mais à la seule condition que les technologies citées ci-dessus ne soient pas en mesure de répondre **efficacement** à nos besoins. Pour chacune de ces technologies (ou pour tout autres), si l'envie s'en fait sentir, il convient de bien étudier le **problème** et ses **alternatives** avant de mettre en place un nouveau domaine à maîtriser au sein du projet.

II.B.1. PHP

Comme décrit plus haut, il peut être considéré comme équivalent aux JSP. Le PHP a déjà fait ses preuves et se montre très efficace pour l'implémentation de serveurs Web. La principale différence étant le fait qu'on puisse avoir un code plus **propre** (bien séparer la logique des données) et une exécution plus **rapide** (compilations VS interprétation) avec les JSP. On peut aussi rajouter le côté "sûr" de la machine virtuelle Java, mais cela se limite au côté serveur (pas d'influence sur la communication client-serveur, qui reste la principale source de failles de sécurité).

II.B.2. ASP

Tout comme le PHP est mis en face du JSP, ASP peut-être confronté à JavaScript. Une seule de ses contraintes le met hors-courses pour ce projet : ASP **nécessite** d'utiliser une plateforme IIS (Microsoft), autrement dit il est loin d'être multi-plateformes, et son caractère **propriétaire** ne laisse rien présager de bons pour un projet ou les finances tendent à être nulles.

II.B.3. C++

Comparé au Java, le C++ est bien plus **difficile** à prendre en main, en particulier parce que c'est du langage C auquel s'est vu rajouté la couche objet que l'on trouve dans toute POO. A titre de comparaison, le Java est du C++ auquel on aurait enlevé la sous-couche C, ce qui le limite à la POO. Non pas que ce soit forcément mieux de faire en Java qu'en C++, mais le Java est **naturellement** multi-plateformes, alors que le C++ nécessite d'utiliser des librairies spécifiques (telles que GTK) ou de mettre en place des structures complexes pour prendre en charge les différents environnement (directives préprocesseur).

De manière plus globale, le C++ offre plus de **libertés** de programmation (on peut descendre plus bas niveau) et de meilleures **performances** (entièrement compilé) mais au détriment du côté **multi-plateformes** (nécessite un codage ou des librairies particulières) et avec une prise en main plus **longue et hétérogène**. Étant donné le caractère haut niveau de notre projet, les contraintes imposées par ce langage ne semblent pas nécessaires, d'où la préférence de Java, renforcée par l'utilisation des JSP.

II.B.4. Flash

De nombreux points noirs pèsent sur le dos de la technologie Flash, plus ou moins subjectifs :

- il est **lourd** (comparé à la charge mémoire prise pour une page sans Flash de même taille)

- il est **propriétaire** (sous licence Macromedia)
- il n'est pas **multi-plateformes** (tourne difficilement sur Linux jusqu'à la version 8)

On lui reconnaît cependant les avantages suivants :

- **fortement** utilisé (montrant une certaine maîtrise, donc potentiellement facile à prendre en main)
- permet la gestion de **flux vidéos** en streaming
- permet de **générer des animations** via des scripts (ActionScript)

Pour notre projet, les inconvénients ayant plus de poids que les avantages, le Flash n'est pas préféré.

II.B.5. Autres langages

De manière générale, les scripts (Python, Bash, Perl, ...) permettent d'automatiser de nombreuses tâches, et chacun développe ses propres avantages et inconvénient. Nous notons cependant ceci :

- côté client, il est recommandé de garder un **environnement simple**, de façon à ne pas surcharger le poste de l'utilisateur inutilement
- côté serveur, le Java tient une part importante de la logique et semble **suffisamment complet** pour répondre à des besoins multiples et variés

De ce fait, l'utilisation de scripts supplémentaires, et de manière générale d'autres langages, est à éviter, à moins que cette utilisation soit **grandement justifiée** par les besoins et les contraintes techniques.

III. Solution proposée

III.A. Choix des technologies

Les technologies retenues sont choisies principalement pour deux caractéristiques :

- leur **généricité**
- leur **avenir** pressenti

En particulier, (X)HTML, CSS et Javascript sont des standards **largement utilisés** aujourd'hui, et devenus **incontournables** dans les projets de développement Web (bien que de nouvelles technologies sortent, notamment il est possible de générer le (X)HTML avec les JSP et le Javascript avec du Ruby). Il va de soi que ces technologies sont les plus utilisées et le resteront encore longtemps. Java est, quant à lui, bien plus jeune vis-à-vis du développement Web. JSP/JSTL est encore en train d'évoluer (nous n'en sommes qu'aux **premières versions**) et va probablement s'enrichir avec le temps. Le choix de ces derniers tient bien plus du fait qu'ils ont le vent en poupe et qu'une certaine maîtrise est déjà acquise au sein du corps enseignant de l'École du Fansub.

III.B. Environnement de développement

Les outils utilisés dans le projet sont les suivants :

- l'EDI **NetBeans**, incluant le plugin Java Web
- le serveur **Apache Tomcat**
- le gestionnaire de versions **Mercurial**

Le choix de l'EDI (Environnement de Développement Intégré) est, comme la plupart des projets de grande taille sûrement, basé sur **Eclipse et NetBeans**. Les deux étant relativement similaires (bien qu'il semble que NetBeans soit plus simple à prendre en main), le responsable du projet est bien plus **expérimenté** sur NetBeans, d'où le choix.

Le serveur Apache Tomcat semble être un bon compromis entre **légèreté et fonctionnalités**, en effet il semble que ce soit le plus à même de prendre en charge un site Web professionnel (pas comme Jetty, très léger mais apparemment peu efficace face à une montée en charge), tout en évitant de nécessiter une machine dédiée relativement puissante (contrairement à GlassFish, impossible à faire tourner sur un PC personnel peu récent). Étant donné que le projet Fansub Site Project nécessite une **bonne stabilité** pour les tests et que les développeurs travaillent sur **PC personnel** principalement, ce choix semble des plus judicieux.

Pour la gestion de version, indispensable pour un projet partagé comme celui-ci, le choix devait d'abord être fait entre les gestionnaires **centralisés** (avec serveur principal) et **distribués** (chaque dépôt est aussi important qu'un autre). Un gestionnaire centralisé (CVS, Subversion, ...) aurait été plus problématique, principalement à cause de la **mise en place du serveur** en question, nécessitant d'assurer une certaine autonomie (utilisation d'un serveur dédié) et difficile à utiliser avec une **connexion peu disponible** ou pour faire des **développements expérimentaux** (typique en

innovation). La gestion distribuée en revanche n'apporte pas plus de contraintes que de savoir installer son gestionnaire de version (pas de différence à faire entre client et serveur) et plusieurs sites proposent de mettre en place gratuitement des dépôts (http://mercurial.intuxication.org dans notre cas). Pour le choix entre les différents gestionnaires distribués (Mercurial, Git, Bazaar, …), NetBeans **intègre de base** Mercurial, ce qui a directement orienté le choix.

III.C. Définitions

Avant de définir les fonctionnalités à implémenter, il nous faut connaître les **caractéristiques** d'une équipe de Fansub (III.C.1. Les métiers au sein d'une équipe de Fansub et III.C.2. Les statuts) et **l'organisation globale** qui doit être mise en place pour cette équipe (III.C.3. L'organisation d'une équipe de Fansub). Pour cela, nous définissons chacun des termes utilisés dans la suite de ce document. Nous commencerons ici avec la plus importante :

Une **équipe de Fansub** est un ensemble de personnes s'activant dans le but de fournir des œuvres audiovisuelles étrangères (animés, dramas, films, …) de façon à ce qu'elles soient **accessibles** à un certain public. Nous partons du principe que cela se passe de manière **virtuelle** (œuvres récupérées depuis Internet ou recopiées sur PC depuis un support physique) et de manière **totalement bénévole** (aucune rémunération ne doit être demandée pour accéder au contenu du site, pour obtenir les productions de l'équipe ou pour l'intégrer). Bien entendu, cela n'exclut pas les **dons et ventes annexes** (goodies, publicité, ...), et les contraintes économiques de l'équipe sont à considérer (notamment le paiement des serveurs). Le côté virtuel implique généralement que les membres d'une équipe de Fansub sont des internautes, bien que cela ne soit pas obligatoire. Étant donné que l'objectif de ce projet est de développer un site Web, nous faisons tout de même cette simplification (nous ne considérons donc que les internautes).

