

08/01/2016

Version 1.2

Maxime HELARY

DOSSIER DE SOUTENANCE POUR LE TITRE CONCEPTEUR DEVELOPPEUR INFORMATIQUE

Application internet pour améliorer le suivi et la communication
autour des projets de la société ATLANTEDATA.

INTRODUCTION AU PROJET.....	6
REMERCIEMENTS.....	7
PRESENTATION DE L'ENTREPRISE.....	8
PROJECT SUMMARY	9
PARTIE 1 : DESCRIPTION ET CAHIER DES CHARGES DU PROJET	11
ANALYSE DE L'EXISTANT	12
L'existant au sein de l'entreprise	12
Les logiciels du marche	13
LES UTILISATEURS DU PROJET	13
Les développeurs.....	13
Les clients	13
LES ECHEANCES DU PROJET	14
Début du projet	14
Date limite du projet	14
Disponibilités des personnes concernées	14
CONTEXTE TECHNIQUE.....	14
DEFINITION DES ENTITÉS.....	15
Le projet	15
Le dossier.....	15
Le développement.....	15
Une demande.....	15
LES FONCTIONNALITÉS ATTENDUES	15
Fonctionnalités primordiales	15
Fonctionnalités secondaires	16
PARTIE 2 : GESTION DE PROJET.....	17
PARTIE 3 : ANALYSE FONCTIONNELLE	19
LES ACTEURS.....	20
Chef de projet.....	20
Développeur	20

Client.....	20
SEPARATION DES DOMAINES FONCTIONNELS	20
Gestion des demandes	20
Gestions des projets	20
Gestion des dossiers/developpements	20
Autres	21
CAS D'UTILISATION.....	22
Gestion des demandes	22
Gestion des projets.....	23
Gestion des dossiers/realisations.....	24
DESCRIPTION TEXTUELLE.....	25
Valider une tache d'un developpement.....	25
Faire une recherche de dossier auquel le developpeur DEV-A est affilie	25
Ajouter un document electronique a un dossier	26
SPECIFICATIONS	27
MAQUETTAGE.....	28
Connexion	28
Ecran d'accueil chef de projet.....	29
Ecran d'accueil développeur.....	30
Ecran d'accueil client	31
Page recherche	32
Page projet.....	33
Page dossier	34
Page développement (realisation/SAV)	35
Page nouvelle demande	36
Page documentation.....	37
Page GED	38
DIAGRAMME DE NAVIGATION	40
PARTIE 4 : ANALYSE TECHNIQUE	41
DIAGRAMME DES CLASSES METIERS	42

LA BASE DE DONNEES.....	46
Exemple de dictionnaire de donnees	46
ARCHITECTURE LOGICIELLE	50
PARTIE 5 : RÉALISATION	51
CHOIX DE DEVELOPPEMENTS.....	52
Choix des langages	52
Choix des frameworks	52
Logiciels et autres outils.....	52
L'ARBORESCENCE DU PROJET ET EXPLICATION	53
La BO (Business Object).....	53
Le Controller_php.....	54
LA DAL (Data Access Layer).....	54
Le dossier « gesprj »	54
Le dossier « lib ».....	55
Le dossier « Slim »	55
Le dossier « Tools »	55
La racine du projet.....	56
EXEMPLE DE REALISATION N°1 : LA CONNEXION UTILISATEUR.....	57
La gestion de l'application par Slim	57
La page index.html	58
Le fichier app.js.....	59
La page login.html	60
Le contrôleur LoginController	61
La fonction login de l'authenticateservice.....	62
La route /login en POST dans l'index.php.....	63
Le fichier main.html.....	65
La directive « central »	65
EXEMPLE DE REALISATION N°2 : LA GESTION DE PROJET POUR UN CHEF DE PROJET.....	67
Le système d'onglet	67
le tableau des projets d'un chef de projet.....	68

Le contrôleur du tableau des projets	69
Le formulaire projet	72
CONCLUSION.....	78
Sur le respect du cahier des charges.....	79
Sur la gestion de projet.....	79
Sur l'avenir du projet.....	80
ANNEXE.....	81
EXEMPLE DE COMPTE-RENDU DES REUNIONS D'ENQUETES	82
Réunion du 04/08/2015 + Réunion du 07/08/2015 : Recueil des besoins utilisateurs	82

INTRODUCTION AU PROJET

REMERCIEMENTS

Tout d'abord, je souhaite remercier l'ENI qui m'a donné la chance de poursuivre mes études comme je souhaitais, c'est-à-dire en alternance pendant ces deux années. C'était primordial pour moi d'intégrer très vite le milieu du travail en informatique. Je remercie aussi leurs formateurs pour leur compétence et leur connaissance de l'entreprise et de ses enjeux.

Ensuite, je remercie M. Valery HAUTON et toute la société ATLANTEDATA de m'avoir fait confiance pendant ces deux années, et de la sympathie dont a fait preuve toute l'équipe, que cela soit les développeurs ou les administrateurs système.

Enfin, je tiens à remercier vivement Mme Marie PERRET, qui a toujours été disponible quand il y a avait besoin et qui a toujours su insufflé de la motivation et de l'énergie aux bons moments.

PRESENTATION DE L'ENTREPRISE

atlantedata

ATLANTEDATA est une société de services informatique située au cœur de Nantes. Elle a été créée en 1993. Elle comporte un service système/réseau de deux personnes et d'un service développement de 7 personnes (dont un chef de projet).

Les entreprises clientes sont majoritairement des PME-PMI, qui nécessitent des outils de gestion de leur entreprise, que cela soit la gestion de leur planning, de leurs ventes, de leur comptabilité. Elles proviennent de domaines variés, de la vente de vêtements en ligne, d'un distributeur de matériel électrique, ou de la gestion de négoce.

Philippe ARNAUD

Directeur

Valery HAUTON

Directeur / Chef de projet

NELLY DELAURENS

Secrétaire de direction

SYSTEME/RESEAU

Gaetan BOISMARTEL

Administrateur
Réseau/Système

Sebastien COGNACQ

Administrateur
Réseau/Système

DEVELOPPEMENT

Vincent HERAUD

Chef de projet /
Concepteur développeur

Didier HEURTAUX

Concepteur développeur

Cédric FAUCHER

Concepteur développeur
en alternance

Pascal RAUX

Concepteur développeur

Véronique BOUYER

Concepteur développeur

Maxime HELARY

Concepteur développeur
en alternance

PROJECT SUMMARY

This project was launch in mid-2015 to make it easier communication between all of the persons who take part of an IT project for the company ATLANTEDATA, to improve the project monitoring.

Indeed, there were no other tools than a wiki, emails and phone to communicate, which is not so great.

The wiki was responsible for a significant waste of time and was not used by most developers because it lacked organization. So we decided to launch this project, on which I was the only designer and the only developer.

The first phase was to discuss with the others developers, to understand what was fundamental for them in this type of application. The biggest wish was to have all the information around a project on a single page or easily accessed on this page. Other requests were for example the fact that the application sends automatically emails when it is necessary, an electronic document management, or a documentation section.

Obviously, the application had to manage user accounts, creating and modifying projects or owing a schedule management. But the goal was not to make a project management application so the schedule management had to stay very basic.

The second phase was to write the technical and functional specifications, and the third was to make the application, or at least make the main part and make it easy for another developer to continue the project. I had between 50 and 60 man/day for that.

ANNEXE : FICHE DE SUIVI DU PROJET EN ENTREPRISE
TITRE PROFESSIONNEL CONCEPTEUR(TRICE) DEVELOPPEUR(SE) INFORMATIQUE
 (Arrêté du 11 février 2013 relatif au titre CDI)

Document complété d'un commun accord entre le stagiaire, le responsable du stage en entreprise et le responsable du centre organisateur, à joindre aux résultats des évaluations passées en cours de formation.

Compétences Voir le détail dans le référentiel d'emploi, d'activités et de compétences du titre CDI	Cocher les compétences mises en œuvre lors du projet en entreprise (en totalité ou partiellement)
1. Maquetter une application	☐
2. Développer une interface utilisateur	☐
3. Développer des composants d'accès aux données	☐
4. Développer des pages web en lien avec une base de données	☐
5. Concevoir une base de données	☐
6. Mettre en place une base de données	☐
7. Développer des composants dans le langage d'une base de données	☐
8. Utiliser l'anglais dans son activité professionnelle en informatique	☐
9. Concevoir une application	☐
10. Collaborer à la gestion d'un projet informatique	☐
11. Développer des composants métier	☐
12. Construire une application organisée en couches	☐
13. Développer une application de mobilité numérique	☐
14. Préparer et exécuter les plans de tests d'une application	☐
15. Préparer et exécuter le déploiement d'une application	☐

Observations éventuelles :

Du stagiaire :

Du centre organisateur :

De l'entreprise :

Nom et prénom du stagiaire :

Signature :

Entreprise :

Nom du responsable du stage :

Signature et cachet :

Centre organisateur :

Nom du responsable du centre organisateur :

Signature et cachet :

PARTIE 1 : DESCRIPTION ET CAHIER DES CHARGES DU PROJET

Aujourd'hui, la société ATLANTEDATA est en demande d'une application qui doit permettre de :

- faciliter la communication entre les différents acteurs d'ATLANTEDATA :
 - chefs de projets
 - développeurs
 - clients
- et d'améliorer le suivi des projets informatiques de l'entreprise.

Les bénéfices d'une telle application sont triples :

- Améliorer la relation client/entreprise
- Faire gagner du temps à tous les acteurs
- Réduire le coût global d'un projet

Attention, cela ne doit pas être un logiciel de gestion de projet pour le moment. En effet, le but est d'avoir une application claire et simple afin de « fidéliser » les clients à cette application (qui ne sera pas obligatoire pour eux).

Par contre, il n'est pas à exclure par la suite d'ajouter des fonctionnalités qui pourront en faire un logiciel de gestion de projet à part entière.

Le projet est mené de A à Z par moi-même.

ANALYSE DE L'EXISTANT

L'EXISTANT AU SEIN DE L'ENTREPRISE

LE TELEPHONE

La voie téléphonique a deux problèmes majeurs :

- 1) Problème de disponibilité : les personnes ne sont pas forcément joignables par téléphone
- 2) Problème de perte d'information : que cela soit les développeurs ou les clients, il est possible que des informations se transmettent mal et il en résulte une perte d'information qui peut potentiellement être critique.
- 3) L'information ne passe qu'entre 2 acteurs du projet et n'est pas stockée

Mais elle a aussi des avantages :

- 1) Rapidité de la transmission de l'information
- 2) Facilité de réaction à l'information

L'EMAIL

Problèmes majeurs de l'email :

- 1) Manque de priorisation de l'information entre les emails
- 2) Retrouver un email n'est pas simple
- 3) Le tri des emails est quelque chose de fastidieux

Avantage :

- 1) Il est facile et rapide d'envoyer un mail
- 2) L'information est stockée
- 3) L'email peut réunir plus de deux acteurs du projet

LE WIKI

Aujourd'hui, une partie des développeurs et des clients, mais peu, utilisent un Wiki (DokuWiki) afin de communiquer. Ils aimeraient l'utiliser mais ils lui trouvent trop de problèmes :

- 1) Manque de clarté
- 2) Pas d'organisation évidente entre les pages
- 3) L'outil de création de contenu textuel n'est pas ergonomique

Et peu d'avantages :

- 1) Possibilité d'avoir beaucoup d'informations (textuelles et images) sur une même page

LES LOGICIELS DU MARCHÉ

L'optique est d'avoir un logiciel fait en interne car

- devant répondre exactement aux besoins des développeurs d'ATLANTEDATA,
- devant pouvoir intégrer facilement d'autres fonctionnalités par la suite,
- facilitant la transmission du développement entre développeurs

Il n'a donc pas été question d'utiliser un logiciel existant sur le marché, qu'il soit open source ou payant

LES UTILISATEURS DU PROJET

Les destinataires du projet sont :

- Les développeurs (et chef de projet) d'ATLANTEDATA
- Les clients d'ATLANTEDATA.

LES DEVELOPPEURS

Il y a 7 développeurs actuellement dans l'équipe. Ils sont tous familiers avec l'outil informatique et le suivi de projet.

LES CLIENTS

La société a une vingtaine de clients, de taille d'entreprise variable, possédant ou non une équipe informatique. Ils ne sont pas donc pas forcément à l'aise avec la technologie en général et le suivi de projet.

Cela implique un travail important à réaliser sur l'ergonomie et la facilité d'accès.

Il est difficile d'avoir une idée du nombre de clients qui utiliseront l'application pour le moment, mais il a été établi qu'une cinquantaine de personnes devront pouvoir être connectés à l'application en même temps (clients + développeurs).

Tous les acteurs ont accès à Internet (mais uniquement à Internet, pas à un réseau interne donc).

LES ECHEANCES DU PROJET

DEBUT DU PROJET

Le projet a été lancé le 03 aout 2015.

DATE LIMITE DU PROJET

La date limite du projet est de fin décembre 2015.

Le projet étant un objectif d'entreprise sur le long terme et ne nécessite pas une mise en production rapide, le développement sera axé sur 3 points primordiaux :

- Une analyse complète du projet
- La réalisation des fonctionnalités principales telles que les onglets, les tableaux et formulaires.
- La maintenabilité du code pour que d'autres personnes puissent continuer le projet par la suite

La date de clôture est donc incitative et non pas primordiale.

DISPONIBILITES DES PERSONNES CONCERNEES

Le seul développeur sur le projet est moi-même. J'ai, sur ce projet, entre 1 et 3 jours ouvrés par semaine à m'y consacrer. Voir la partie Gestion de projet pour d'amples informations.

Les autres développeurs sont disponibles de façon ponctuelle, à ma demande, pour de courtes réunions.

Ces réunions concerneront le recueil des attentes des utilisateurs ainsi que la validation/discussions autour des fonctionnalités attendues et des maquettes.

Les clients d'ATLANTEDATA ne seront pas disponibles pour réagir au sujet de l'application.

CONTEXTE TECHNIQUE

Les utilisateurs disposant de PC avec une connexion internet, et étant majoritairement distant, il faut une application client léger (navigateur web).

L'application devra fonctionner de la même façon dans les dernières versions des navigateurs suivants :

- Firefox
- Chrome

Pour l'instant, IE n'est pas une cible prioritaire.

La base de données devra être une base de données relationnelle et gratuite.

Les langages utilisés ne sont pas imposés.

DEFINITION DES ENTITÉS

Voici la liste des entités concernant le suivi de projet chez ATLANTEDATA :

LE PROJET

- C'est la plus grande entité.
- Elle peut contenir plusieurs dossiers.

LE DOSSIER

- Un dossier est une sous-entité d'un projet.
- Un dossier doit être rattaché à un projet.
- Un dossier peut contenir des développements.

LE DEVELOPPEMENT

- Un développement est une sous-entité d'un dossier.
- Il peut être soit de type SAV soit de type REALISATION

UNE DEMANDE

- Une demande est un souhait émis par un client.
- Une demande peut être typée en tant que projet, dossier, réalisation.
-

LES FONCTIONNALITÉS ATTENDUES

Suite aux réunions utilisateurs, voici la liste des fonctionnalités finales attendues par ATLANTEDATA ainsi que quelques détails souhaités.

FONCTIONNALITES PRIMORDIALES

PAGE DE CONNEXION A L'APPLICATION

L'application ne doit être accessible qu'en passant par une page de connexion. Celle-ci ne comportera que le module de connexion et aucune autre donnée.

Les utilisateurs peuvent se connecter selon 3 rôles :

- Développeur
- Chef de projet
- Client

PAGE D'ACCUEIL

Après la connexion, chaque utilisateur a accès à une page d'accueil lui affichant les informations primordiales selon son type.

RESUME D'UNE ENTITE

Chaque entité doit avoir une page simple et claire résumant ce qu'elle est :

- Sa description
- La liste de ses sous-entités
- Un lien rapide vers son entité-mère si elle en a une

Toutes les entités doivent avoir un suivi de son avancée interne.

MAILING

L'application doit comporter un système de mailing. Les utilisateurs créant du contenu peuvent choisir ou non d'envoyer des notifications par mails à d'autres utilisateurs, selon une liste prédéfinie par projet.

GESTION ELECTRONIQUE DE DOCUMENTS (GED)

Chaque entité doit pouvoir avoir des documents liés à elle et visibles rapidement.

GESTION D'UNE ENTITE

La gestion (création/modification/suppression) d'une entité doit n'être accessible que par les responsables de projet.

UN SYSTEME DE COMMENTAIRES SUR DES ENTITES

Il est demandé que les utilisateurs puissent commenter les entités.

FONCTIONNALITES SECONDAIRES

UN OUTIL DE PLANNING

Un projet devrait potentiellement avoir un outil de planning basique.

UN OUTIL DE RECHERCHE

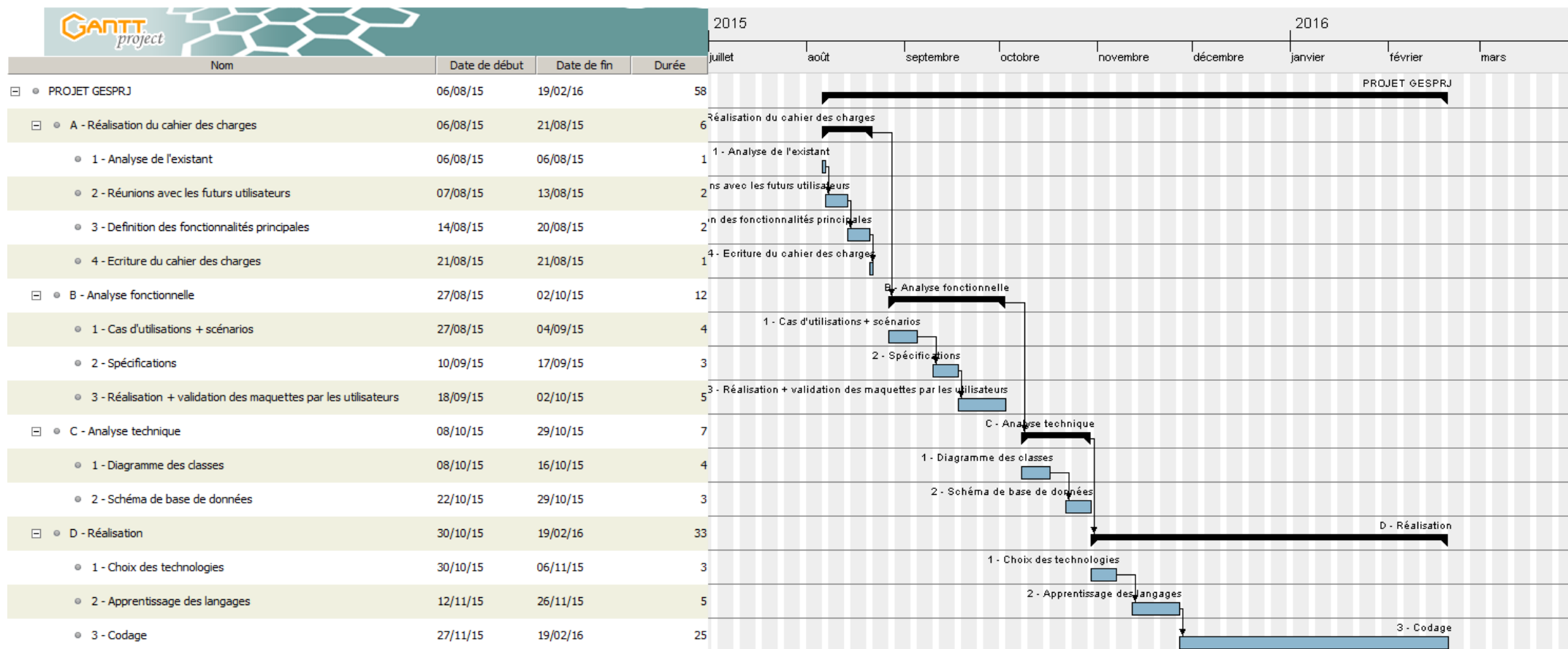
Il devrait être facile d'effectuer des recherches précises de projet ou de contenus.

UN ESPACE DE DOCUMENTATION

Chaque entité doit avoir une partie documentation séparée.

PARTIE 2 : GESTION DE PROJET

Comme je travaille seul, que je mène d'autres tâches en parallèle du projet et que je ne savais pas à l'avance les jours précis pour travailler sur le projet, j'ai décidé de faire un planning de GANTT en ne mettant que 2 jours/hommes par semaine sur le projet, chiffre qui me paraît le plus logique. Pour rappel, le projet a été lancé le



Comme on peut le constater, la fin du contrat étant fin janvier, je n'aurais à priori pas le temps de faire tout le codage ni les tests. Il y aura logiquement donc 2 livrables réalisés pour le projet, la livraison du produit finalisé n'étant pas faisable dans le temps imparti :

- Cahier des charges : 21 août 2015
- Maquettes finalisées et acceptées par les utilisateurs : 02 octobre 2015
- Réalisation des fonctionnalités principales

PARTIE 3 : ANALYSE FONCTIONNELLE

LES ACTEURS

Suite à l'expression des besoins, 3 rôles ont été définis :

CHEF DE PROJET

Le chef de projet est l'acteur « maître ». Il a un rôle d'administration : création et gestion de comptes utilisateurs, de projet, affiliation des utilisateurs aux entités ...

Il a, en plus de siennes propres, accès à toutes les fonctionnalités des autres acteurs.

DEVELOPPEUR

Le développeur est l'acteur « de base » de l'application. Il doit pouvoir gérer (ajouter, supprimer, modifier) les entités sous-jacentes de l'entité projet auquel il est rattaché. Il a accès aux outils d'aide à la gestion tels que le planning, l'avancement du projet ou la gestion électronique de document, et interagir avec.

CLIENT

Le client est le dernier acteur de l'application. Il peut créer/gérer des demandes. A l'instar du développeur, il peut interagir (de façon simple via des commentaires) avec les entités auxquelles il est lié (sauf l'entité projet), et aux outils d'aide à la gestion de projet.

SEPARATION DES DOMAINES FONCTIONNELS

Afin d'appréhender plus aisément le projet, je l'ai découpé en domaines fonctionnels.

GESTION DES DEMANDES

Ce domaine fonctionnel regroupe toute la partie concernant les demandes clients (création par le client (acceptation par le chef de projet, passage de la demande dans une autre entité...))

GESTIONS DES PROJETS

Cette partie va regrouper tout ce qui concerne l'entité « projet », comme par exemple :

- Création d'un projet par le chef de projet
- Affiliation des utilisateurs (clients/développeurs) par le chef de projet
- Outils d'aide à la gestion de projet (planning, avancement du projet...)

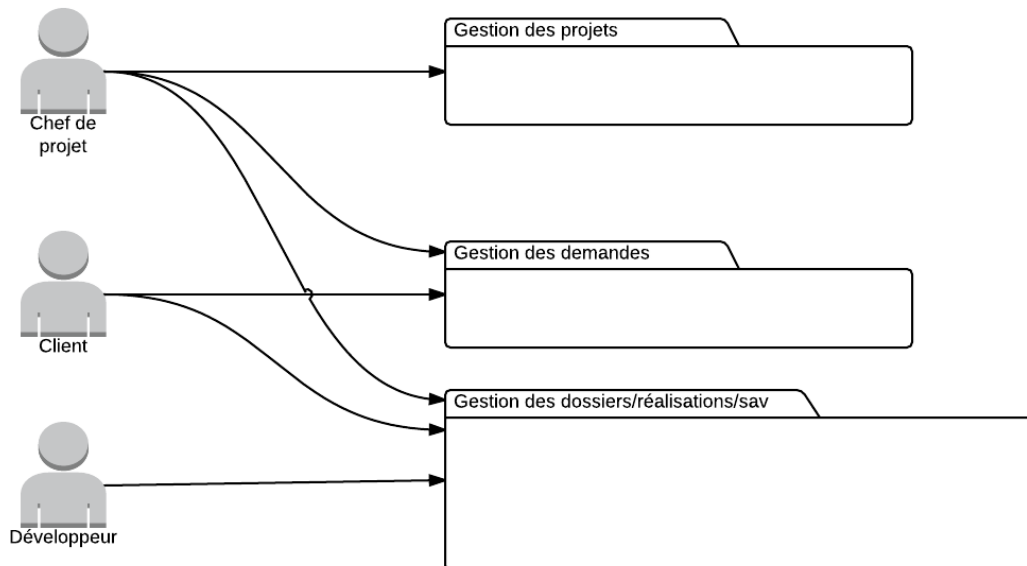
GESTION DES DOSSIERS/DEVELOPPEMENTS

Ce domaine est la plus grosse partie fonctionnelle. Elle est le cœur de l'application. Elle consiste en la gestion (création/modification/suppression/affichage) des entités sous-jacentes au projet. Le développeur est l'utilisateur principal des fonctionnalités de ce domaine.

AUTRES

Voici les domaines restants :

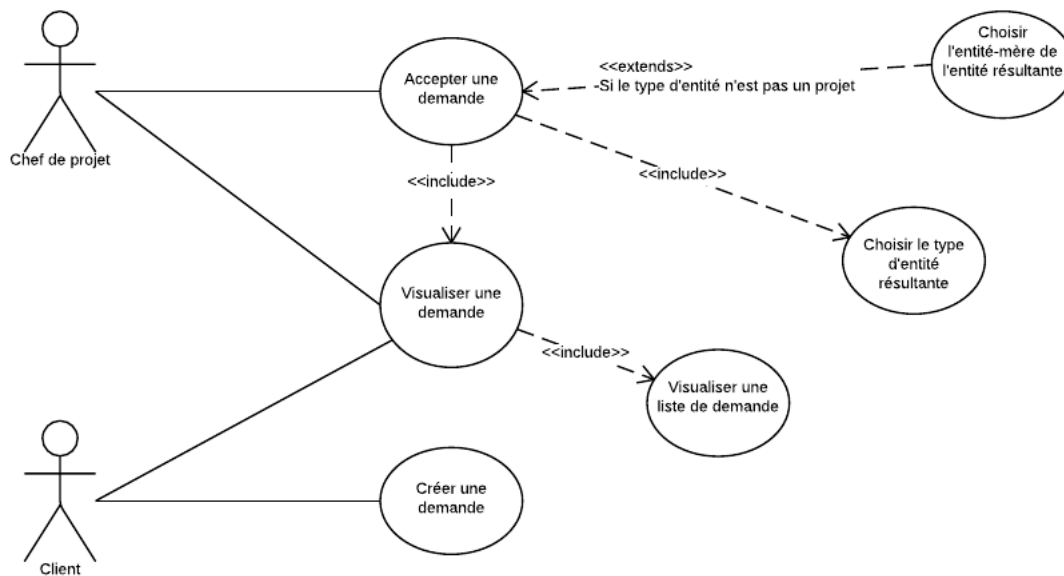
- La Gestion Electronique de Documents (GED)
- La recherche d'entités
- La partie Documentation
- Le planning
- L'envoi d'email entre utilisateurs