III.C.1. Rôles, degrés et classes : les métiers au sein d'une équipe de Fansub

Bien que nous pouvons avoir des grands classiques, comme les RAW hunter, traducteurs, checkeurs, … ces rôles peuvent avoir des **nominations différentes** selon les équipes et même être plus ou moins différents sur la **définition** du rôle en lui-même (certaines équipes auraient des traducteurs EN-FR, d'autres JP-FR, certaines auraient les deux et il faudrait pouvoir faire la différence entre chacun dans ce dernier cas). Par conséquent les rôles disponibles sont définis par l'équipe elle-même.

Bien que ces rôles ne soient alors pas connus à l'avance, un rôle ne définit qu'une fonction que joue un membre dans l'équipe, une autre caractéristique utile à connaître est le **niveau de compétences** que peut garantir le membre dans un rôle donné, que nous nommons **degré**. Ce degré, associé à un rôle particulier, affiche donc les compétences globales du membre dans ce rôle, et suit les descriptions suivantes :

Degrés	Officiels	Officieux
Confirmé	certifié	autodidacte
Apprenti	apprenti	
Testé	essai	

Le degré ***essai*** définit un niveau inconnu, le membre est alors mis à l'épreuve par l'équipe. C'est le degré par défaut donné lors de **l'acquisition d'un nouveau rôle**. Par la suite, l'équipe peut choisir d'accepter officiellement le membre pour ce rôle particulier, auquel cas un degré **officiel** lui est donné, autrement le rôle est considéré comme une simple « curiosité » de la part du membre, assignant donc le degré officieux ***autodidacte*** (il souhaite s'y former, mais on n'attend pas de lui qu'il s'en serve de manière consistante au sein de l'équipe). Si un degré officiel est donné, il est alors fonction **des compétences** du membres : si elles ne sont pas suffisantes pour qu'il soit autonome, il prend un degré ***apprenti***, autrement il obtient un degré ***certifié***. Bien entendu, il est possible d'assigner un degré officiel à la place d'un degré officieux, tout dépend de la **reconnaissance de l'équipe**. A contrario, un degré officiel ne peut pas être remplacé par un degré officieux : nous prenons le parti que quand on a prouvé de quoi on est capable, il n'y a pas de raison de le remettre en doute, on ne peut que s'améliorer.

À ce stade, nous connaissons le rôle du membre et son niveau global de compétences, mais certaines tâches nécessites d'avoir un **niveau précis** (être un *apprenti* ayant certains acquis, ou un *certifié* maitrisant des points spécifiques non requis à la base). Cette différence est donnée par une **classe**. Cette classe est associée au degré pour montrer le niveau précis du membre dans le rôle considéré. Les classes possibles sont **définies par l'équipe et ordonnées**. Cet ordre est arbitrairement fixé par l'équipe, l'idée étant de connaître les classes qui sont de niveau égales ou supérieur aux autres. Ces classes ne sont valables que pour les degrés **exclusivement officiaux** (*essai* et *autodidacte* n'ont donc pas de classes) car elles montrent ce que l'équipe reconnaît comme maîtrisé par le membre. Voici des exemples de classes ordonnées pour le rôle de traducteur JP-FR (les classes de même niveau sont dans une même case, séparés par des virgules) :

Degré	Classe
certifié	Japonais et accents
	Japonais courant
	Japonais théorique
apprenti	Phrases avancées
	Conjugaison avancée, Grammaire avancée
	Phrases de base
	Conjugaison de base, Grammaire de base
	kana
	hiragana, katakana

Ainsi on peut interpréter cette liste comme : un apprenti commence par apprendre **les hiragana ou les katakana**. Une fois l'un des deux fait, il apprend le second et passe donc en apprenti **kana** (hiragana+katakana). Les kanas maîtrisés il peut se mettre à la **conjugaison ou la grammaire de base**. Après avoir appris l'un des deux il passe au second pour maîtriser les **phrases de bases**. Le processus se répète pour la partie avancée, et une fois l'apprentissage des **phrases avancées** terminé le membre passe en degré **certifié** pour le **japonais théorique**. L'évolution du membre se fera alors en fonction de son expérience pratique : à force d'écouter et de traduire depuis des vidéos (animés, dramas, …) il obtiendra le niveau certifié pour le **japonais courant**. La traduction de passages particulièrement difficiles l'amènera à maîtriser les différents accents de la langue japonaise, lui donnant accès au plus haut niveau de certification : **japonais et accents**.

Même si les rôles et classes sont définis par l'équipe, un rôle spécifique est tout de même fixé : **chef d'équipe**. Ce rôle donne **tous les droits** sur le site, il est semblable au classique « administrateur » et, lorsqu'on parle du « choix de l'équipe », c'est aux chefs d'équipe de s'assurer que ce choix est correctement appliqué. Ce rôle existe principalement pour **régler les problèmes de droits** relatifs à l'évolution du site (les droits étant définis selon les rôles de chacun, il faut un rôle spécial pour régler les litiges et problèmes techniques, comme l'acceptation des premiers membres, et la configuration des droits de plus haut niveaux). Ce rôle ne dispose d'**aucun degré** et donc d'**aucune classe**.

La modification d'un degré ou d'une classe d'un membre ne peut se faire que par :

- un **chef d'équipe**
- un autre membre de **même rôle**, de **degré *certifié*** et de **classe strictement supérieure**
  - ex : aucun *apprenti* ne peut modifié de degré, mais un traducteur *certifié* « Japonais courant » peut faire monter en grade un traducteur *certifié* « Japonais théorique ».

III.C.2. Les statuts : cycle de vie d'un membre de l'équipe

Nous donnons ici une définition exacte des différents **statuts**, termes employés pour qualifier les personnes qui se connectent au site :

- un **internaute** est une personne se connectant à internet pour accéder au site de l'équipe de Fansub
- un **connecté** est un internaute inscrit sur le site
- un **candidat** est un connecté souhaitant intégrer l'équipe de Fansub pour des rôles particuliers
- un **membre** est un connecté ayant au moins un rôle de degré officiel supérieur à *essai*
- un **retraité** est un connecté ayant déjà été membre de l'équipe de Fansub

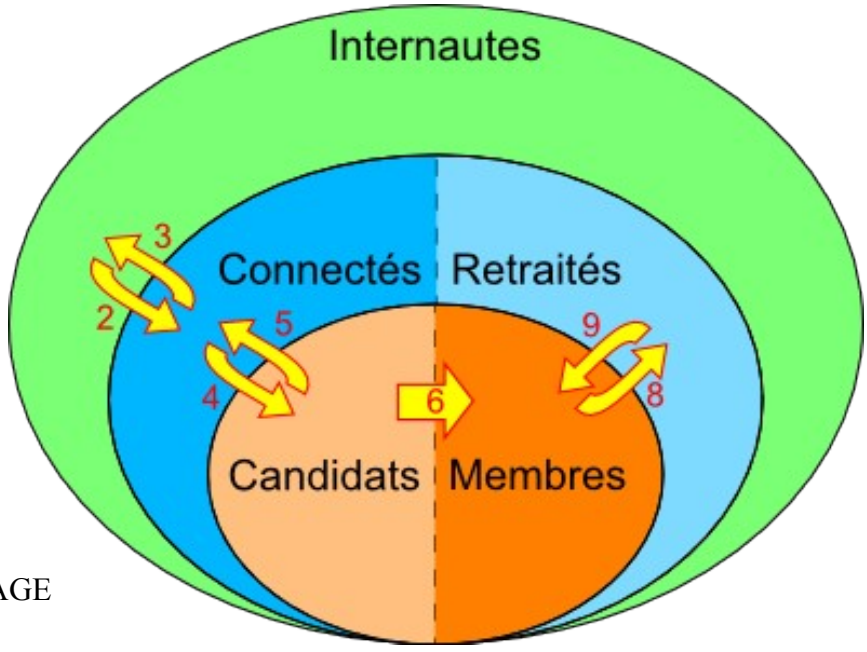
Les statuts que peuvent avoir un internaute suivent un cycle de vie précis, en voici la description :