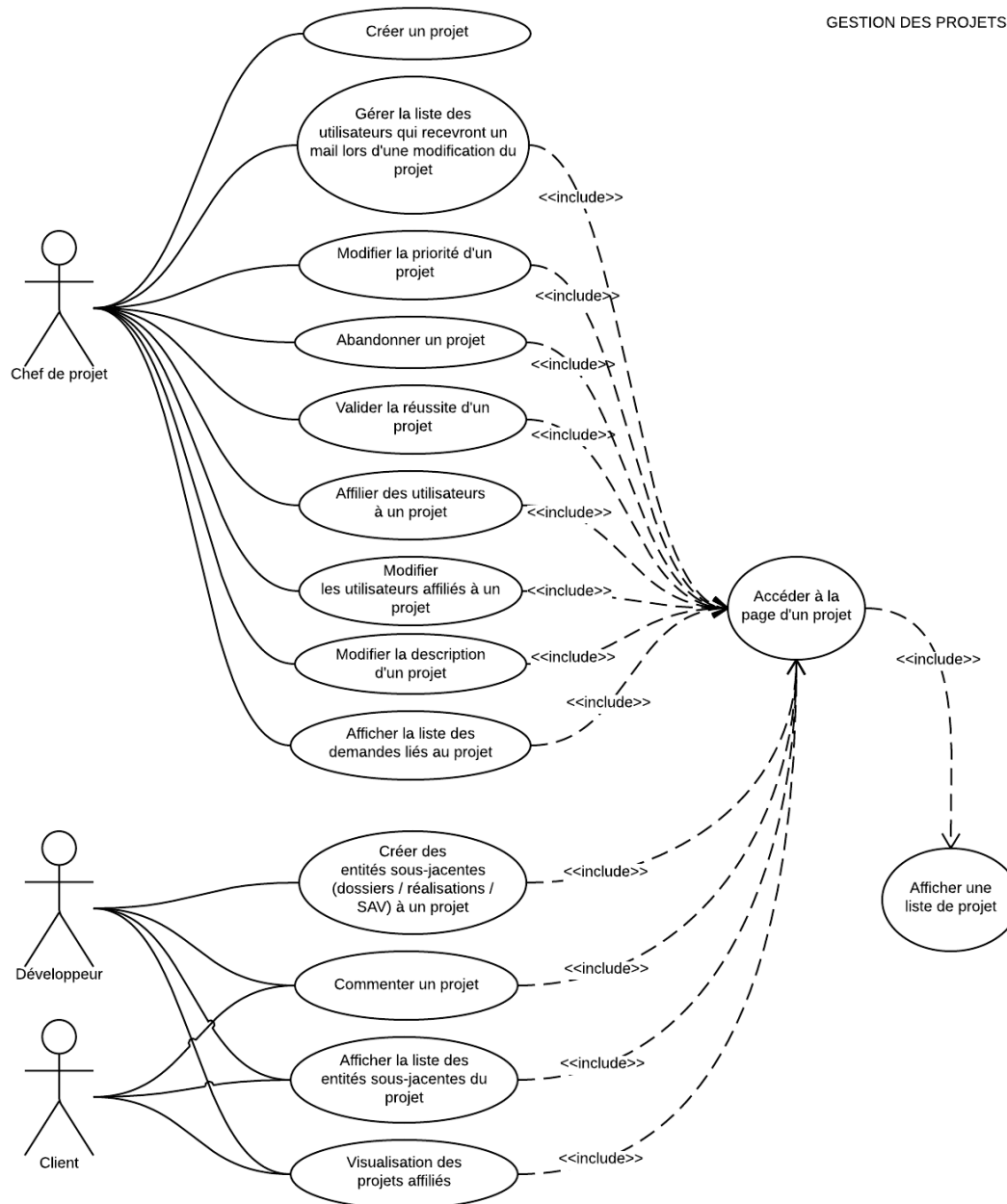


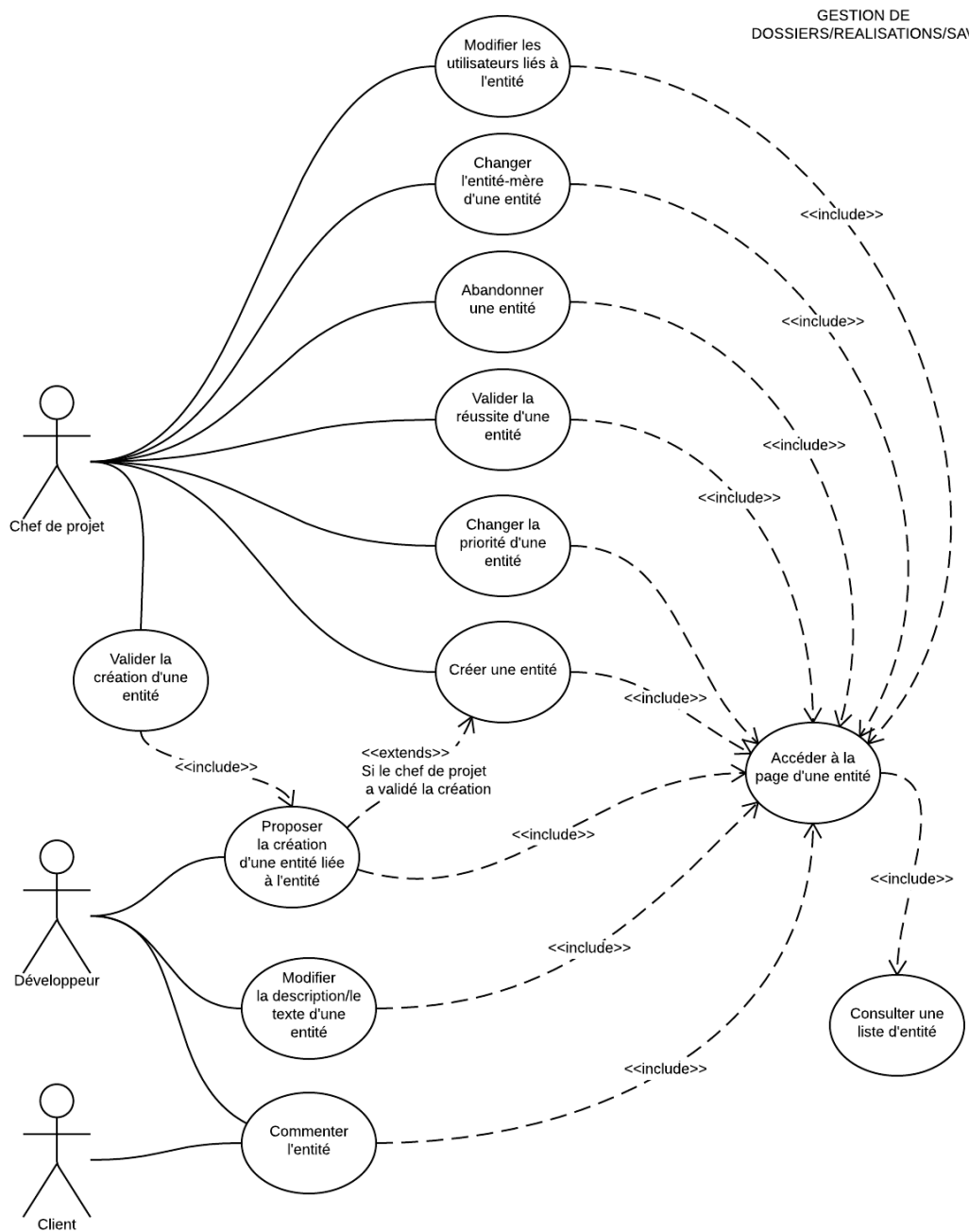
CAS D'UTILISATION

Ensuite, voici les 3 principaux package de cas d'utilisation, séparés par leurs domaines fonctionnels :

GESTION DES DEMANDES







DESCRIPTION TEXTUELLE

Ci-dessous quelques exemples de descriptions textuelles que j'ai réalisées dans l'optique de bien planifier le déroulement des actions sur le site.

VALIDER UNE TACHE D'UN DEVELOPPEMENT

Acteur(s) : Développeur

Description : Le développeur doit pouvoir valider un développement

Préconditions : L'utilisateur doit être authentifié en tant que « Développeur »

Démarrage : Le développeur est sur sa page d'accueil

SCENARIO NOMINAL

Etape	Utilisateur	Système
1.		Le système affiche une liste des développements auxquelles le développeur est attaché
2.	Le développeur sélectionne une des développements	
3.		Le système affiche la page du développement
4.	Le développeur sélectionne une des tâches	
5.	Le développeur valide la tâche sélectionnée	
6.	Le développeur clique sur « Valider les modifications »	
7.		Le système met à jour la tâche sélectionnée
8.		Le système réaffiche la liste des tâches mise à jour

SCENARIO ALTERNATIF

Etape	Utilisateur	Système
2.a	Le développeur n'a pas de réalisations attachées	
4.a	Le développeur revient sur la page du dossier	
6.a	Le développeur clique sur « Annuler les modifications »	
6.b		Le système réaffiche la page comme à l'origine

FAIRE UNE RECHERCHE DE DOSSIER AUQUEL LE DEVELOPPEUR DEV-A EST AFFILIE

Acteur(s) : Chef de projet

Description : Le client doit pouvoir faire une recherche de dossier avec critères

Préconditions : L'utilisateur doit être authentifié en tant que « Chef de projet »

Démarrage : Le chef de projet est sur sa page d'accueil

SCENARIO NOMINAL

Etape	Utilisateur	Système
1.	Le chef de projet clique sur «Recherche »	
2.		Le système affiche la page de recherche par défaut (projet)
3.	Le chef de projet clique sur « Recherche un dossier »	
4.		Le système affiche la page de recherche de dossier
5.	Le chef de projet remplit la zone « Développeur » en choisissant le développeur « DEV-A »	
6.	Le chef de projet clique sur « Rechercher selon ces critères »	
7.		Le système affiche une liste des dossiers dont le développeur « DEV-A » est affilié

AJOUTER UN DOCUMENT ELECTRONIQUE A UN DOSSIER

Acteur(s) : Client

Description : Le développeur doit pouvoir ajouter un document électronique à un dossier

Préconditions : L'utilisateur doit être authentifié en tant que « Client »

Démarrage : Le client est sur sa page d'accueil

SCENARIO NOMINAL

Etape	Utilisateur	Système
1.		Le système affiche la liste des dossiers (et projets/demandes) du client
2.	Le client clique sur un dossier et clique sur « Accéder au dossier »	
3.		Le système affiche la page du dossier
4.	Le client clique sur « GED »	
5.		Le système affiche la liste des documents liés au dossier
6.	Le client clique sur « Ajouter un document »	
7.		Le système ouvre une boîte de dialogue de sélection de dossier
8.	Le client choisit le document à ajouter et valide	
9.		Le système ajoute le fichier sur le serveur Le système affiche la liste des documents actualisés

SCENARIO ALTERNATIF

Etape	Utilisateur	Système
8.a		Le système indique que le document existe déjà
8.b		Le système propose de remplacer l'ancien fichier par le nouveau

8.c.1	Le client accepte	
8.d.1		Le système remplace l'ancien fichier par le nouveau
8.c.2	Le client refuse	
8.d.2		Le système réaffiche la liste des fichiers originaux

SPECIFICATIONS

J'ai préféré, dans ce dossier, mettre plutôt des maquettes très complètes plutôt que toutes les spécifications, étant à mes yeux plus visuel et aisé à comprendre.

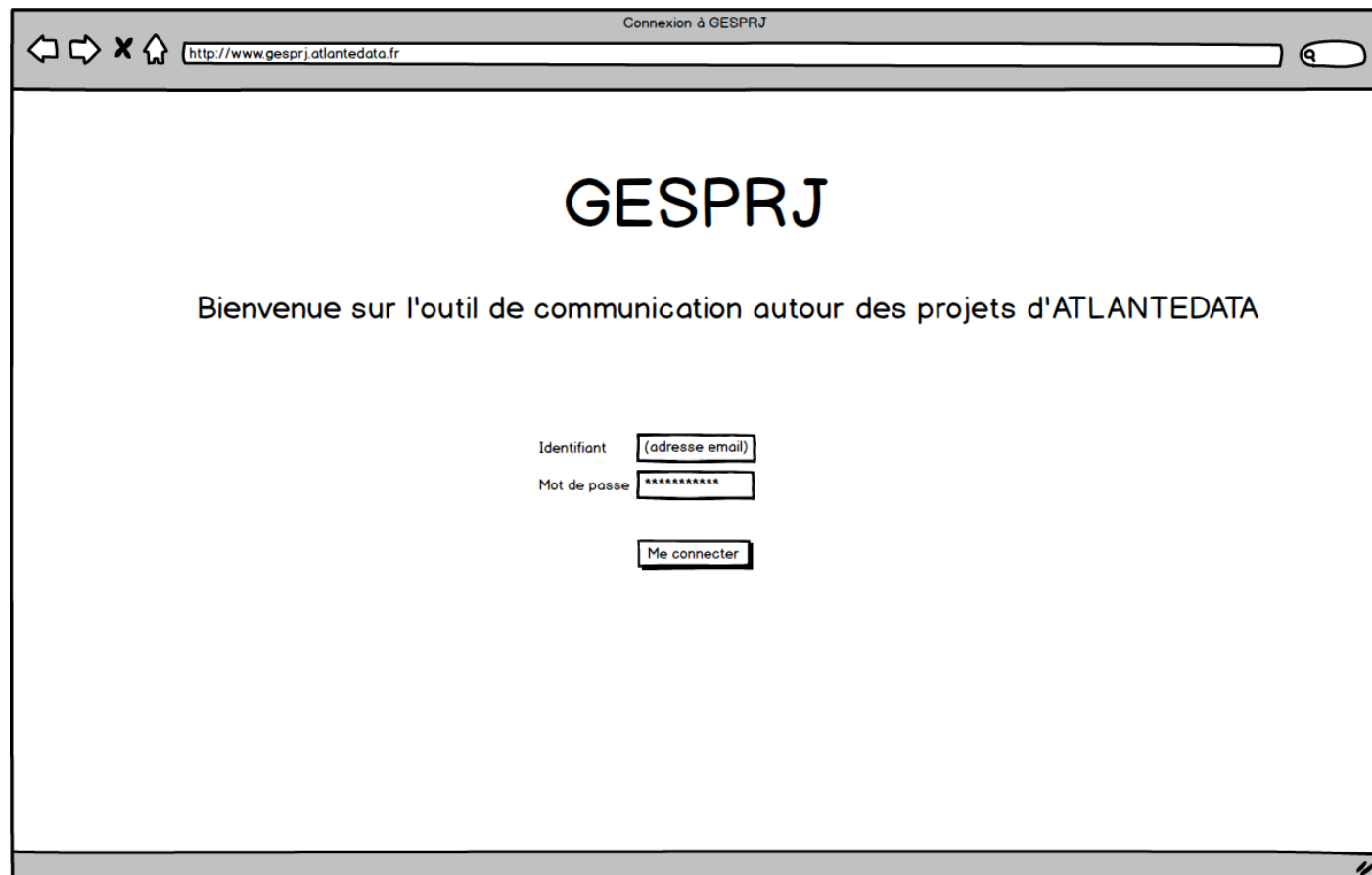
Voici donc un exemple des spécifications réalisés pour la page « création ou modification de développement », sachant qu'on part du principe que seuls les utilisateurs affiliés au développement (Chef de projet du projet dont fait partie le développement, Employé Client lié au développement, Développeur affilié au développement) peuvent accéder à la page de ce développement.

- Tous les utilisateurs peuvent voir les informations principales d'un développement
- Le chef de projet peut modifier les informations principales d'un développement :
 - o Libelle
 - o Employé(s) client
 - o Projet (si le projet est changé, il faudra aussi changer le dossier affilié. On ne peut choisir un projet que s'il a des dossiers affiliés)
 - o Dossier
 - o Priorité
 - o Développeur(s)
- Le développeur et le chef de projet peuvent valider le développement
- Le chef de projet peut abandonner le développement
- Tous les utilisateurs peuvent voir la description du développement
- Le chef de projet peut modifier la description du développement
- Le chef de projet et le développeur peuvent voir la liste des tâches
- Le chef de projet peut ajouter ou modifier une tâche
- Le chef de projet et le développeur peuvent signaler qu'ils ont effectué la tâche.
- Tous les utilisateurs peuvent voir les commentaires
- Tous les utilisateurs peuvent modifier les commentaires qu'ils ont effectués
- Tous les utilisateurs peuvent ajouter un commentaire
- Tous les utilisateurs peuvent accéder à la GED du développement
- Tous les utilisateurs peuvent accéder à la documentation du développement

MAQUETTAGE

Ci-dessous les maquettes réalisées pour les principaux écrans. Elles ont été réalisées grâce à l'outil *Balsamiq* disponible sur *Internet*.

CONNEXION



Connexion à GESPRJ

<http://www.gesprj.atlantedata.fr>

GESPRJ

Bienvenue sur l'outil de communication autour des projets d'ATLANTEDATA

Identifiant

Mot de passe

ECRAN D'ACCUEIL CHEF DE PROJET

[illegible]

ONGLET "Liste des développements à valider"

[illegible]

Clic droit sur un des projets, ouverture d'un menu avec "Accéder à ce projet" qui permet d'afficher la page du projet sélectionné.

Clic droit sur un des SAV/ Réalisation, ouverture d'un menu avec "Accéder à ce SAV" (ou "réalisation"), qui permet d'afficher la page du SAV ou de la réalisation

ONGLET "Liste des demandes clients"

Nom	Client	Projet	Dossier	Real	SAV
Demande 10	A	Projet 1	Dossier 2	☒	☐
Demande 20	A	Projet 1	Dossier 4	☐	☒
Demande 30	C	Projet 3	Dossier 1	☒	☐
Nouveau projet	E			☐	☐
Nouveau dossier	A	Projet 3		☐	☐

Même principe de clic doit que pour les autres tableaux

GESPRJ

ACCUEIL - Développeur - Mr Dev2

Mon compte Alertes (2)

Recherche Déconnexion

Menu

- [Accueil](#)
- [Projets](#)
- [Mon planning](#)
- [Historique](#)

Liste des projets en cours (4) Liste des réalisations en cours (4) Liste des SAV en cours (4)

Projet	Clients	Priorité	Chef de projet	Développeur(s)	Date de lancement
Projet 1	A	5	CP1	Dev1/Dev2	10/08/2015
Projet 2	A	3	CP1	Dev2	11/08/2015
Projet 4	A	2	CP2	Dev2/Dev3	12/08/2015
Projet 6	A	2	CP1	Dev2	13/08/2015

Le lien "historique" permet d'afficher la page des projets/dossiers/réalisation qui ont été soit clôturés soit abandonnés.

Même principe de clic droit que pour les autres tableaux

ONGLET "Liste des réalisations en cours"

[illegible]

ONGLET "Liste des SAV en cours"

[illegible]

Accueil - Client

[←](#) [→](#) [X](#) [🏠](#)
http://www.gesprj.atlantedata.fr/accueil/client
🔍

GESPRJ

ACCUEIL - CLIENT - Mr Client client

Mon compte
Alertes (2)

Recherche
Déconnexion

Menu

- [Accueil](#)
- [Projets](#)
- [Demandes](#)
- [Faire une demande](#)

- [Historique](#)

Liste des projets en cours (4)
Liste des dossiers en cours (2)
Liste des demandes effectuées(5)

Nom	Chef de projet	Date de lancement	Date de clôture souhaitée
Projet 1	CP1	11/08/2015	15/08/2015
Projet 3	CP1	10/05/2015	20/05/2015
Projet 5	CP1	01/03/2015	01/08/2015
Projet 7	CP1	21/06/2015	10/08/2015

Le lien "historique" permet d'afficher la page des projets/dossiers/réalisation qui ont été soit clôturés soit abandonnés.