1. un **internaute** accède aux parties publiques du site en toute liberté (si tant est qu'il détient une connexion Internet),
2. un **internaute** s'inscrit ou se connecte sur le site de l'équipe, disposant alors d'un **compte** qui lui est dédié et contenant certaines informations, il obtient alors le statut de **connecté** ; aucune information du compte n'est **techniquement obligatoire** (si rien n'est fournit, le compte est créé quand même), cependant certaines informations sont obligatoires pour

poser une candidature (comme une adresse mail vérifiée pour pouvoir le joindre par exemple, ces informations obligatoires sont choisies par l'équipe),

3. un **connecté** peut choisir de supprimer son compte, toutes les informations de son compte sont alors supprimées et son statut de connecté lui est retiré, les traces de son passage (comme les posts sur le forum) sont alors assignés à un statut spécial *ancien visiteur* ; notons que **seul un compte de statut connecté** peut être supprimé, ce n'est pas le cas des autres statuts,
4. un **connecté** pose une candidature (sous condition qu'il ait fournit les informations obligatoires sur son compte) pour **un ou plusieurs rôles** sur lesquels il souhaite travailler au sein de l'équipe, il prend alors le statut de **candidat** et les rôles qu'il a demandé lui sont alors assignés avec un degré *essai*,
5. un **candidat** peut recevoir des degrés *autodidacte* si les rôles qu'il demande ne sont pas souhaités de la part de l'équipe, ou s'il ne montre pas de réel apprentissage sur le sujet ; si tous ses rôles passent en *autodidacte*, ses rôles et son statut de candidat lui sont retirés (il repasse donc au statut de **connecté**),
6. un **candidat** peut recevoir un degré officiel supérieur à *essai* pour au moins un rôle, son statut de candidat est alors remplacé par celui de **membre**,
7. un **membre** participe aux activités de l'équipe en :
   - offrant ses services à l'équipe dans les **rôles** qui lui sont donnés, les degrés associés peuvent augmenter selon des critères définis par l'équipe,
   - prenant de **nouveaux rôles** (ajout uniquement) qui suivent alors les même règles que n'importe quel rôle (départ à *essai* puis évolution),
8. un **membre** peut choisir d'arrêter ses activités, auquel cas le membre prend le statut de **retraité**,
9. un **retraité** peut reprendre son titre de **membre**, de façon à participer de nouveau aux activités de l'équipe, avec une mise à jour des rôles si nécessaires (suivant les même règles que précédemment) ; un retraité dispose des même droits qu'un visiteur, à l'exception qu'il n'a pas besoin de poser une candidature pour rejoindre l'équipe.

Une remarque importante à faire sur les statuts de **candidat** et **retraités** : un candidat est un **membre limité**, il dispose donc des même fonctionnalités qu'un membre sauf la possibilité de devenir retraité. De même un retraité est un **connecté évolué**, il dispose de toutes les fonctionnalités d'un connecté mais n'a pas besoin de poser une candidature pour intégrer l'équipe. Voici un schéma illustrant ce cycle de vie (les nombres de la figure correspondent aux étapes du cycle) :



On peut constater l'existence de 3 blocs :

1. **tous** : internautes
2. les **internes** : connectés/retraités
3. l'**équipe** : candidats/membres

Le point important est sur les transitions entre les **blocs 2 et 3** : un connecté ne peut **intégrer l'équipe** qu'en devenant candidat, et un candidat ne peut **quitter l'équipe** qu'en redevenant un simple connecté (flèches 4-5). De même un membre ne peut **quitter l'équipe** qu'en devenant retraité, alors qu'un retraité ne peut **réintégrer l'équipe** qu'au statut de membre (flèches 8-9). On peut ainsi interpréter sur la figure (pour les blocs 2 & 3) que ceux qui sont à droite sont les seuls à avoir déjà été **reconnus** comme membres de l'équipe, ceux étant à gauche étant uniquement de **futurs** membres potentiels.

La flèches 6 indique ce **changement de reconnaissance** : un candidat devient membre lorsqu'au moins un **rôle officiel** lui est donné au sein de l'équipe (un degré officiel parmi les rôles qu'il a demandé). Les flèches 2-3 indique la possibilité qu'ont les internautes de **s'inscrire** et les connectés de se **désinscrire**. Les étapes 1 et 7 du cycle de vie n'entraine aucun changement de statut, ils ne sont donc pas sur la figure.

III.C.3. Les rapports : organisation d'une équipe de Fansub

L'organisation d'une équipe est très peu réglementée, pour la simple raison qu'elle prend en compte la **travail à effectuer** (partie globalement similaire quelque soit l'équipe) associé aux **disponibilités des membres** et aux **priorités de l'équipe**. Ces deux derniers points introduisent une part fortement **subjective** et particulièrement **irrégulière** : il n'y a pas de règles générales pouvant définir n'importe quelle organisation, mais plutôt des **modèles d'organisations** particuliers, eux-même nécessitant certaines adaptations pour être réellement applicables, ils ont donc plutôt un rôle de classes pour catégoriser les différentes organisations possibles.

Pour éviter de restreindre toute équipe de Fansub utilisant notre site à une organisation particulière, nous préférons mettre en place une **structure organisatrice**. L'objectif est donc non pas d'organiser le travail des membres, mais de leur donner les outils leur permettant de s'organiser **par eux-même**, sans mettre en place d'organisation spécifique, simplement en constatant les faits et en choisissant quelle est la prochaine étape à effectuer. Cela nécessite notamment une **communication accrue**, où chaque membre est mis au courant dès que nécessaire des changements qui le concerne. Il convient cependant de ne pas tomber dans de la **sur-information** : trop de messages à traiter entraine une baisse du rendement général (on passe plus de temps à se mettre au courant, et donc moins à travailler) et une désinformation de l'équipe (les nouvelles informations demande un certain temps d'acquisition plus ou moins long selon leur nombre et leur complexité, chacun ayant alors sa propre version selon ce qu'il en a compris et retenu pendant ce laps de temps).

Le système mis en place doit donc pouvoir générer **automatiquement** tout un ensemble de messages, que nous appellerons **rapports**. Ces rapports ont pour objectif de centraliser les informations et d'informer les membres sur ce qui les **intéressent**, et uniquement cela (pas de message « génériques » où chacun doit chercher ce qui l'intéresse). Nous distinguons d'ailleurs deux types de rapport :

- le **rapport thématique** est un rapport sur un sujet précis (une tâche, un projet, les informations d'un membre, …) et accessible depuis l'objet du rapport (la tâche, le projet, le membre, …), son contenu change donc en fonction du sujet traité,
- un **rapport personnel** est un rapport adapté à un membre particulier, listant tous les points dont il doit prendre connaissance (qui ont changé depuis le dernier rapport), son contenu change donc en fonction du lecteur.

Il peut sembler important de laisser certains membres écrire des **rapports à la main**, de façon à pouvoir générer des rapports qui ne seraient pas pris en compte par le système mais important pour l'équipe. Une telle génération est néanmoins **à bannir** : l'utilité d'un rapport est de donner une description **exacte et formatée** des informations à connaître, impliquant un **format fixe et explicite**. Aussi l'écriture « manuelle » de rapports empêche la vérification de ces contrainte : une base de données n'oublie pas, un algorithme ne change pas, un être humain oui. Pour ces raisons de **dynamisme**, il ne doit pas être possible de générer un rapport à la main. Il faudra privilégier les conversations pour cela, dont c'est justement l'objectif.