Clic droit sur un des projets, ouverture d'un menu avec "Accéder à ce projet" qui permet d'afficher la page du projet sélectionné.

De plus, comme c'est un utilisateur "Client", une option "Créer une demande sur ce projet" est proposée.

ONGLET "Liste des demandes effectuées"

[illegible]

ONGLET "Liste des dossiers en cours"

[illegible]

PAGE RECHERCHE

Pour le client, les zones "Chef de projet", "Développeurs" et "Priorité" ne sont pas visibles.
Le tableau n'affiche pas les colonnes "Client" et "priorité"

Même principe de clic droit que pour les autres tableaux

Pour un client, une option "Créer une demande sur cette entité" est proposée.

[illegible][illegible]

PAGE PROJET

(Cette page est identique pour la création de projet et la modification de projet)

Projet - Informations principales

http://www.gesprj.atlantedata.fr/projet/info

GESPRJ

PROJET - Nom du projet

Mon compte | Alertes (2)
Recherche | Déconnexion

Menu
Accueil
Projets
Créer un projet
Demandes
Validation de développements
Administration
Plannings
Gestion des clients
Gestion des développeurs
Créer un compte utilisateur

Informations principales | GED (7) | Documentation

Libellé: Nom du projet | Priorité: 3 | Date de création: / / | Clôturer ce projet
Date de lancement: 10/08/2015 | Développeur(s): Dev1, Dev2 | Date de clôture: / / | Abandonner ce projet
Client: Client A

Description

Un chef de projet peut directement modifier toutes les informations (et seulement lui), sauf pour la zone "Description", où il doit cliquer sur le bouton correspondant. Pour la zone client, une alerte "Êtes-vous sûr de vouloir modifier le client attaché ?" apparaît.

Là on est en mode "le chef de projet a cliqué sur l'icône de modification de la description". Un outil de création de texte HTML s'affiche à la place de la description (et la description est chargée). Quand il clique sur "Valider la description", ça affiche la nouvelle description.

Ces boutons ne sont visibles que si l'utilisateur est Chef de projet.

Validation de la modification du projet

Liste des dossiers

Dossier	Priorité	Développeur(s)
Dossier 1	5	Dev1
Dossier 2	3	Dev2

Ajouter un dossier

Quand on sélectionne un dossier, la liste des réalisations/SAV's rattachés se met à jour.

Liste des réalisations/SAV

Libellé	Type	Priorité	Développeur(s)	Validé(e)
Réalisation 1	Réalisation	5	Dev1/Dev2	☒
Réalisation 2	Réalisation	3	Dev2	☐
Réalisation 3	Réalisation	2	Dev2/Dev3	☒
SAV 1	SAV	5	Dev2/Dev3	☐

Ajouter une réalisation ou un SAV

Même principe de clic droit que pour les autres tableaux.

Commentaires

Titre du commentaire: Par CP1 le 17/08/2015 à 15:00
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Titre du commentaire: Par Client A le 17/08/2015 à 16:00
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Poster un commentaire

Poster le commentaire

Avertir par email les utilisateurs concernés | Liste destinataires

Ce bouton ouvre une fenêtre pop-up et permet de sélectionner les destinataires de l'avertissement. Par défaut, tous les utilisateurs liés à cette entité seront avertis.

Attention

Tous les dossiers ne sont pas clôturés. Si vous confirmez, ils seront clôturés automatiquement (ainsi que les SAV/réalisations liés à ces dossiers). Êtes-vous sûr de vouloir clôturer ce projet ?

Non | Oui

Clôture

Êtes-vous sûr de vouloir clôturer ce projet ?

Non | Oui

Abandon

Tous les dossiers, réalisations et SAV liés à ce projet seront abandonnés. Êtes-vous sûr de vouloir abandonner ce projet ?

Non | Oui

Comme indiqué, si l'abandon est confirmé, toutes les réalisations liées seront abandonnées.

Le bouton "valider" permet de valider les modifications (ou la création) d'un projet. S'il manque des zones obligatoires :
- libellé
- client
La validation ne s'effectue pas et on signale les manquements à l'utilisateur par "Informations manquantes" associé à des astérisques rouges à côté des champs manquants.
Ce bouton n'est visible que pour le chef de projet.

34

PAGE DEVELOPPEMENT (REALISATION/SAV)

(Cette page est identique pour la création de réalisation/SAV et la modification de réalisation/SAV)

Realisation - Informations principales

GESPRJ

REALISATION - Nom de la réalisation

Mon compte | Alertes (2)

Recherche | Déconnexion

Menu

- Accueil
- Projets
- Créer un projet
- Demandes
- Validation de développement

Administration

- Plannings
- Gestion des clients
- Gestion des développeurs
- Créer un compte utilisateur

Informations principales | GED (1) | Documentation

Libellé: Nom de la réalisation

Client: Client A

Projet: Projet A

Dossier: Dossier 1

Priorité: 2

Développeur(s): Dev1

Date de création: / /

Date de clôture souhaitée: / /

Date de clôture effective: / /

Valider cette réalisation

Abandonner cette réalisation

Ajouter un développeur

Description

Un chef de projet peut directement modifier toutes les informations (et seulement lui).

S'il modifie la zone client, une alerte "Êtes-vous sûr de vouloir modifier le client attaché ?" apparaît.

S'il dit oui:

- la combobox Dossier est vidée
- la combobox Projet se recharge avec les projets du nouveau client et n'a aucun projet de choisi par défaut.

S'il modifie la zone Projet:

- la combobox Dossier se recharge avec les dossiers du nouveau projet et n'a aucun dossier de choisi par défaut.

Là on est en mode "le chef de projet a cliqué sur l'icône de modification de la description". Un outil de création/modification de texte HTML s'affiche à la place de la description normalement affichée dans une zone de texte classique. Quand il clique sur "Valider la description", ça affiche la nouvelle description.

por incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Valider les modifications de la réalisation

Liste des tâches

Nom	Durée	Développeur assigné	Effectuée	le	Par
Tâche 1	2 jours	Dev1	☐		
Tâche 2	1 jours	Dev1	☒	10/08/2015	Dev1

Le client ne visualisera pas cette liste de tâche

Seulement pour un développeur, un clic droit sur une des tâches permet d'afficher un menu avec "Tâche effectuée" ou "Tâche non effectuée" si elle est déjà en effectuée. Si il clique dessus, la tâche est donc soit affichée comme Effectuée et le nom du développeur est mis dans "par", soit la zone "par" est vidée et la combobox n'est plus cochée.

Ajouter une tâche

Commentaires

Titre du commentaire: Par CP1 le 17/08/2015 à 15:00

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Titre du commentaire: Par Client A le 17/08/2015 à 16:00

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Poster un commentaire

Avatar | Texte | Boutons de formatage

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

☐ Avertir par email les utilisateurs concernés

Liste destinataires

Poster le commentaire

Ce bouton ouvre une fenêtre pop-up et permet de sélectionner les destinataires de l'avertissement. Par défaut, tous les utilisateurs liés à cette entité seront avertis.

Attention

Tous les tâches ne sont pas effectuées. Si vous confirmez, elles le seront automatiquement. Êtes vous sûr de vouloir valider cette réalisation?

Non | Oui

Validation

Êtes vous sûr de vouloir valider cette réalisation ?

Non | Oui

Abandon

Êtes vous sûr de vouloir abandonner cette réalisation?

Non | Oui

Comme indiqué, si l'abandon est confirmé, toutes les réalisations liés seront abandonnées.

Le bouton "valider" permet de valider les modifications (ou la création) d'un projet. S'il manque des zones obligatoires d'un projet. S'il manque des zones obligatoires:

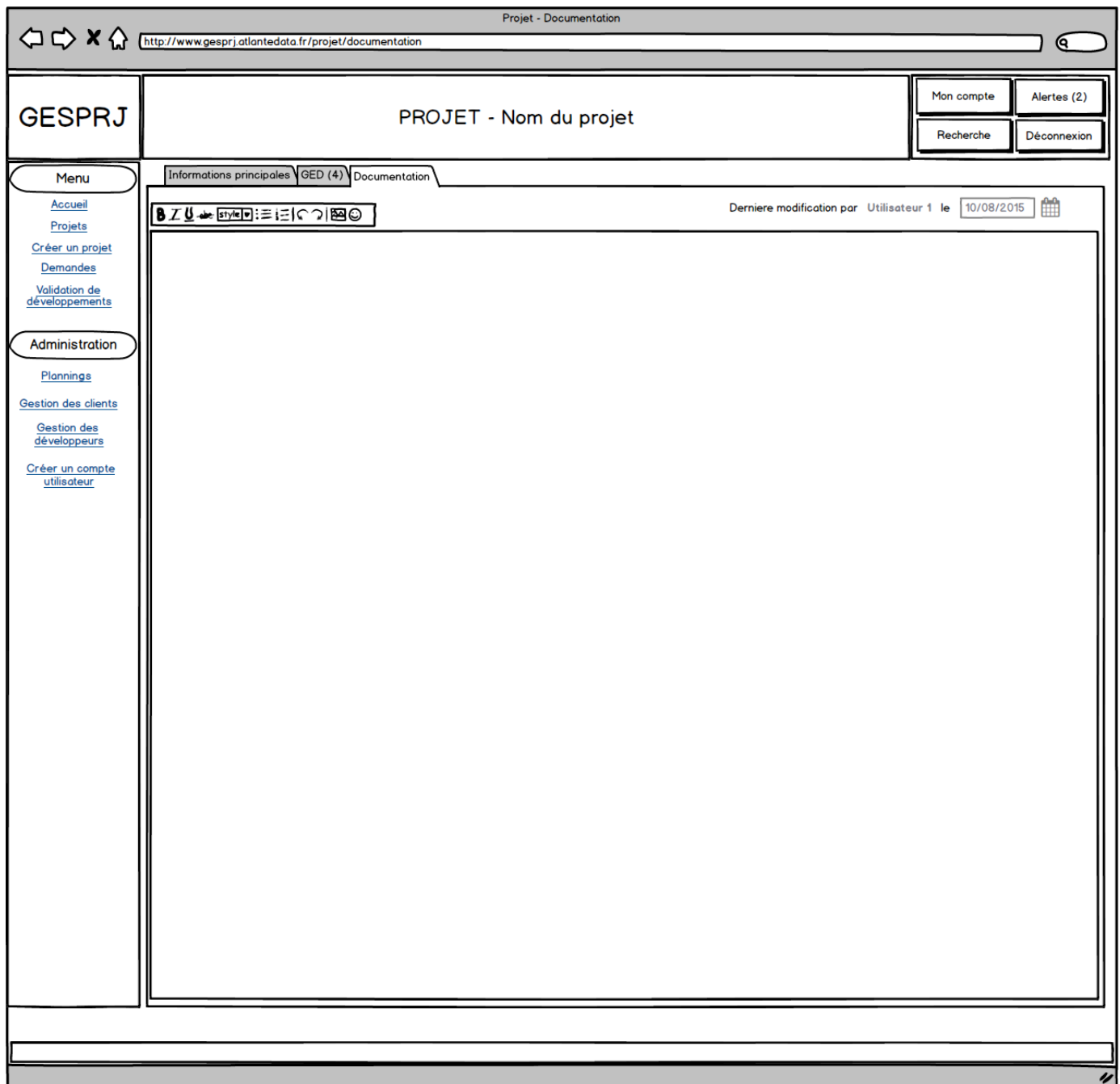
- libellé
- client
- projet
- dossier

La validation ne s'effectue pas et on signale les manquements à l'utilisateur par "Informations manquantes" associé à des astérisques rouges à côté des champs manquants.

Ce bouton n'est visible que pour le chef de projet

une fenêtre s'ouvre et demande:

- le nom de la tâche,
- la durée
- le développeur assigné (combobox)



Projet - GED

http://www.gesprj.atlantedata.fr/projet/GED

GESPRJ **PROJET - Nom du projet** Mon compte Alertes (2)
Recherche Déconnexion

Menu
[Accueil](#)
[Projets](#)
[Créer un projet](#)
[Demandes](#)
[Validation de développements](#)
Administration
[Plannings](#)
[Gestion des clients](#)
[Gestion des développeurs](#)
[Créer un compte utilisateur](#)

Informations principales **GED (4)** **Documentation**

Liste des documents Ajouter un document

DOC	DOC	XLS	PDF
document1.doc Ceci est une description du document1.doc. Elle a été mise par l'utilisateur qui a mis le document dans la GED Ajouté par : CLIENT 1 Le : 18/08/2015 à : 15:45	document2.doc Ceci est une description du document2.doc. Elle a été mise par l'utilisateur qui a mis le document dans la GED Ajouté par : DEV 1 Le : 18/08/2015 à : 15:50	tableur1.xls Ceci est une description du tableur1.xls. Elle a été mise par l'utilisateur qui a mis le document dans la GED Ajouté par : CLIENT 1 Le : 18/08/2015 à : 15:46	pdf1.pdf Ceci est une description du pdf1.pdf. Elle a été mise par l'utilisateur qui a mis le document dans la GED Ajouté par : CP1 Le : 18/08/2015 à : 16:00

On peut sélectionner un des documents Télécharger le document sélectionné

Aperçu du document Aperçu du document

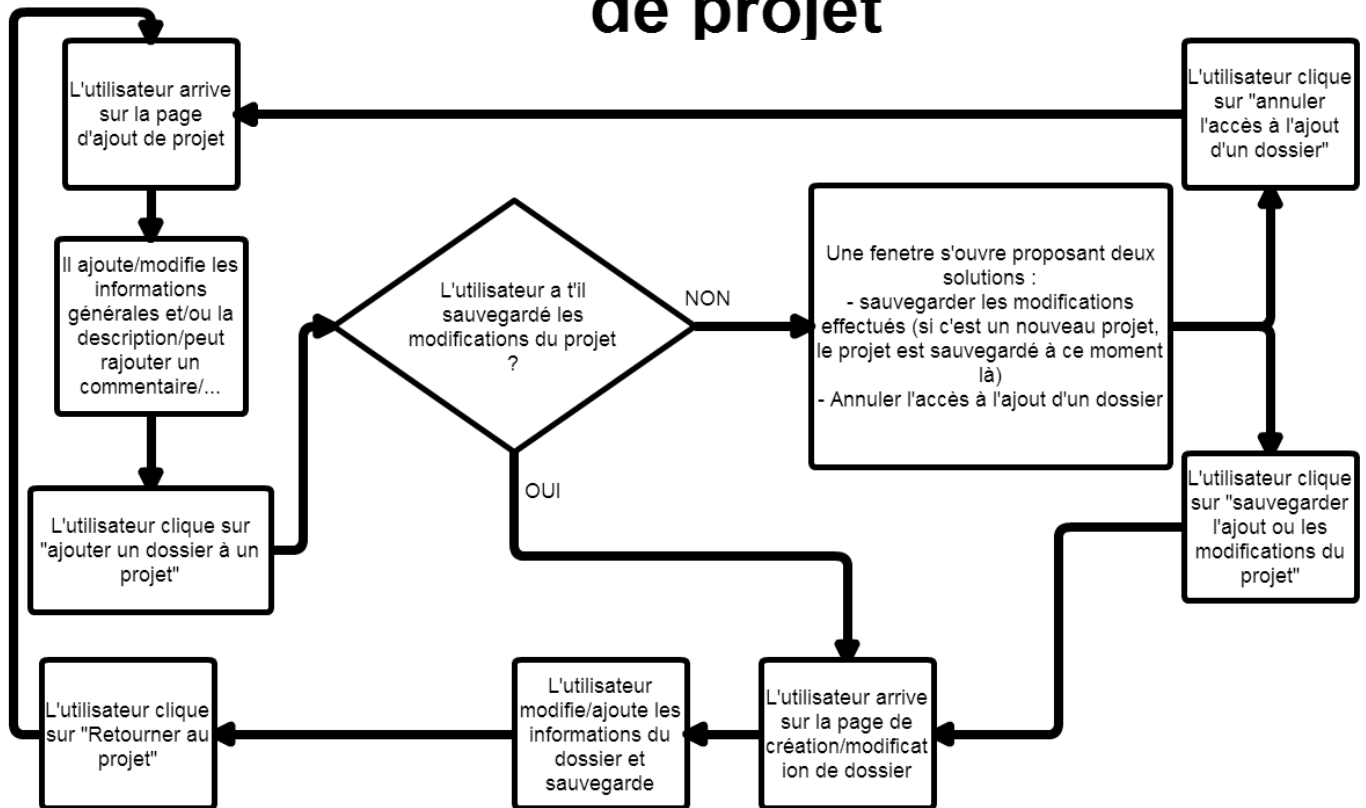
Mise à jour : L'application « Balsamiq » que j'utilisais était une version d'essai, je n'ai plus eu accès à mes maquettes pour montrer les choses suivantes (dont je n'avais pas sauvegardé l'image après modification) :