III.C.4. Les conversations : discussions et débats

Le Fansub étant une activité humaine comme une autre, il en découle que certaines choses ne puissent être gérées de manière automatique et nécessitent la participation de plusieurs personnes (brainstorming, débats, remises en causes et prises de décisions, ...). Cela nécessite que chaque personne concernée puisse s'exprimer librement et de façon aisée sur le sujet. Les conversations peuvent être de deux ordres :

- conversations privées
  - création sur simple demande
  - accessible sur invitation
  - retrait de participants volontaire
  - suppression de la conversation lorsque tous les participants l'ont quittée
- conversations publiques
  - création sur demande d'un membre autorisé
  - accessible pour tout le monde (accès des internautes discutable)
  - retrait de participants volontaire ou sur demande d'un membre autorisé (ban possible)
  - la suppression de la conversation se fait sur demande d'un membre autorisé

Bien entendu, une personne peut participer à plusieurs discussions simultanément, elle doit donc pouvoir accéder à chacune facilement et aller de l'une à l'autre rapidement. En mettant de côté les solutions techniques qui permettrait de répondre à la seconde contraintes (cadres, onglets, fenêtres multiples), l'accès aux conversations doit se faire simplement : les conversations privées depuis les invitations reçues, les conversations publiques depuis une page d'accès (disposant de plusieurs sections).

TODO : quel type de rangement ? Recopier les sections du système de fichier ? Aller jusqu'à traiter les conversations comme des fichiers ? (étudier les droits d'accès, gestion de version, sauvegarde des conversations, ...)

III.D. Fonctionnalités

Les fonctionnalités à développer ne se limite pas à ce simple cahier des charges, néanmoins celles définies ici sont à implémenter **en priorité**, pour la simple raison que c'est la base nécessaire pour mettre en place un site orienté sur les besoins d'une équipe de Fansub. Les fonctionnalités peuvent (et même devraient) être **agrémentées** au fur et à mesure du développement.

III.D.1. Gestion des fichiers

Les systèmes de fichiers classiques (FAT16/32, NTFS, ext2/3, ...), bien que permettant de faire beaucoup de choses, disposes de certains **inconvénients**. Nous listons tout d'abord les différents points que nous souhaiterions voir corrigés :

- la **gestion de l'espace**, où les systèmes classiques se contentent généralement d'écrire la suite de bits sans trop se soucier de son importance. En particulier, il est possible de **compresser des données** rarement utilisées ou de **supprimer des fichiers inutilisés**, mais cela se solde généralement par : une demande explicite (pas d'automatisme permettant une certaine progressivité), un temps de traitement long (conséquence directe du point précédent appliqué à un grand nombre de fichiers), une perte d'espace (fichiers temporaires non supprimés car mal placés et indétectables sans outils spécialisés),

- la **gestion des droits**, elle se retrouve limitée à une régulation **réduite** (lecture/écriture, utilisateur/groupe) dont les critères réunissent des composantes qui aujourd'hui on vocation à être **dissociées** (écriture = création + modification + suppression + déplacement + renommage) ou dont les critères sont **sous-exploités** (groupes d'utilisateurs uniquement, pas de hiérarchie de groupe ou de récursivité), ce qui oblige de faire des constructions complexes (voire de considérer une autre approche) pour des cas spécifiques,

- le **typage des fichiers**, reconnu soit par son contenu (un code au sein du fichier permet de reconnaître son type), soit par son nom (généralement l'extension du fichier). Le premier cas implique que le type du fichier **dépend de son contenu**, et donc peut être modifié en changeant le contenu du fichier, ce qui est sémantiquement peu crédible (un fichier vidéo n'a aucune raison d'avoir un contenu textuel et inversement), à l'exception de fichiers contenant plusieurs types de données (qui au final se contentent d'être des fichiers dits « binaires », donc sans type particulier si ce n'est celui qu'on veut bien leur donner). Le second cas implique que le type puisse être **modifié via le nom du fichier**, ce qui est encore pire que le cas précédent (ce n'est pas parce qu'il est marqué « Compter avec les animaux » sur sa couverture que ce livre est un livre de comptes). En particulier le type d'un fichier est fortement lié à l'utilisation qu'on en fait (un fichier vidéo, qu'il soit d'extension AVI, MKV ou d'autre extension vidéo, qu'il soit compressé en H264 ou via d'autres codecs, est généralement lisible sur VLC ou Media Player Classic) et ne change pas de toute sa durée de vie (un fichier vidéo ne se retrouve pas du jour au lendemain à être un fichier texte),

- la **gestion des versions**, en effet deux versions différentes d'un même fichier sont reconnues par le système comme **deux fichiers** différents, alors que des liens évidents sont fait du point de vue utilisateur, nous obligeant à utiliser des logiciels de gestions de versions pour y palier. En particulier il est possible de déplacer les versions **indépendamment** les unes des autres, entrainant notamment de retrouver certaines versions du fichier à différents endroits, et ainsi de ne plus s'y retrouver et de risquer de **perdre** ou d'**effacer** des versions utiles, ou d'utiliser des versions **obsolètes**. De plus l'information contenue dans ces fichiers est **redondante** : de petites modifications dans un fichier même volumineux nécessite de créer un nouveau fichier contenant une grande part d'informations déjà connue, ce qui est loin d'être économique.

Pour notre propre système de fichier, nous distingueront notamment les notions suivantes :

- un **fichier** est un élément atomique, au sens « ne pouvant être divisé sans perdre de l'information »
  - il est structuré par son **type**, qui est la sémantique associée au fichier (vidéo, texte, ASS, … un type « indéfini » pour les fichiers complexes peut être utile),
  - son **identifiant** permet de l'identifier de manière unique (code généré automatiquement et définitivement),
  - son **nom** permet de l'identifier de manière rapide et sûre (chaîne de caractères définie par l'utilisateur, mais unique, la différence avec l'identifiant étant son caractère modifiable),
  - son **contenu** détient **toute l'information** du fichier depuis sa création, en particulier toutes les versions du fichier peuvent être accessibles (seule la dernière est obligatoire, contenu effectif du fichier, les autres peuvent être « purgées » si elles sont jugées inutiles) ; le contenu peut être modifié **manuellement** (saisie sur le site), **automatiquement** (modification par le système) ou **indirectement** (un fichier chargé sur le site pour faire office de nouvelle version),
- une **section** est un conteneur (de fichiers et de sections), l'idée est similaire aux classiques « dossiers » à quelques détails prêts,
  - un fichier peut être dans plusieurs sections à la fois, il est important de pouvoir rassembler les fichiers sans avoir besoin d'en faire des copies inutiles (idée similaire aux liens Unix),
  - nous ne prenons pas le parti qu'une section est un **fichier** (comme le fait Unix), une section est une entité purement virtuelle qui sert à organiser les fichiers, par conséquent nous n'avons pas de droits de « lecture » ou d'« écriture » par exemple, mais plutôt de « visibilité » et de « modification du contenu ».

Bien entendu nous ne créons pas un système de fichier de toute pièce, nous utilisons le système de fichier de la **machine hôte** auquel nous ajoutons une couche logicielle, sur laquelle nous disposons les fonctionnalités qui nous intéressent. Ces fonctionnalités sont les suivantes :

- pour un fichier :
  - **créer** (version initiale vide, nom choisit, type choisit et définitif, section choisit)
  - **renommer**
  - **afficher** son contenu
  - **détailler** ses informations systèmes (identifiant, date de création, date de la dernière version, nombre de versions, nombre de versions inutiles, taille totale, taille récupérable après une purge, ...

    - **modifier** le contenu (saisie directe, chargement d'un nouveau fichier)
    - **ajouter** une version (en sauvegardant les modifications)
    - **changer** de version (désigner une ancienne version comme version effective, cela crée notamment un double de la version choisie : lors de sa création et lors de sa reprise)
    - **purger** le fichier (supprimer certaines versions, la dernière exclue)
    - **accéder** à n'importe quelle version
    - **comparer** deux versions (du même fichier)
    - **supprimer** (avec toutes ses versions, cette opération doit être de l'ordre de l'automatisme, parce que plus rien n'utilise le fichier, et non d'un choix utilisateur)
    - **lier** à une section (le rajouter dans la section)
    - **délier** d'une section (l'enlever de la section sans le supprimer pour autant)
- pour une section :
    - **créer**
    - **renommer**
    - **afficher** son contenu, notamment avec une vue arborescente et une police adaptée (permettant par exemple de repérer les fichiers inutilisés, volumineux, ayant des versions à purger, ...)
    - **déplacer**
    - **purger** (délier les fichiers et supprimer les sous-sections qu'elle contient)
    - **supprimer** (purger puis effacer la section, à condition qu'elle ne soit pas la section racine)
- plus généralement
    - définir une **section racine** (éventuellement définir des sections génériques pour offrir un rangement type)
    - effectuer une **compression progressive** pendant les baisses de charge (uniquement pour les versions rarement utilisées)
    - permettre de demander **différents types** de purges (par exemple « versions obsolètes », « fichiers inutilisés », « purge complète », … spécifier différentes possibilités et voir les avantages/inconvénients qu'elles apportent)

III.D.2. Gestion des évènements

TODO : compléter après les parties précédentes

TODO : penser à la notification aux membres pour un nouveau rapport les concernant

Des événements interviennent à tout moment, plus ou moins régulièrement et à des fréquences plus ou moins grandes. En particulier, toute fonctionnalité développée dans ce chapitre est une source importante d'évènements à prendre en considération, en particulier pour la gestion de rapports et la gestion des fichiers.

-

III.D.3. Gestion des rôles

TODO : compléter

III.D.4. Gestion des droits

Une gestion de droits efficace doit pouvoir prendre en compte **plusieurs dimensions**, de façon à répondre à des besoins divers et variés, notamment :

- **hiérarchiques** (de internaute à membre, ou gestion verticale)
- **fonctionnels** (par rôle – incluant degrés et classes – ou gestion horizontale)
- **personnels**, pour les cas particuliers (utilisateurs spécifiques, comme des « VIP », n'ayant pas besoin d'être membres mais ayant certains accès supplémentaires par exemple)

De plus, les droits eux-même (leur définition, et non leur application) peuvent être regroupés sur des thèmes spécifiques :

- fichiers
- sections
- tâches
- projets
- plus généralement chaque fonctionnalité du site

Nous prenons le parti que moins on a de droits, moins on a la possibilité de faire des bêtises, et que si des droits doivent être donnés c'est parce qu'il y a de bonnes raisons (principe sécuritaire). De ce fait, **tous les droits** doivent être **par défaut** ouverts aux **chefs d'équipe uniquement**. Un droit ne peut être ouvert à d'autres membre que par une autorisation explicite dans la configuration des droits.

TODO : compléter

III.D.5. Gestion des tâches

une tâche est une **action à effectuer**. Une **tâche atomique** est une tâche où l'opération a effectuer ne peut pas être divisée en plusieurs tâches, une **tâche composée** est un **regroupement** de tâches (appelées sous-tâches). Une tâche, qu'elle soit atomique ou composée, se définie par ses **entrée/sortie** (fichiers utilisés/créés). Il n'y a pas de contraintes de nombre, il peut y avoir zéro, un ou plusieurs éléments pour chaque entrée/sortie.

TODO : image tâche avec E/S

Une tâche atomique dispose en plus d'une **fonction** (travail à faire), qui est définie par des **contraintes** d'application (membres nécessaires, dates limites, …) et l'**opération** à effectuer. Une tâche atomique nécessitant au moins un membre est dite **manuelle**, à l'inverse une tâche atomique ne nécessitant aucun membre est dite **automatique**.

Les entrées/sorties d'une tâche atomique sont bien entendues dépendantes de la fonction à effectuer, et doivent donc être soigneusement choisies. À ce titre, une tâche automatique apporte certaines simplifications : la fonction de la tâche est une **fonctionnalité du site**, les entrées de la tâche sont donc parmi les **entrées nécessaires à cette fonctionnalité** et respectivement pour les **sorties**. A contrario, une tâche manuelle est réglée à la main : les entrées/sorties sont choisies et la fonction à réaliser est simplement décrite.

TODO : image tâche manu + auto

Une tâche composée est un regroupement de tâches (atomiques ou composées). Elle dispose d'une **description** qui décrit la fonction globale réalisée. Les entrées de la tâche composée sont les **entrées non satisfaites** de ses sous-tâches (celles qui ne sont pas liées aux sorties d'autres sous-tâches). Les sorties de la tâche composée sont **toutes les sorties** des sous-tâches. Ainsi une tâche composée est intégralement définie par ses sous-tâches, la description venant clarifier son utilité.

Le cycle de vie d'une tâche atomique est le suivant :

1. la tâche est créée et configurée (entrées/sorties et fonction, avec ses conditions), on parle alors d'une **tâche nouvelle**,
2. lorsque toutes les entrées de la tâche sont satisfaites (les fichiers existent), on parle d'une **tâche prête** (ou en pause),
3. lorsque toutes les conditions de la fonction sont satisfaites, ont considère que le travail peut débuter et on parle de **tâche ouverte** (ou en cours); si des membres sont inscrit à la tâche ils sont prévenus, si une fonctionnalité doit être utilisée elle l'est aussitôt ; si les conditions ne sont **plus vérifiées** à un moment donné, la tâche repasse à l'étape **précédente**,
4. si un problème survient durant la tâche (problème d'entrée/sortie, fonctionnalité impossible à exécuter, ...) la tâche est marquée comme **impossible**, elle nécessite alors une intervention de la part des membres de l'équipe pour régler le litige (ce qui implique une relance de la tâche, cf. étape 6),
5. une fois les sorties obtenues, on obtient une **tâche validée**,
6. une tâche validée ou marquée comme impossible peut être **relancée** (manuellement, par un membre, ou automatiquement, car au moins une entrée dispose d'une nouvelle version), la tâche repasse alors à l'étape 1 ; c'est aux membres de chaque tâche relancée de décider si le travail doit être mis à jour ou laissé en l'état (dans le cas de tâches automatiques, la tâche est refaite).

Une tâche composée a un cycle de vie qui dépend intégralement des sous-tâches qui la compose, ainsi on retrouve un format similaire mais entièrement automatisé :

1. création de la tâche → tâche prête
2. satisfaction des entrées, satisfaction des conditions (immédiate du fait de leur absence) → tâche ouverte
3. validation de toutes les sous-tâches → tâche validée
4. relance d'au moins une sous-tâche → relance de la tâche composée

Voici quelques exemples de tâches imaginables :

ID	Type	Description	Entrées	Conditions	Sorties
1	atomique manuelle	Traduction de l'épisode X	- RAW	- un traducteur	1 – traduction
2	atomique manuelle	Adaptation de l'épisode X	- tâche 1, sortie 1	- un adapteur	1 - sous-titres
3	atomique manuelle	Time de l'épisode X	- tâche 2, sortie 1 - RAW	- un timeur	1 - ASS

4	atomique automatique	Encodage de l'épisode X (fonctionnalité)	- tâche 3, sortie 1 - RAW - configuration SD - configuration HD	aucune	1 – épisode SD 2 – épisode HD
5	composée (tâches 1 à 4)	Subbing de l'épisode X	- RAW - configuration SD - configuration HD		1 – traduction 2 – sous-titres 3 – ASS 4 – épisode SD 5 – épisode HD

Notons que les entrées de la tâche composée correspondent à tout ce que nous avons besoin pour les sous-tâches qu'elle contient (les autres fichiers étant générés au sein de la tâche). Les entrées des tâches peuvent bien entendu être reçues depuis d'autres tâches, mais elles doivent pouvoir être spécifiées manuellement : ici la RAW est un fichier récupéré sur Internet, c'est aux membres à informer la tâche que le fichier « RAW » qu'elle cherche est tel fichier du site (et qui aura été précédemment transféré sur le site).

Voici la liste des fonctionnalités à implémenter pour la gestion des tâches :

- gérer des modèles de tâches
- modifier une tâche (incluant l'association de fichiers aux entrées de la tâche)
- s'inscrire à/se désinscrire d'une tâche (tâches atomiques manuelles)
- afficher l'avancement d'une tâche
- valider/relancer une tâche
- afficher le rapport d'une tâche

III.D.6. Gestion des projets

Un projet est une tâche à effectuer au sein de l'équipe. Ce terme de tâche est le même que celui développé au chapitre précédent, un projet sera d'ailleurs généralement représenté par une tâche composée, bien qu'il puisse être une tâche atomique (tout dépend de la taille du projet).