- le clic droit sur une ligne d'un tableau pour accéder, par exemple, au dossier sélectionné, n'est plus la solution retenue, remplacé par des boutons « Accéder à ... » sur chaque ligne des tableaux.
- Rajout du bouton « Retourner à » sur les pages de visualisation de dossier/développement
- Tableaux de clients plutôt qu'un combobox sur les pages de visualisation de projet/dossier/développements
- Séparer le bouton « Valider les modifications du ... » dans les pages de visualisation de projet/dossier/développements en deux boutons : « Valider les modifications sur les informations principales », positionné en bas à droite du bloc des informations principales, et « Valider la description » en bas à droite du bloc de la description.
- Indiquer « Développement » plutôt que « Réalisation »

DIAGRAMME DE NAVIGATION

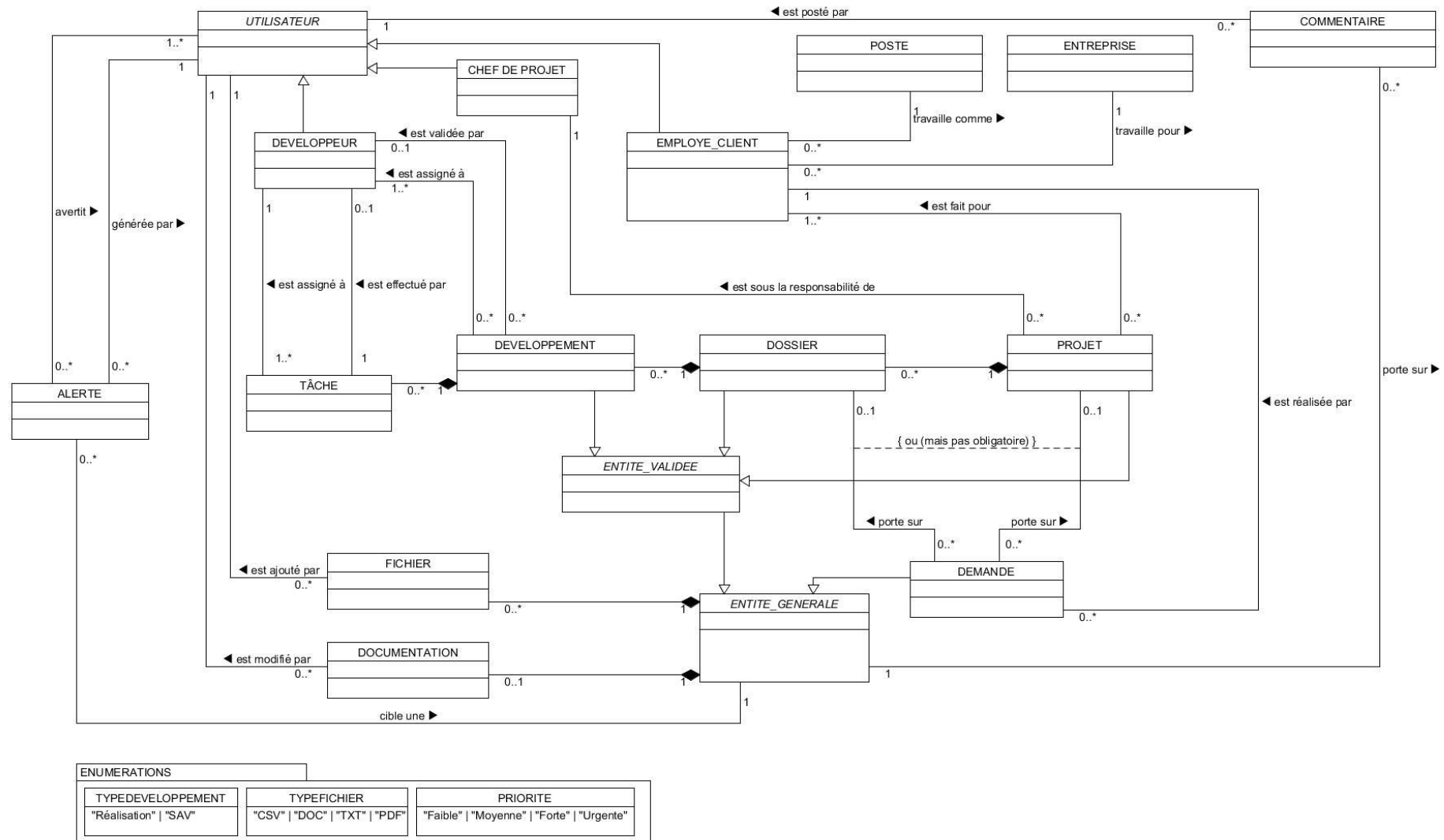
Pour clarifier la navigation entre les pages et pour être certain de son enchainement, j'ai réalisé quelques diagrammes de navigation, donc en voici un exemple :

Ajout d'un dossier suite à un ajout de projet



PARTIE 4 : ANALYSE TECHNIQUE

DIAGRAMME DES CLASSES METIERS



ALERTE
- id : int - dateVu : Date - dateTraitee : Date - dateCreation : Date - utilisateurCreation : Utilisateur - listeUtilisateursCibles : liste<Utilisateur> - entiteGeneraleLiee : EntiteGenerale
ASSESEURS + getId() : int + getDateVu() : Date + getDateTraitee() : Date + getDateCreation() : Date + getUtilisateurCreation() : Utilisateur + getListeUtilisateursCibles() : liste<Utilisateur> + getEntiteGeneraleLiee() : EntiteGenerale + setId(id:int) : void + setDateVu(dateVu:Date) : void + setDateTraitee(dateTraitee:Date) : void + setDateCreation(dateCreation:Date) : void + setUtilisateurCreation(utilisateurCreation:Utilisateur) : void + setListeUtilisateursCibles(listeUtilisateursCible:liste<Utilisateur>) : void + setEntiteGeneraleLiee(entiteGeneraleLiee:EntiteGenerale) : void

CHEF DE PROJET
- listeProjetsEnCours : liste<Projet>
ASSESEURS + getListeProjetsEnCours() : liste<Projets> + getTypeUtilisateur() : TypeUtilisateur + setListeProjetsEnCours(listeProjets:liste<Projet>) : void

COMMENTAIRE
- id : Integer - titre : String - texte : String - dateCreation : Date - dateModification : Date - utilisateurCreation : Utilisateur - entiteGeneraleLiee : EntiteGenerale
ASSESEURS + getId() : Integer + getTitre() : String + getTexte() : String + getDateCreation() : Date + getDateModification() : Date + getUtilisateurCreation() : Utilisateur + getEntiteGeneraleLiee() : EntiteGenerale + setId(id:Integer) : void + setTitre(titre:String) : void + setTexte(texte:String) : void + setDateCreation(dateCreation:Date) : void + setDateModification(dateModification:Date) : void + setUtilisateurCreation(utilisateurCreation:Utilisateur) : void + setEntiteGeneraleLiee(entiteGeneraleLiee:EntiteGenerale) : void

DEMANDE
- dateReponseSouhaitee : Date - dateValidation : Date - ChefProjetValidation : ChefProjet - dateRefus : Date - ChefProjetRefus : ChefProjet - EmployeClientDemandeur : EmployeClient
ASSESEURS + getDateReponseSouhaitee() : Date + getDateValidation() : Date + getChefProjetValidation() : ChefProjet + getDateRefus() : Date + getChefProjetRefus() : ChefProjet + getEmployeClientDemandeur() : EmployeClient + setDateReponseSouhaitee(dateReponseSouhaitee:Date) : void + setDateValidation(dateValidation:Date) : void + setChefProjetValidation(chefProjetValidation:ChefProjet) : void + setDateRefus(dateRefus:Date) : void + setChefProjetRefus(chefProjetRefus:ChefProjet) : void + setEmployeClientDemandeur(employeClient:EmployeClient) : void

DEVELOPPEMENT
- typeDeveloppement : typeDeveloppement - listeTaches : liste<Tache> - listeDeveloppeurs : liste<Developpeur>
ASSESEURS + getTypeDeveloppement() : typeDeveloppement + getListeTaches() : liste<Tache> + getListeDeveloppeurs : liste<Developpeur> + setTypeDeveloppement(typeDeveloppement:typeDeveloppement) : void + setListeTache(listeTache:liste<Tache>) : void + setListeDeveloppeurs(listeDeveloppeurs:liste<Developpeur>) : void

DEVELOPPEUR
ATTRIBUTS - listeDevsAssignes : liste<Developpement> - listeTachesAssigne : liste<Tache>
ASSESEURS + getListeDevsAssigne() : liste<Developpement> + getListeTachesAssigne() : liste<Tache> + getTypeUtilisateur() : TypeUtilisateur + setListeDevsAssigne(listeDevsAssigne:liste<Developpement>) : void + setListeTachesAssigne(listeTachesAssigne:liste<Tache>) : void

DOCUMENTATION
- id : int - libelle : String - texte : String - dateDerniereModification : Date - dateCreation : Date - utilisateurCreation : Utilisateur - utilisateurDerniereModification : Utilisateur - entiteGeneraleLiee : EntiteGenerale
ASSESEURS + getId() : int + getLibelle() : String + getTexte() : String + getDateDerniereModification() : Date + getDateCreation() : Date + getUtilisateurCreation() : Utilisateur + getUtilisateurDerniereModification() : Utilisateur + getEntiteGeneraleLiee() : EntiteGenerale + setId(id:int) : void + setLibelle(libelle:String) : void + setTexte(texte:String) : void + setDateDerniereModification(dateDerniereModification:Date) : void + setDateCreation(dateCreation:Date) : void + setUtilisateurCreation(utilisateurCreation:Utilisateur) : void + setUtilisateurDerniereModification(utilisateurDerniereModif:Date) : void + setEntiteGeneraleLiee(entiteGeneraleLiee:EntiteGenerale) : void

DOSSIER
- projetLie : Projet - listeDeveloppements : liste<Developpement>
ASSESEURS + getProjetLie() : Projet + getListeDeveloppements() : liste<Developpement> + setProjetLie(projetLie:Projet) : void + setListeDeveloppements(listeDeveloppement:liste<Developpement>) : void

EMPLOYE_CLIENT
- telephoneFixe : String - telephonePortable : String
ASSESEURS + getTelephoneFixe() : String + getTelephonePortable() : String + getTypeUtilisateur() : TypeUtilisateur + setTelephoneFixe(telephoneFixe:String) : void + setTelephonePortable(telephonePortable:String) : void

ENTITE_GENERALE
id: Integer # libelle: String # description: String # dateCreation: Date # listeCommentaire: liste<Commentaire> # listeFichier: liste<Fichier> # listeDocumentation: liste<Documentation>
ASSESEURS + getId() : String + getLibelle() : String + getDescription() : String + getDateCreation() : Date + getListeCommentaire() : liste<Commentaire> + getListeFichier() : liste<Fichier> + getListeDocumentation() : liste<Documentation> + setId(id:Integer) : void + setLibelle(libelle:String) : void + setDescription(description:String) : void + setDateCreation(dateCreation:Date) : void + setListeCommentaire(listeCommentaire:liste<Commentaire>) : void + setListeFichier(listeFichier:liste<Fichier>) : void + setListeDocumentation(listeDocumentation:liste<Documentation>) : void

ENTITE_VALIDEE
- priorite : Priorite - dateClotureSouhaitee : Date - dateClotureEffective : Date - dateValidation : Date - dateAbandon : Date - entiteValideeMere : EntiteValidee
ASSESEURS + getPriorite() : Priorite + getDateClotureSouhaitee() : Date + getDateClotureEffective() : Date + getDateValidation() : Date + getDateAbandon() : Date + getEntiteValideeMere() : EntiteValidee + setPriorite(priorite:Priorite) : void + setDateClotureSouhaitee(dateClotureSouhaitee:Date) : void + setDateClotureEffective(dateClotureEffective:Date) : void + setDateValidation(dateValidation:Date) : void + setDateAbandon(dateAbandon : Date) : void + setEntiteValideeMere(entiteValideeMere : EntiteValidee) : void

ENTREPRISE
- id : int - libelle : String - adresse1 : String - adresse2 : String - codePostal : String - ville : String - pays : String - telephoneFixe : String
ASSESEURS + getId() : int + getLibelle() : String + getAdresse1() : String + getAdresse2() : String + getCodePostal() : String + getVille() : String + getPays() : String + getTelephoneFixe() : String + setId(id:int) : void + setLibelle(libelle:String) : void + setAdresse1(adresse1:String) : void + setAdresse2(adresse2:String) : void + setCodePostal(codePostal:String) : void + setVille(ville:String) : void + setPays(pays:String) : void + setTelephoneFixe(telephoneFixe:String) : void

FICHIER - id : int - chemin : String - dateAjout : Date - typeFichier : TypeFichier - utilisateurAjout : Utilisateur - entiteGeneraleLiee : EntiteGenerale ASSESEURS + getId() : int + getChemin() : String + getDateAjout() : Date + getTypeFichier() : typeFichier + getUtilisateurAjout : Utilisateur + getEntiteGeneraleLiee : EntiteGenerale + setId(id:int) : void + setChemin(chemin:String) : void + setDateAjout(dateAjout:date) : void + setTypeFichier(typeFichier:TypeFichier) : void + setUtilisateurAjout(utilisateurAjout:Utilisateur) : void + setEntiteGeneraleLiee(entiteGeneraleLiee:EntiteGenerale) : void	POSTE - id : int - libelle : String ASSESEURS + getId() : int + getLibelle() : String + setId(id:int) : void + setLibelle(libelle:String) : void	PROJET - dateLancement : Date - chefProjet : ChefProjet - listeEmployesClients : liste<EmployeClient> - listeDossiers : liste<Dossier> - listeDemande : liste<Demande> ASSESEURS + getDateLancement() : Date + getChefProjet() : ChefProjet + getListeEmployesClients() : liste<EmployeClient> + getListeDossier() : liste<Dossier> + getListeDemande() : liste<Demande> + setDateLancement(dateLancement:Date) : void + setChefProjet(chefProjet:ChefProjet) : void + setListeEmployesClients(listeEmployesClients:liste<EmployeClient>) : void + setListeDossier(listeDossier:liste<Dossier>) : void + setListeDemande(listeDemande:liste<Demande>) : void
TACHE - id: Integer - libelle: String - duree : Integer - dateCreation : Date - dateEffectuee : Date - utilisateurAssigne : Utilisateur - utilisateurEffectue : Utilisateur - developpementLie : Developpement ASSESEURS + getId() : String + getLibelle() : String + getDuree() : Integer + getDateCreation() : Date + getEffectuee() : Boolean + getDateEffectuee() : Date + getUtilisateurAssigne() : Utilisateur + getUtilisateurEffectue() : Utilisateur + getDeveloppementLie() : Developpement + setId(id:Integer) : void + setLibelle(libelle:String) : void + setDuree(duree:Integer) : void + setDateCreation(dateCreation:Date) : void + setDateEffectuee(dateEffectuee:Date) : void + setUtilisateurAssigne(utilisateurAssigne:Utilisateur) : void + setUtilisateurEffectue(utilisateurEffectue:Utilisateur) : void + setDeveloppementLie(developpementLie:Developpement) : void	UTILISATEUR ATTRIBUTS # id : int # email : String # password : String # nom : String # prenom : String # dateAbandon : Date # dateCreation : Date # listeAlerte : liste<Alerte> ASSESEURS + getId() : int + getEmail() : String + getPassword() : String + getNom() : String + getPrenom() : String + getDateCreation() : Date + getDateAbandon() : Date + getListeAlerte() : liste<Alerte> + setId(id:int) : void + setEmail(email:String) : void + setPassword(password:String) : void + setNom(nom:String) : void + setPrenom(prenom:String) : void + setDateCreation(dateCreation:Date) : void + setDateAbandon(dateAbandon:Date) : void + setListeAlerte(listeAlerte:liste<Alerte>) : void	

Précisons ici que chaque classe a un ou plusieurs constructeurs (cela dépendra du langage ou de l'utilité), sauf *EntiteGenerale* et *EntiteValidee* qui sont des classes abstraites. Par contre, comme j'ai besoin d'instancier des objets Utilisateur (pour la connexion), j'ai décidé de ne pas la rendre abstraite.

LA BASE DE DONNEES

EXEMPLE DE DICTIONNAIRE DE DONNEES

J'ai réalisé le dictionnaire des données pour toutes les tables. En voici deux exemples :

TABLE « TACHE »

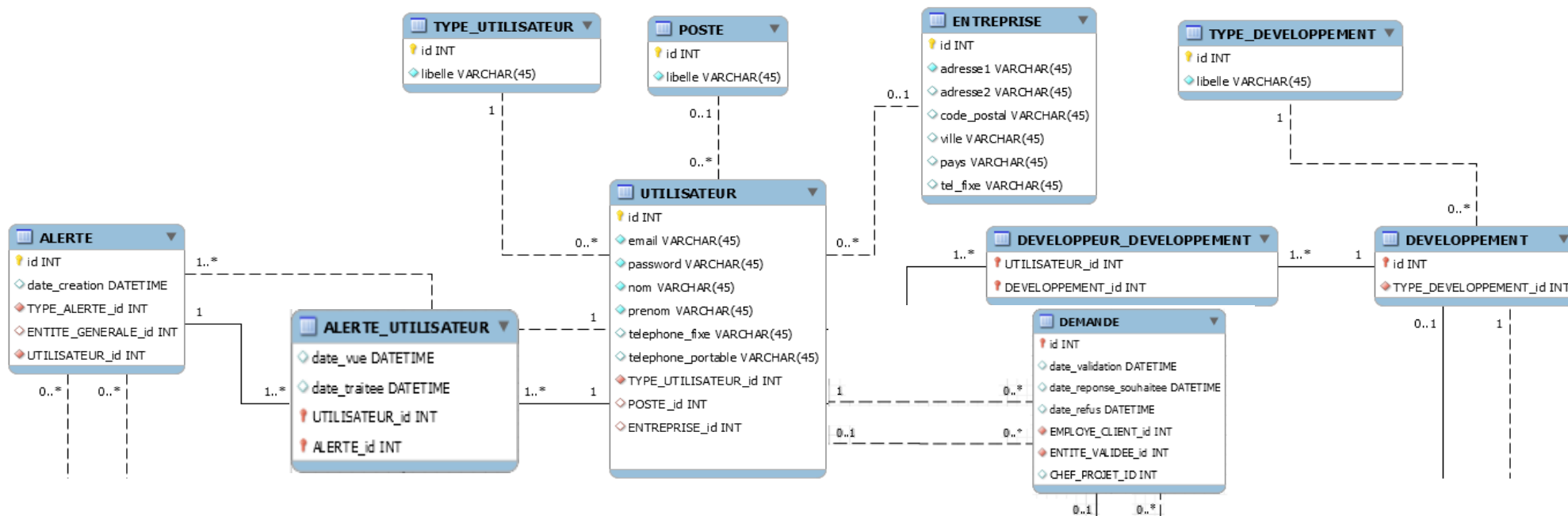
Colonne	Remarque	Type	Null ?	Défa ut	Commentaire
id	Clé primaire	INT	NOT NULL		Identifiant de la tâche
libelle		VARCHAR (45)	NOT NULL		Libelle de la tâche
durée		INT	NOT NULL		Durée de la tâche
date_effectuee		DATETIME	NULL		Date à laquelle la tâche a été effectuée
date_creation		DATETIME	NOT NULL	Date du jour	Date à laquelle la tâche a été créée
developpement_id	Clé étrangère	INT	NOT NULL		Id du développement auquel la tâche est rattachée
developpeur_assignment	Clé étrangère	INT	NULL		Id du développeur qui est assigné à la tâche
developpeur_realisation	Clé étrangère	INT	NULL		Id du développeur qui a réalisé la tâche

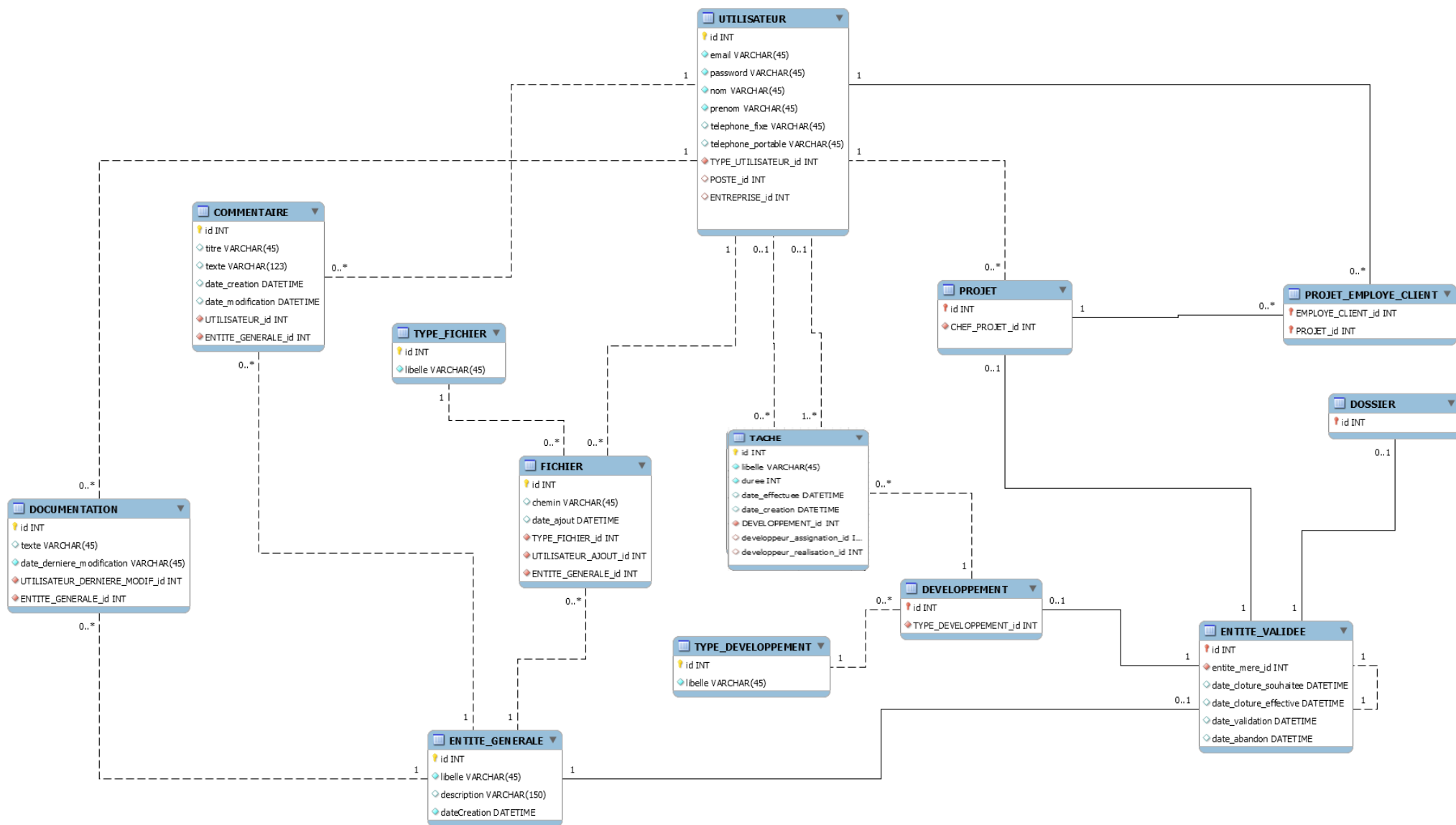
TABLE « UTILISATEUR »

Colonne	Remarque	Type	Null ?	Défa ut	Commentaire
id	Clé primaire	INT	NOT NULL		Identifiant de l'utilisateur
email	Unique	VARCHAR (45)	NOT NULL		Email de l'utilisateur (servira pour l'identification)
password		VARCHAR (45)	NOT NULL		Password de l'utilisateur
nom		VARCHAR (45)	NOT NULL		Nom de l'utilisateur
prenom		VARCHAR (45)	NOT NULL		Prénom de l'utilisateur
telephone_fixe		VARCHAR (45)	NULL		Téléphone fixe de l'utilisateur
telephone_portable		VARCHAR (45)	NULL		Téléphone portable de l'utilisateur
type_utilisateur_id	Clé étrangère	INT	NOT NULL		Type de l'utilisateur (Employé client, développeur ou chef de projet)
poste_id	Clé étrangère	INT	NULL		Poste de l'utilisateur
entreprise_id	Clé étrangère	INT	NULL		Entreprise de l'utilisateur

Pour la création de la base de données (MySQL pour rappel), j'ai utilisé l'outil MySQL Workbench 6.3. En partant de mon diagramme de classes, j'ai donc créé mon schéma de base de données afin qu'il génère un script complet. Comme le schéma est trop complet pour le mettre en entier, en voici 2 parties « découpés » comme exemple (pour améliorer la lisibilité, j'ai dû supprimer visuellement des liens.)

Je précise ici que j'ai choisi volontairement de ne pas utiliser de triggers et autres procédures métiers. En effet, je considère qu'il est préférable d'avoir du code métier dans l'application qui gère ça : au cas où la base de données viendrait à changer, il n'est pas nécessaire de réécrire les triggers et les procédures stockées qui auraient été créés.

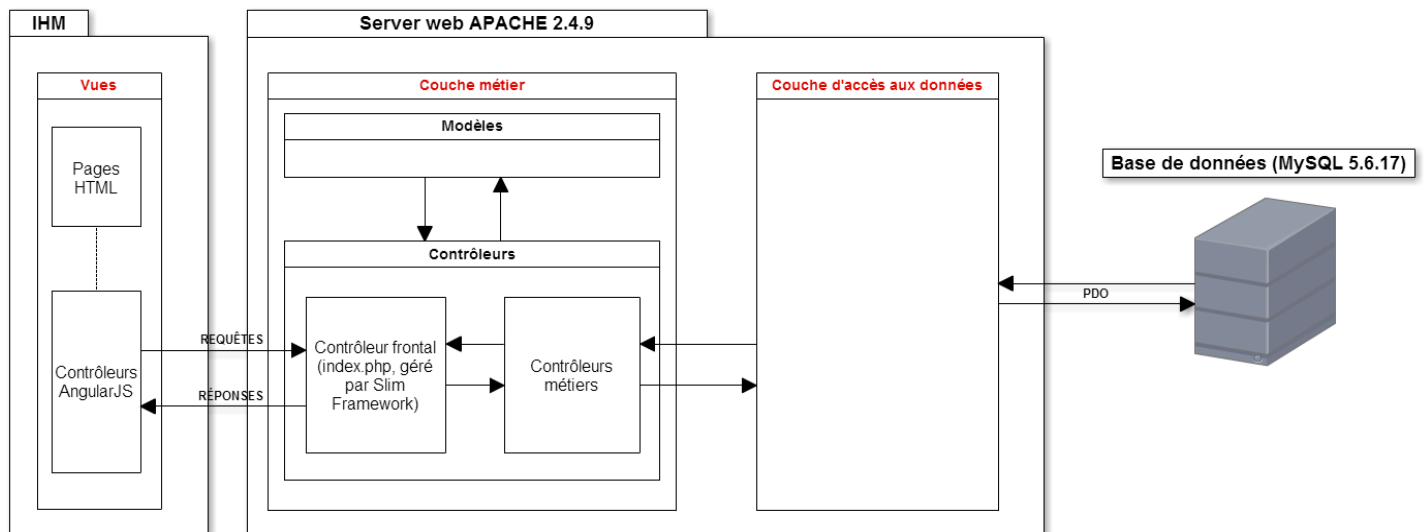




Ci-dessous des exemples de scripts (généré en partie par MySQL Workbench, vérifiés et corrigés par mes soins) :

```
CREATE TABLE IF NOT EXISTS `UTILISATEUR` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `email` VARCHAR(127) NOT NULL,  
  `password` VARCHAR(127) NOT NULL,  
  `nom` VARCHAR(127) NOT NULL,  
  `prenom` VARCHAR(127) NOT NULL,  
  `telephone_fixe` VARCHAR(31) NULL,  
  `telephone_portable` VARCHAR(31) NULL,  
  `TYPE_UTILISATEUR_id` INT NOT NULL,  
  `POSTE_id` INT NULL,  
  `ENTREPRISE_id` INT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `email_UNIQUE` (`email`),  
  INDEX `fk_UTILISATEUR_TYPE_UTILISATEUR_idx` (`TYPE_UTILISATEUR_id`),  
  INDEX `fk_UTILISATEUR_POSTE_idx` (`POSTE_id`),  
  INDEX `fk_UTILISATEUR_ENTREPRISE_idx` (`ENTREPRISE_id`),  
  CONSTRAINT `fk_UTILISATEUR_TYPE_UTILISATEUR` FOREIGN KEY (`TYPE_UTILISATEUR_id`) REFERENCES `TYPE_UTILISATEUR` (`id`),  
  CONSTRAINT `fk_UTILISATEUR_POSTE` FOREIGN KEY (`POSTE_id`) REFERENCES `POSTE` (`id`),  
  CONSTRAINT `fk_UTILISATEUR_ENTREPRISE` FOREIGN KEY (`ENTREPRISE_id`) REFERENCES `ENTREPRISE` (`id`))  
  
CREATE TABLE IF NOT EXISTS `TACHE` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `libelle` VARCHAR(45) NOT NULL,  
  `duree` INT,  
  `date_creation` DATETIME,  
  `date_effectuee` DATETIME,  
  `DEVELOPPEMENT_id` INT NOT NULL,  
  `developpeur_assignment_id` INT NULL,  
  `developpeur_realisation_id` INT NULL,  
  PRIMARY KEY (`id`),  
  INDEX `fk_TACHE_DEVELOPPEMENT1_idx` (`DEVELOPPEMENT_id` ASC) ,  
  INDEX `fk_TACHE_UTILISATEUR1_idx` (`developpeur_assignment_id` ASC) ,  
  INDEX `fk_TACHE_UTILISATEUR2_idx` (`developpeur_realisation_id` ASC) ,  
  CONSTRAINT `fk_TACHE_DEVELOPPEMENT` FOREIGN KEY (`DEVELOPPEMENT_id`) REFERENCES `DEVELOPPEMENT` (`id`),  
  CONSTRAINT `fk_TACHE_UTILISATEUR_ASSIGN` FOREIGN KEY (`developpeur_assignment_id`) REFERENCES `UTILISATEUR` (`id`),  
  CONSTRAINT `fk_TACHE_UTILISATEUR_REAL` FOREIGN KEY (`developpeur_realisation_id`) REFERENCES `UTILISATEUR` (`id`))
```

ARCHITECTURE LOGICIELLE



PARTIE 5 : RÉALISATION

CHOIX DE DEVELOPPEMENTS

CHOIX DES LANGAGES

Les langages de base que j'ai choisis pour le projet sont :

- PHP 5.5
- Javascript 1.8
- HTML 5
- CSS 3

CHOIX DES FRAMEWORKS

J'ai décidé de partir sur trois frameworks :

- Un framework PHP pour le côté serveur : Slim (version 3)
<http://www.slimframework.com/>
- Un framework Javascript pour le côté client : Angular JS (version 1.4.8)
<https://angularjs.org>
- Un « framework » pour le design : Bootstrap (version 3.3.6)
<http://getbootstrap.com/>

LOGICIELS ET AUTRES OUTILS

L'IDE : PHPSTORM

De l'avis de beaucoup le meilleur IDE pour des applications web. J'ai eu l'occasion de l'utiliser lors d'un petit projet à l'ENI, et il est très complet et assez facile à prendre en main.