Un projet est avant tout composé des points suivants :

- un **titre**
- une **description** du projet (contenu, objectifs, …)
- une **tâche** associée au projet
- des **membres** responsables du projet (automatiquement inscrits aux tâches du projet)

Bien que ces points semblent naturels, ils imposent certaines **contraintes d'organisation** : la tâche associée au projet est **unique**, si plusieurs tâches doivent y être associé c'est soit au sein d'une **tâche composée**, soit en découpant le projet en **plusieurs projets**. Les membres responsables sont automatiquement inscrits aux tâches associées (sous-entendu les tâches **atomiques manuelles** intégrées dans la tâche associée au projet) sous condition que ces membres respectent les conditions des tâches elles-même (on n'inscrit pas un adapteur là où on n'a besoin que d'un traducteur).

Voici une liste d'exemples de projets imaginables :

- une nouvelle série à sous-titrer (animé, drama, …)
- un film ou un épisode spécial (pour contraster avec le côté « multi » de l'exemple précédent)
- modifications du site (projet de développement)
- partenariat (nécessitant des vérifications et des modifications, ajout de liens notamment)
- ...

Et enfin la liste des fonctionnalités à implémenter :

- proposer un nouveau projet (création complète du projet, incluant la définition des tâches)
- accepter/refuser un projet proposé (lancement ou non de la tâche associée)
- modifier le projet (notamment la tâche associée)
- mettre en pause/relancer un projet
- valider/abandonner un projet (lié à la validation/l'abandon de la tâche)
- afficher le rapport d'un projet

III.D.7. Gestion des conversations

TODO : revoir cette partie

Les conversations ont pour principale fonction de faire passer l'information souhaitée, aussi elles doivent le faire passer le plus simplement du monde tout en offrant un maximum de possibilités. Les fonctionnalités à implémenter pour ces conversations sont les suivantes :

- créer une conversation d'un type donné (privée/publique)
- quitter une conversation
- pour les conversations privées :
  - inviter un ou plusieurs participants
  - annuler une invitation (qui n'aurait pas encore été consommée)
- pour les conversations publiques :
  - bannir un connecté (l'empêchant de participer à la discussion)
  - retirer un ban
  - retirer un participant
  - supprimer une conversation (automatique pour les conversations privées)

III.D.8. Gestion des modules

Au delà des fonctionnalités nécessaires au bon fonctionnement du système, il convient d'assurer une certaine part de personnalisation, permettant à toute équipe de Fansub d disposer d'outils spécifiques dont elle aurait besoin pour fonctionner. Ces outils peuvent être de multiples natures (outils de travail, adaptation du système, accessoires, …). Nous détaillons dans cette partie quelques exemples de modules qu'il serait intéressant de développer, sans pour autant entrer dans les détails techniques.

Ces fonctionnalités demandent généralement un **haut niveau de technicité** et ne s'appliquent qu'à des besoins particuliers, elles doivent donc être prises comme **techniquement facultatives** (le site doit pouvoir tourner sans elles), c'est pourquoi nous leur donnons le nom de **« modules »**. Ces modules doivent pouvoir être gérés dynamiquement de façon à ce que l'ajout, la mise à jour ou le retrait de l'un de ces modules ne corrompent pas le système.

Pour que cela se passe au mieux, chaque module doit suivre un format précis, qui l'oblige à définir ses entrées, ses sorties, les pages Web qu'il met à dispositions ainsi que d'autres contraintes telles que ses dépendances avec d'autres modules ou des versions du système (bien que ces dépendances soient à éviter autant que possible). Les gestion des modules doit permettre de :

- rajouter/enlever un module,
- activer/désactiver un module présent,
- vérifier les dépendances et autres contraintes qui entraineraient une utilisation dangereuse de ce module (comme des dépendances non satisfaites),
- générer un rapport complet donnant toutes les informations nécessaires sur un module

III.D.8.a. Modules de travail

Les rôles au sein d'une équipe de Fansub (traducteur, check, encodeur, …) sont des correspondances métier, et nécessitent donc l'utilisation d'outils spécifiques (dictionnaires, vérificateur orthographique, ...). Un point important à garder en tête est le fait que ces rôles sont définis par l'équipe, ils ne sont **pas fixés**, par conséquent nous ne spécifions ici aucun outils relativement aux rôles qui les utilisent (bien que cela soit tout à fait possible), mais par rapport à **l'utilité** que l'on peut leur reconnaître dans certains rôles. En voici une liste non exhaustive :

- **comparateur de textes**, aide le traducteur en comparant différentes traductions ou repérant les fautes corrigées (axes de recherche : commande « diff » des gestionnaires de versions)
- **dictionnaire de traduction** (mot à mot), aide le traducteur à trouver des mots particuliers (axes de recherche : traduction en ligne)
- **traducteur automatique** (expressions), donne au traducteur une première version de traduction (axes de recherche : traduction en ligne)
- **correcteur orthographique** (mot à mot), renseigne les fautes de frappe des textes traduits/adaptés (axes de recherche : dictionnaires en ligne)
- **correcteur grammatical** (expressions), renseigne les fautes de structure des textes traduits/adaptés (axes de recherche : livres de grammaires, correcteurs existants)
- **dictionnaire de synonymes**, aide l'adapt à éviter les répétitions fréquentes qui alourdissent les dialogues (axes de recherche : dictionnaires en ligne)
- **générateur de sous-titres**, génère un fichier (ASS, SRT, texte, …) d'après une description donnée (sous-titres, temps, styles, …)
- **encodeur vidéo**, allège le poste de travail de l'encodeur, qui a alors la possibilité de demander plusieurs encodages et de comparer ceux déjà fait pendant la création des suivants (axes de recherche : encodage en ligne de commande, encodage par fichier de configuration)

III.D.8.b. Modules de distraction

Une équipe de Fansub est avant tout composée de gens qui se rassemble pour effectuer une activité précise. Néanmoins cette équipe n'est pas sans **une communauté** pour apprécier son travail. La mise à disposition d'activités pour cette communauté est un facteur important de **convivialité**. Elle offre à chacun l'occasion de créer des liens, ce qui facilite d'autant plus les échanges et, ainsi, l'amélioration du travail de l'équipe (que ce soit par l'ambiance ou par les critiques échangées). Pour cela on peut imaginer de nombreux modules, parmi les plus connus (au sein de forums actuels) :

- des jeux basés sur les connaissances (quizz, deviner des noms, imaginer des histoires, …)
- des jeux basés sur la participation (compte/décompte, deviner qui sera le prochain à participer, …)
- des jeux plus génériques (cf. tous les jeux Flash trouvables sur le net)
1. ...

III.E. Interface graphique

L'interface graphique est la partie visible de l'iceberg qu'est ce projet : toutes les fonctionnalités développées dans ce projet doivent être **accessibles** (si tant est qu'elles peuvent être utilisées manuellement) dans cette interface, aux **moments opportuns** et de la manière la plus **pratique** que possible. Cela implique notamment un agencement intelligent de cette interface (III.D.1. Interface) et une étude orientée selon chaque point de vue possible : internaute (III.D.2. Pages publiques), connecté (III.D.3. Pages privées) et membre(III.D.4. Pages réservées).

III.E.1. Interface

L'interface doit être facile et rapide à utiliser, aussi un haut degré d'optimisation doit être trouvé, les optimisations les plus évidentes étant les suivantes :

- **tout doit s'afficher** d'un bloc (ne pas avoir à jouer avec l'ascenseur pour trouver les éléments), dans le cas des longs textes on préfèrera réduire leur place (leur assigner un ascenseur à eux) plutôt que de les afficher pleinement (et risquer de cacher les autres éléments lors du déplacement).
- l'accès à une zone particulière doit s'effectuer en un **minimum de cliques** (3 est une bonne limite)
- l'intégralité de la page doit-être générée **selon les besoins** de l'utilisateur (afficher ce qui est nécessaire, sans superflus, de façon à ne pas perdre l'utilisateur et économiser de la place)
- l'ensemble de ces contraintes doivent être appliquées dans la **majorité des cas** (en particulier pour ceux qui désactivent *cookies* et *JavaScript*, ou pour les terminaux mobiles tels que les téléphones ou les mini-PC), il est donc important de tester ces différents cas de façon à s'assurer que les contraintes sont bien respectées

Voici l'agencement que nous souhaitons avoir :