GESTION DE VERSION : SUBVERSION

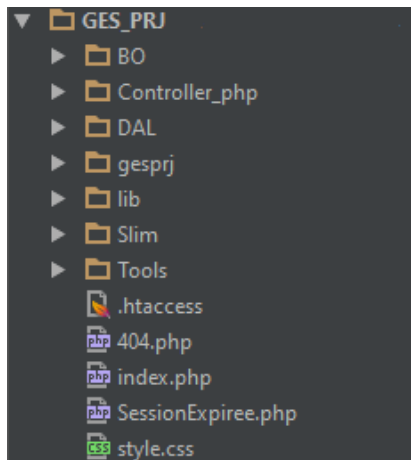
J'ai choisi Subversion, dont le plugin PhpStorm est facile à mettre en place et simple d'utilisation.

SERVEUR WEB : APACHE 2.4.9

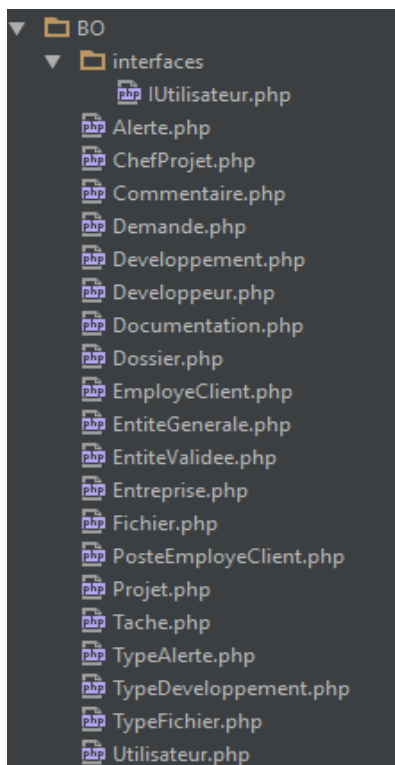
BASE DE DONNEES : MYSQL 5.6.17

L'ARBORESCENCE DU PROJET ET EXPLICATION

Ci-dessous une explication sommaire des différentes couches de l'application, qui seront détaillés dans les exemples suivant.



LA BO (BUSINESS OBJECT)

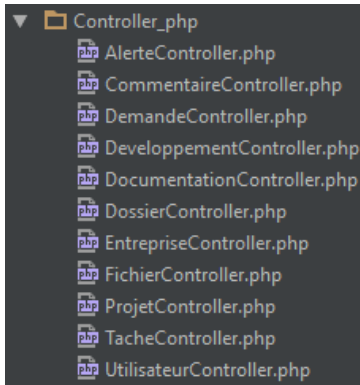


Le dossier BO contient les classes de chaque modèle prévu dans l'application.

De plus, une interface « IUtilisateur » permet d'imposer la méthode *getTypeUtilisateur* aux classes *ChefProjet*, *EmployeClient* et *Developpeur*.

Chaque classe contient les attributs et les méthodes qui ont été définies dans le diagramme des classes.

LE CONTROLLER_PHP

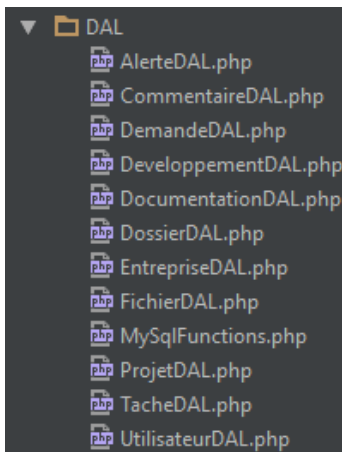


Le dossier CONTROLLER_PHP contient les contrôleurs qui vont s'occuper de la liaison entre le contrôleur frontal (défini plus bas) et la DAL (ou d'autres contrôleurs).

Ces contrôleurs vont recevoir de la part du contrôleur frontal des traitements à effectuer puis vont lui renvoyer les données souhaitées.

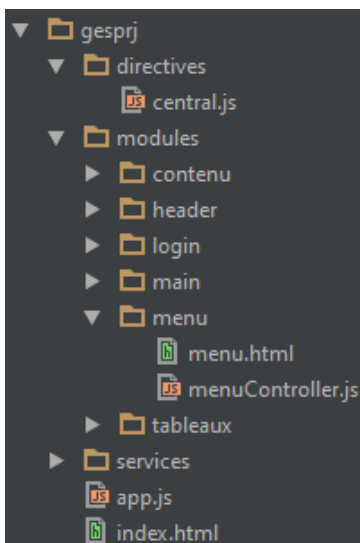
Ce ne sont donc pas eux qui vont directement recevoir la requête de la vue, ni fournir les données présentés comme il faut à la vue (voir partie sur le contrôleur frontal).

LA DAL (DATA ACCESS LAYER)



La DAL est la couche qui s'occupe de la récupération des données qui sont stockées sur la base de données MySQL.

LE DOSSIER « GESPRJ »



Le dossier « GESPRJ » contient la partie « vue » de l'application : les pages HTML et leurs contrôleurs Angular.

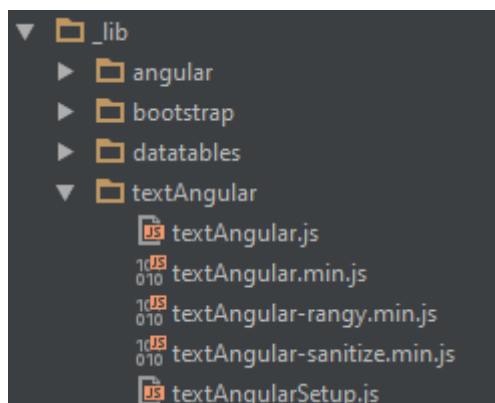
3 sous-dossiers sont présents :

- Le dossier « directives » qui contient les directives que j'ai créé (pour l'instant une seule). Cette directive sera expliquée en détail dans l'exemple 2 de réalisation.
- Le dossier « modules » qui contient les différentes grosses parties visuelles de l'application
- Le dossier « services » qui contient les services créés pour l'application.

En plus de ces 3 sous-dossiers, il y a l'*index.html* qui est la page « squelette ». C'est à elle qu'on va fournir le code HTML des modules que l'on veut afficher.

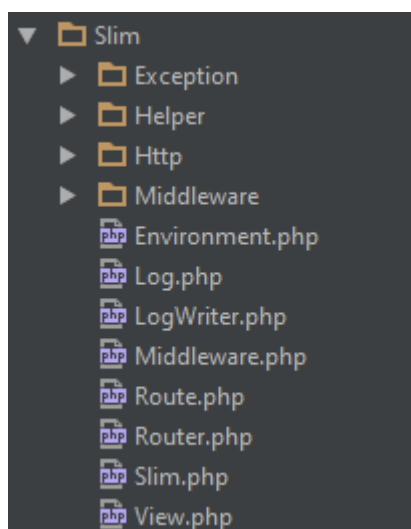
Enfin, le fichier *app.js* va contenir la définition générale de l'application du côté client.

LE DOSSIER « LIB »



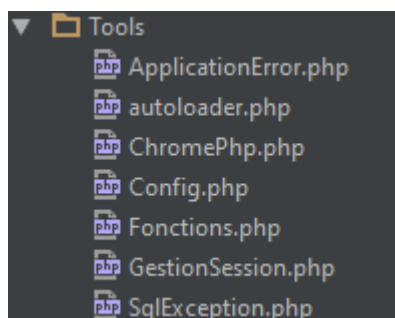
Le dossier *lib* contient toutes les bibliothèques nécessaires pour les différents composants utilisés dans l'application, comme le composant *datatables* pour les tableaux ou *textAngular* pour l'éditeur HTML intégré.

LE DOSSIER « SLIM »



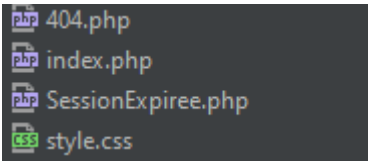
Le dossier *Slim* contient tous les fichiers nécessaires au fonctionnement du framework Slim. Il y avait juste à intégrer ces fichiers dans le dossier de l'application et de pointer sur la classe *Slim.php* dans la page *index.php*.

LE DOSSIER « TOOLS »



Le dossier *Tools* contient les classes d'outils ou de configuration pour l'application. Par exemple, *ChromePhp.php* contient une fonction qui permet d'afficher des logs dans la console du navigateur.

LA RACINE DU PROJET



A la racine du projet se trouvent :

- La page *404.php* qui affiche une page 404 quand la ressource demandée par l'utilisateur n'existe pas.
- La page *index.php* qui est le contrôleur côté serveur (avec le framework slim)
- La page *SessionExpiree.php* qui affiche une page informant l'utilisateur que sa session a expiré et qu'il doit recharger la page et se reconnecter.
- La feuille de style *style.css* qui contient du CSS.

Pour faciliter la lisibilité du code dans le dossier, j'ai remis toutes les constantes (par exemple les requêtes appelées dans la DAL directement dans chaque méthode. A l'origine, elles sont bien séparées et regroupées pour faciliter le développement.

Comme je n'ai pas la place de montrer toute la réalisation effectuée, et qu'au final, le principe de déroulement de l'application et des pages HTML est souvent le même, j'ai décidé de montrer les 2 exemples significatifs et les plus complets de l'application pour expliquer comment elle fonctionne :

- La connexion utilisateur
- La gestion de l'entité Projet par un chef de projet

EXEMPLE DE REALISATION N°1 : LA CONNEXION UTILISATEUR

Le premier exemple de réalisation va être sur la connexion utilisateur. Cela va me permettre de présenter les deux frameworks utilisés, Slim et Angular.

LA GESTION DE L'APPLICATION PAR SLIM

```
<?php

require './Slim/Slim.php';
\Slim\Slim::registerAutoloader();
$app = new \Slim\Slim();
$env = $app->environment();

// ===== AUTOLOADER CLASSES PHP =====
require_once './Tools/Config.php';
require_once './Tools/autoloader.php';

// ===== GESTION ERREUR 404=====
/**
 * Méthode levée lors d'une erreur 404
 */
$app->notFound(function () use ($app) {
    ob_start();
    include "./404.php";
    ob_end_flush();
});

// ===== MIDDLEWARE : $authenticate =====
/**
 * Middleware de gestion d'authentification de l'utilisateur
 */
$authenticate = function () use ($app, $env) {

    // On récupère le gestionnaire de session et on lance la session si elle n'est pas active
    $mgtSession = GestionSession::getGestionnaireSession();
    $mgtSession->startSession();

    // On récupère l'utilisateur connecté et on le met dans la variable d'environnement de Slim
    $env['user'] = $mgtSession->getConnectedUser();

    // Si on a bien récupéré l'utilisateur connecté, on le met dans le header de réponse afin
    // que les pages côté client puisse le récupérer
    if (isset($env['user'])) {
        $app->response()->header("token", $env['user']->getId());
    }
};

// ===== ENTREE DE L'APPLICATION =====
$app->map('/', $authenticate, function () {
    include "./gesprj/index.html";
})->via('GET', 'POST');
```

Lors du lancement de l'application, on arrive sur la page *index.html*, qui sert de contrôleur frontal pour le framework Slim.

On peut voir la définition et l'utilisation d'un middleware. C'est une fonction qui va (quand elle est appelée sur une route) être exécutée **avant** le code lancé par l'appel de l'URL cible.

Comme l'application est paramétrée pour arriver à la racine quand on la lance, on va donc atteindre l'URL « / ». Le fait de mapper une URL via la fonction `->map` permet d'exécuter le code seulement sur les méthodes de requête définies avec le « via ». Ici, on va donc lancer la page HTML *index.html*.

LA PAGE INDEX.HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Authentification</title>
  <!-- Bootstrap -->
  <!--<link href="Bootstrap/css/bootstrap.css" rel="stylesheet">-->
  <link href="lib/bootstrap/bootstrap.min.css" rel="stylesheet">
  <link href="lib/datatables/jquery.dataTables.min.css" rel="stylesheet">
  <link href="lib/datatables/select.dataTables.min.css" rel="stylesheet">
  <link href="lib/datatables/buttons.dataTables.min.css" rel="stylesheet">
  <link href="./style.css" rel="stylesheet">
  <link rel="stylesheet prefetch" href='http://netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.min.css'>
</head>

<body ng-app="gesprj">

  <div ng-view></div>
  <!-- DEPENDANCES JQUERY$/DATATABLES -->
  <script type="text/javascript" src="./lib/datatables/jquery-1.11.3.js"></script>
  <script type="text/javascript" src="./lib/datatables/jquery.dataTables.min.js"></script>
  <script type="text/javascript" src="./lib/datatables/dataTables.select.min.js"></script>
  <script type="text/javascript" src="./lib/datatables/dataTables.buttons.min.js"></script>
  <script type="text/javascript" src="./lib/datatables/dataTables.bootstrap.min.js"></script>
  <!-- DEPENDANCES ANGULAR -->
  <script type="text/javascript" src="./lib/angular/angular.js"></script>
  <script type="text/javascript" src="./lib/angular/angular-route.min.js"></script>
  <script type="text/javascript" src="./lib/angular/angular-cookies.min.js"></script>
  <script type="text/javascript" src="./gesprj/app.js"></script>
  <!-- DEPENDANCES CONTROLLERS -->
  <script type="text/javascript" src="./gesprj/modules/main/mainController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/login/loginController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/header/headerController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/menu/menuController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/contenu/accueilController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/contenu/tableProjetsController.js"></script>
  <script type="text/javascript" src="./gesprj/modules/contenu/pageProjet/pageProjetController.js"></script>
```

L'image de la page *index.html* ici présentée est tronquée, la suite n'étant que d'autres appels de scripts de contrôleurs et la fermeture des balises HTML ouvertes.

Je dois préciser ici que l'application est single-page, c'est-à-dire qu'on ne va pas recharger une nouvelle page à chaque action de l'utilisateur afin de fluidifier son expérience. On va donc modifier cette page HTML dynamiquement grâce à Angular.

Cette page contient :

- le `<head>` de l'application, où sont chargés les pages de style,
- le `<body>` avec l'attribut `ng-app`, qui permet de dire à la page que le contrôleur global Angular de l'application est *gesprj*,
- la `<div ng-view>` qui est la partie de la page qui va être modifiée dynamiquement,
- les différents scripts à charger pour le bon fonctionnement de l'application :
 - o les scripts des bibliothèques

- les scripts des contrôleurs des pages html

Comme le <body> est lié au ng-app *gesprj*, le framework Angular va aller chercher dans le dossier « *gesprj* » (à la racine de l'application) le fichier *app.js*.

LE FICHIER APP.JS