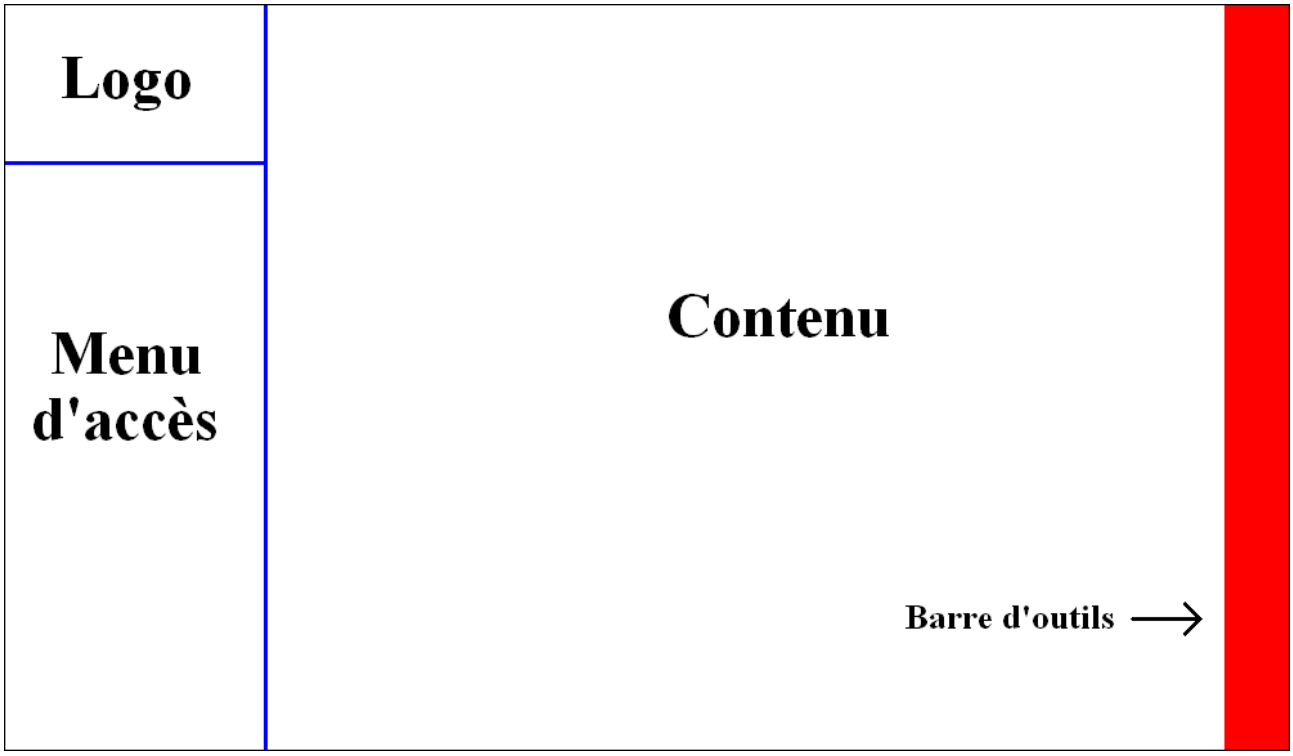


Illustration 1: agencement de l'interface

Cet agencement reprend les ergonomies classiques : menu à gauche, logo dans le coin, contenu au centre de la page. À cela nous ajoutons une barre d'outils. Avant de préciser son rôle, précisons clairement celui des autres parties :

- le **logo** est une image uniquement (donc sans texte, à moins qu'il ne soit intégré dans l'image) représentant l'équipe de Fansub
- le **menu** est une liste de liens permettant d'accéder à différentes parties du site, tel que l'accueil, la présentation de l'équipe, les projets actuels, …
- le **contenu** est l'information à donner à l'internaute, selon ce qu'il a demandé (les pages référencées par le menu entre-autres)
- la **barre d'outils** est une barre personnalisable, son objectif principal est de centraliser les fonctionnalités intéressant l'internaute, ce qui peut inclure la connexion/déconnexion, la création de nouvelles conversations, les outils propres aux membres (activation et désactivation de la correction automatique, ouverture d'un dictionnaire pour y chercher un mot, …), cependant l'ensemble de ces fonctionnalités doit être accessible sans passer par cette barre d'outils (ce ne sont que des raccourcis)

En soit, l'utilité de la barre d'outils se limite à offrir un moyen **rapide** d'accéder à des fonctionnalités particulières. Cela semble similaire à une liste de favoris d'un navigateur, mais il est important de ne pas la limiter à cette utilisation : les favoris d'un navigateur remplace la page courante par une nouvelle (celle des favoris), l'objectif de la barre d'outils est d'**interagir avec l'interface** de manière plus intelligente (remplacement de la page -similaire aux favoris-, ouverture d'une « popup », activation/désactivation de fonctionnalités automatiques, …). De ce fait, on peut sentir que son rôle se limite principalement aux **connectés** et aux **membres**. Cela reste néanmoins à discuter pour chaque fonctionnalité (si plusieurs fonctionnalités sont intéressantes pour les internautes non connectés, la barre d'outils peut être définitivement intégrée à l'interface).

Un exemple rapide et très épuré basé sur cette organisation :

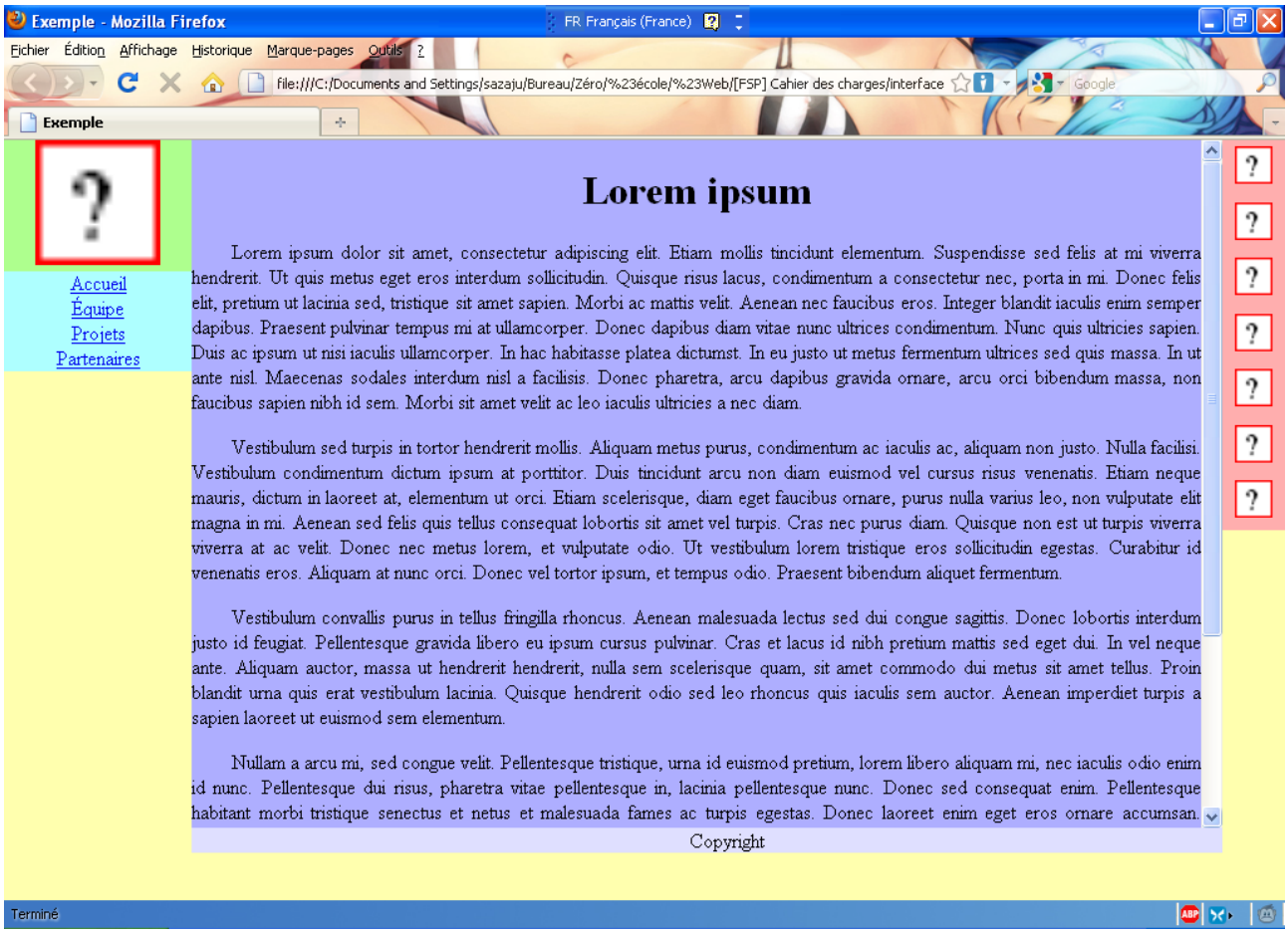


Illustration 2: exemple épuré basé sur l'agencement donné

Dans cet exemple le bas de page n'est pas atteint, en effet pour éviter d'avoir un ascenseur on privilégie fixer une taille inférieure à celle de la page, plutôt que risquer de voir un ascenseur sur la page entière, entrainant la disparition du menu (à gauche) et de la barre d'outils (à droite) lors du déplacement. Notons que cela est compensé par un ascenseur assigné au contenu lui-même, permettant d'accéder à l'ensemble du texte. Dans le cas où la barre d'outils n'aurait pas été nécessaire, le cadre de contenu peut prendre la place libérée pour son propre usage :

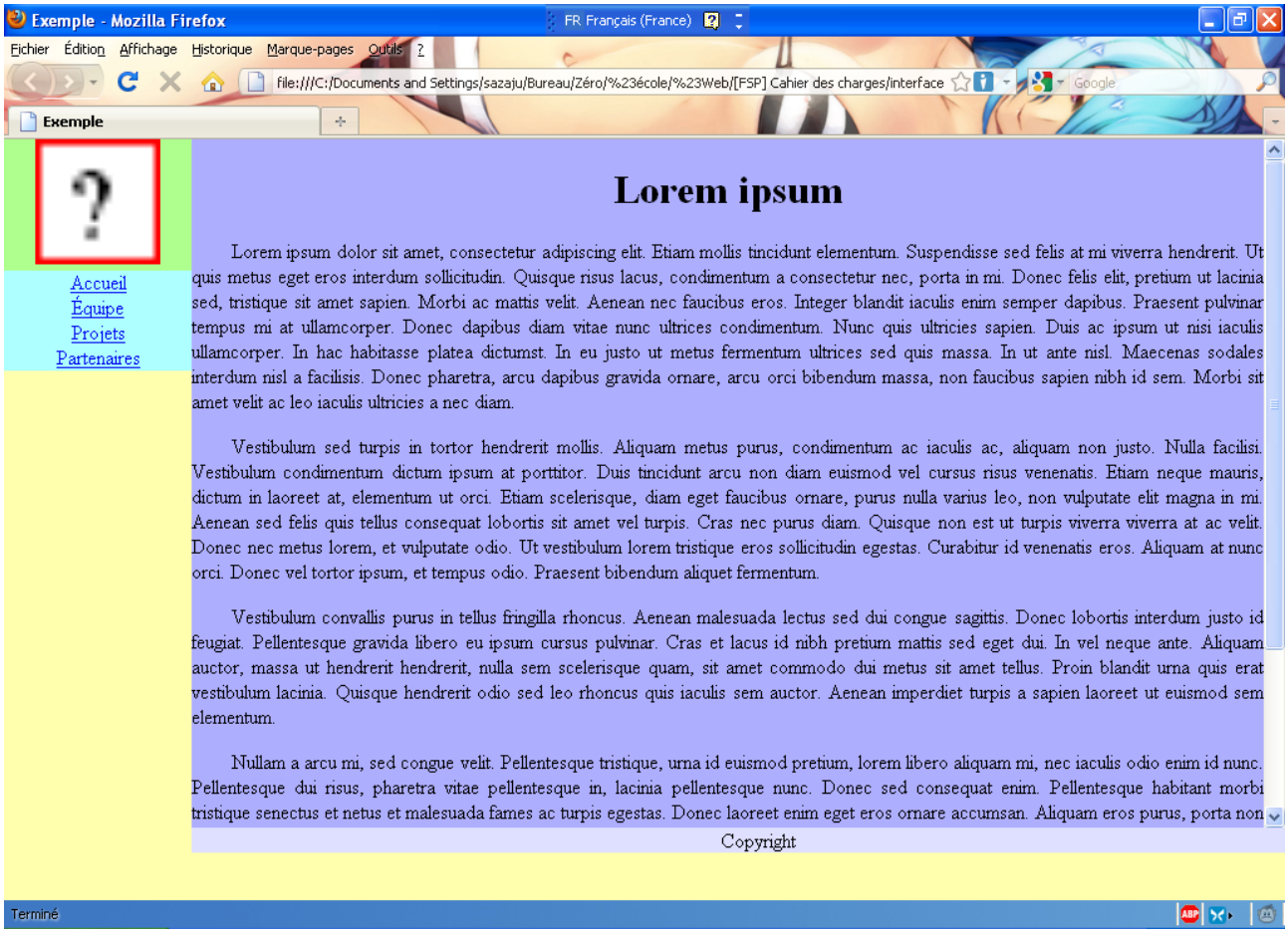


Illustration 3: exemple sans barre d'outils

Le menu est ici très basique (un seul niveau), mais ce n'est pas une contrainte : **plusieurs niveaux** peuvent être affichés, mais il convient de faire **attention à la place** que cela doit prendre (il serait dommage de devoir ajouter un ascenseur au menu). Selon la place disponible, il est possible de **rajouter** des éléments (liste de partenaires, publicité, …), cependant cela tient plus de la personnalisation que de l'optimisation et n'est donc pas une priorité.

III.E.2. Pages publiques : accès internautes

III.E.2.a. Accueil

III.E.2.b. News

III.E.2.c. Équipe

III.E.2.d. Projets

III.E.2.e. FAQ

III.E.2.f. Autres pages

III.E.3. Pages privées : accès connectés

III.E.3.a. Forum

III.E.4. Pages réservées : accès membres

III.E.4.a. Gestion des projets

III.E.4.b. Gestion des membres

III.E.4.c. Gestion des droits

III.E.4.d. Gestion des fichiers

IV. Implémentation

Ce chapitre ne détaille pas l'implémentation en elle-même, mais les axes à suivre pour celle-ci (ce n'est pas un rapport technique). En se basant sur les **fonctionnalités** souhaitées (III.D. Fonctionnalités) et sur les **contraintes de développement** que cela implique, nous partons sur une architecture **modulaire** permettant l'intégration des fonctionnalités progressivement, avec un « noyau » de fonctionnalités de bases. Ce chapitre s'organise donc suivant cette organisation modulaire : une **description globale** de l'architecture (IV.A. Architecture globale) pour en avoir une vue d'ensemble, et ainsi en comprendre rapidement les différentes parties, suivi par la description détaillée des **différentes couches**.

IV.A. Architecture globale

IV.B. Kernel : le noyau du système

IV.C. Modules : les fonctionnalités avancées

IV.D. Interface Web : l'accès au système

V. Ressources annexes

Sources du projet (dépôt Mercurial) : http://mercurial.intuxication.org/hg/zerofansubsite

Ressources utilisées pour compléter ce dossier :

- XHTML : http://fr.wikipedia.org/wiki/Extensible_HyperText_Markup_Language
- CSS : http://fr.wikipedia.org/wiki/Feuilles_de_style_en_cascade
- Javascript : http://fr.wikipedia.org/wiki/Javascript
- AJAX : http://fr.wikipedia.org/wiki/Asynchronous_JavaScript_And_XML
- JSP : http://fr.wikipedia.org/wiki/JavaServer_Pages
- PHP : http://fr.wikipedia.org/wiki/Php
- ASP : http://fr.wikipedia.org/wiki/Active_server_pages
- Flash : http://fr.wikipedia.org/wiki/Adobe_Flash
- tutoriels de prise en main de http://www.developpez.com/
- gestion des données : http://fr.wikipedia.org/wiki/Cat%C3%A9gorie:Gestion_des_donn%C3%A9es