```
// DÉFINITION DE L'APPLICATION ET DES MODULES UTILISÉS
// =====
var gesprj = angular.module('gesprj', [
    'ngRoute', 'ngCookies', 'ui.bootstrap', 'textAngular'
]);

// AU DÉMARRAGE DE L'APPLICATION
// =====
gesprj.run(['$rootScope', '$location', '$http', '$cookies',
    function ($rootScope, $location, $http, $cookies) {

        // Récupération du token qui est ici l'id de l'utilisateur connecté
        // -----
        $rootScope.token = $cookies.get('token') || "";

        // Gestion du changement de location
        // -----
        $rootScope.$on('$locationChangeStart', function(event, next, current) {

            // Si l'utilisateur est connecté et qu'il est sur la page login, on redirige vers la page home
            if ($location.path() == '/login' && $rootScope.token) {
                $location.path('/main');
            }
            // Si aucun utilisateur n'est connecté, on redigire vers la page de connexion
            if ($location.path() != '/login') {
                if (!$rootScope.token || $rootScope.token == "") {
                    $location.path('/login');
                }
            }
        });

        // On stocke dans le rootScope des informations utiles
        // -----
        // La page affichée à la base est "accueil"
        $rootScope.pageAffichee = "accueil";
        // On prépare des variables qui permettront de faciliter la
        // sélection des données pour les tableaux ou les formulaires
        $rootScope.utilisateurLieTableau = {id:null};
        $rootScope.projetSelectionne = {id:null};
        $rootScope.dossierSelectionne = {id:null};
        $rootScope.demandeSelectionnee = {id:null};
        $rootScope.developpementSelectionne = {id:null};
        $rootScope.demandeSelectionnee = {id:null};
        $rootScope.tacheSelectionnee = {id:null};

    }]);
```

Premièrement, on définit la variable de l'application pour Angular (ici *gesprj*). On lui donne les modules qu'elle devra utiliser dans l'application.

Ensuite, la fonction *.run* est lancée, où l'on gère, entre autres, l'instanciation des variables du scope global de l'application, qui s'appelle le *rootScope*.

Pour l'exemple de connexion utilisateur, ici, aucun utilisateur n'est connecté, on lance le chemin « */login* ».

```
// CONFIGURATION DES ROUTES D'ENTREE DANS L'APPLICATION VIA ANGULAR
// =====
gesprj.config(['$routeProvider',
function($routeProvider) {
$routeProvider.
when('/main', {
templateUrl: 'gesprj/modules/main/main.html',
controller: 'MainController'
}).
when('/login', {
templateUrl: 'gesprj/modules/login/login.html',
controller: 'LoginController'
}).
otherwise({redirectTo: '/main'});
}]);
```

C'est donc dans cette fonction de configuration de l'application que sont gérés les chemins appelés.

Ici, on a donc demandé « /login », on va donc envoyer la page *login.html* tout en définissant son contrôleur, *LoginController*. Pour rappel, la page *login.html* va s'afficher dans la div « ng-view » du fichier *index.html* puisque le <body> contient l'attribut *ng-app = "gesprj"*.

LA PAGE LOGIN.HTML

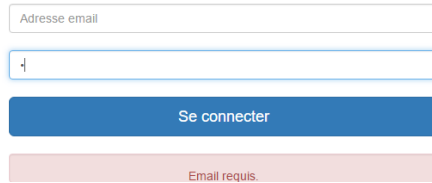
```
<div class="container">
<div class="jumbotron">
<h1 class="text-center">GESPRJ</h1>
<h2>Bienvenue sur l'outil de communication autour des projets d'ATLANTE DATA</h2>
</div>
<div style="..." class="center-block">
<form name="formLogin" data-ng-submit="login()" novalidate
class="form-signin">
<label for="idEmailLogin" class="sr-only">Email</label>
<input class="form-control" type="email" name="nameEmailLogin" id="idEmailLogin"
data-ng-model="connectObj.email"
required autofocus placeholder="Adresse email"/>
<br/>
<label for="idPasswordLogin" class="sr-only">Mot de passe</label>
<input class="form-control" type="password" name="namePasswordLogin" id="idPasswordLogin"
data-ng-model="connectObj.password" required placeholder="Mot de passe"/>
<br/>
<button class="btn btn-lg btn-primary btn-block" type="submit">Se connecter</button>
</form>
<br/>
<div ng-show="formLogin.$submitted || formLogin.nameEmailLogin.$touched">
<span class="alert alert-danger form-control fade in text-center"
ng-show="formLogin.nameEmailLogin.$error.required">Email requis.</span>
<span class="alert alert-danger form-control fade in text-center"
ng-show="formLogin.nameEmailLogin.$error.email">L'email n'est pas valide</span>
</div>
<div ng-show="formLogin.$submitted || formLogin.namePasswordLogin.$touched">
<span class="alert alert-danger form-control fade in text-center"
ng-show="formLogin.namePasswordLogin.$error.required">Mot de passe requis.</span>
</div>
</div>
</div>
```

Les choses intéressantes à noter dans cette page :

- L'utilisation de Bootstrap pour la gestion du css, grâce à l'application de classes css.
- L'attribut « ng-model » qui permet de lier la valeur contenue dans un champ à une variable du contrôleur lié (ici *LoginController*).
- L'attribut « ng-show » qui permet d'afficher le contenu seulement si la condition est respectée. Par exemple, si l'utilisateur a soumis le formulaire via le bouton « Se connecter » ou qu'il est sorti du champ email, et que ce champ email n'est pas renseigné, on affiche une erreur :

GESPRJ

Bienvenue sur l'outil de communication autour des projets d'ATLANTEDATA



- L'attribut « ng-submit » sur la balise <form> permet d'appeler la fonction *login()* lorsque le formulaire est soumis

LE CONTROLEUR LOGINCONTROLLER

```
// ===== LOGIN CONTROLLER =====
gesprj
.controller('LoginController', function ($scope, AuthenticateService) {

    // On crée un objet vide contenant l'objet utilisateur pour la connexion
    $scope.connectObj = {
        email: "",
        password: ""
    };

    // On crée la variable erreurLogin qui sera modifiée en fonction des erreurs
    $scope.erreurLogin = "";

    /**
     * Méthode de connexion lancée lorsque l'utilisateur clique sur le bouton "Se connecter"
     */
    $scope.login = function () {
        // Demande au service rest de faire une requête de connexion au serveur
        AuthenticateService.login($scope.connectObj);
    }

});
```

Dans l'exemple, on va lancer la fonction login de la classe *AuthenticateService* afin de vérifier si les données renseignées par l'utilisateur sont bonnes.

LA FONCTION LOGIN DE L'AUTENTICATESERVICE

```
/**
 * Méthode qui va lancer une requête au serveur pour vérifier les informations de connexion
 * Si OK : envoie vers la location /main
 * Si KO : envoie une alerte à l'utilisateur
 */
service.login = function (connectObj) {

    // On envoie une requête en POST au serveur en lui passant l'objet
    // utilisateur contenant les informations de connexion
    $http.post("/GES_PRJ/login", connectObj)
        .then
        // Une fois la requête effectuée, on vérifie si ça a été un succès ou non
        (function (responseLogin) {
            if (responseLogin.data.success == true) {

                if (responseLogin.data.utilisateurConnecte.token != null &&
                    responseLogin.data.utilisateurConnecte.token != "") {

                    // Si c'est bon, on stocke l'utilisateur connecté dans les cookies et dans le rootscope
                    service.manageToken(responseLogin.data.utilisateurConnecte.token);
                    $rootScope.utilisateurConnecte = responseLogin.data.utilisateurConnecte;

                    // On redirige vers /main
                    $location.path('/main');
                }
            }
            else {
                $rootScope.showAlert('danger', responseLogin.data.error.message, 5000);
            }
        })
    );
};
```

On va donc envoyer une requête au serveur pour vérifier les informations saisies. Pour rappel, c'est l'*index.php* qui va recevoir la requête.

LA ROUTE /LOGIN EN POST DANS L'INDEX.PHP

```
// ===== PAGE DE LOGIN =====
$app->post('/login', function () use ($app) {
    // on prépare le tableau que l'on va envoyer en réponse
    $out = array("success" => false, "message" => "");
    // on récupère les paramètres envoyés dans le corps de la requête effectuée
    $params = json_decode($app->request()->getBody());

    if (isset($params)) {
        // On appelle le controller php pour les utilisateurs
        try {
            $uti = UtilisateurController::getUtilisateurByEmailAndMotDePasse($params->email, $params->password);
            if (isset($uti)) {
                switch (get_class($uti)) {
                    case "ChefProjet": $typeUti = Utilisateur::TYPE_EST_CHEFDEPROJET;
                        break;
                    case "Developpeur": $typeUti = Utilisateur::TYPE_EST_DEVELOPPEUR;
                        break;
                    case "EmployeClient": $typeUti = Utilisateur::TYPE_EST_EMPLOYECLIENT;
                        break;
                    default: $typeUti = -1;
                        break;
                }
                // On met l'utilisateur connecté en session
                $mgtSession = GestionSession::getGestionnaireSession();
                $mgtSession->startSession();
                $mgtSession->setUser($uti);
                // on remplit le tableau de réponse
                $out["success"] = true;
                $out["message"] = "Connexion réussie";
                $out["utilisateurConnecte"] = array(
                    "token" => $uti->getId(),
                    "nom" => $uti->getNom(),
                    "prenom" => $uti->getPrenom(),
                    "typeUtilisateur" => $typeUti);
            } else {
                $error = new ApplicationError("Erreur de connexion.");
                $out = $error->getArrayError();
            }
        } catch (Exception $e) {
            $error = new ApplicationError($e->getMessage());
            $out = $error->getArrayError();
        }
    } else {
        // Permet d'envoyer une erreur si il n'y avait pas de paramètres envoyés
        // dans le corps de la requête, si les contrôles HTML n'ont pas suffi.
        $error = new ApplicationError("L'email et l'identifiant sont obligatoires.");
        $out = $error->getArrayError();
    }
    $app->response()->header("Content-Type", "application/json");
    $app->response()->setBody(json_encode($out));
});
```

Ici, on va donc appeler la méthode statique *getUtilisateurByEmailAndMotDePasse* de la classe Utilisateur Controller :

```
/**
 * @param $email
 * @param $motdepasse
 * @return ChefProjet|Developpeur|EmployeClient|null
 * @throws Exception
 */
static function getUtilisateurByEmailAndMotDePasse($email,$motdepasse) {
    return UtilisateurDAL::selectUtilisateurByEmailAndMotDePasse(new Utilisateur(null,$email,$motdepasse));
}
```

Rien de spécial à noter, à part que la méthode peut renvoyer tout type de classe-fille de la classe Utilisateur ou *null* si les données de connexion ne sont pas bonnes.

```

/**
 * Méthode appelant la base de données pour vérifier si les données de l'utilisateur sont OK
 * et crée un objet selon le type d'utilisateur
 * @param Utilisateur $utilisateur
 * @return ChefProjet|Developpeur|EmployeClient|null
 * @throws Exception
 */
static function selectUtilisateurByEmailAndMotDePasse(Utilisateur $utilisateur)
{
    $email = $utilisateur->getEmail();
    $password = $utilisateur->getPassword();
    $retour = null;
    try {
        $retour = MySqlFunctions::query("SELECT id,nom,prenom,TYPE_UTILISATEUR_id " .
                                         "FROM UTILISATEUR " .
                                         "WHERE email = '$email' AND password = '$password'");

        if (isset($retour[0])) {
            $utilisateur->setNom($retour[0]['nom']);
            $utilisateur->setPrenom($retour[0]['prenom']);
            $utilisateur->setId($retour[0]['id']);

            switch ($retour[0]['TYPE_UTILISATEUR_id']) {
                case Utilisateur::TYPE_EST_CHEFDEPROJET:
                    $utilisateur = new ChefProjet($utilisateur->getId(), $utilisateur->getEmail(),
                                                    $utilisateur->getPassword(), $utilisateur->getNom(), $utilisateur->getPrenom());
                    break;
                case Utilisateur::TYPE_EST_DEVELOPPEUR:
                    $utilisateur = new Developpeur($utilisateur->getId(), $utilisateur->getEmail(),
                                                    $utilisateur->getPassword(), $utilisateur->getNom(), $utilisateur->getPrenom());
                    break;
                case Utilisateur::TYPE_EST_EMPLOYECLIENT:
                    $utilisateur = new EmployeClient($utilisateur->getId(), $utilisateur->getEmail(),
                                                    $utilisateur->getPassword(), $utilisateur->getNom(), $utilisateur->getPrenom());
                    break;
            }
            $retour = $utilisateur;
        } else {
            $retour = null;
        }
    } catch (Exception $e) {
        throw new Exception($e->getMessage());
    }

    return $retour;
}

```

Si la requête a bien fourni un résultat, l'objet sera créé en fonction du type utilisateur, renvoyé au contrôleur *UtilisateurController*, qui renverra l'objet au contrôleur frontal, qui l'enverra en réponse à la vue.

Si tout est ok, la configuration des routes dans l'*app.js* nous renvoie vers le fichier *main.html* (son contrôleur ne servant qu'à aller voir le type d'utilisateur afin d'afficher des informations différentes en fonction du type, il n'est pas utile de le détailler ici).

LE FICHER MAIN.HTML

```
<ng-include src="'gesprj/modules/header/header.html'"></ng-include>

<div ng-controller="MainController" class="container-fluid">
  <div class="row" ng-if="utilisateurConnecte.typeUtilisateur">

    <ng-include src="'gesprj/modules/menu/menu.html'"></ng-include>

    <div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
      <central class="central">
        {{text}}
      </central>
    </div>
  </div>
</div>
```

C'est le « squelette » HTML de la page générale de l'application. On y inclut le header, le menu et une directive créée à la main, <central>.

LA DIRECTIVE « CENTRAL »

```
gesprj
.directive("central", function ($rootScope) {
  return {
    restrict: 'E',
    replace: true,
    template: '<ng-include src="templatePageAffichee"/>',
    link: function ($scope, element, attrs) {

      $scope.templateUrl = "";
      $scope.$watch(function () {
        return $rootScope.pageAffichee;
      }, function (pageAffichee) {

        if (pageAffichee) {

          switch (pageAffichee) {
            case "accueil":
              $scope.templatePageAffichee = '/GES_PRJ/gesprj/modules/contenu/accueil.html';
              $rootScope.headerTitle = "Accueil";
              break;

            case "projets":
              $scope.templatePageAffichee = '/GES_PRJ/gesprj/modules/contenu/tableProjets.html';
              $rootScope.headerTitle = "Liste des projets";
              break;

            case "pageProjet":
              $scope.templatePageAffichee = '/GES_PRJ/gesprj/modules/contenu/pageProjet/pageProjet.html';
              $rootScope.headerTitle = "Page d'un projet";
              break;
          }
        }
      });
    }
  };
});
```

Une directive Angular est la définition d'un comportement qui va être activé via le DOM. Ici, c'est à l'appel de la balise <central> dans n'importe quel page HTML. Celle-ci permet de surveiller la variable du rootScope *pageAffichee*, qui est modifiée à chaque fois que l'on sélectionne une option dans le menu de l'application. Quand c'est le cas, on applique le code qui suit. En l'occurrence, on va aller, entre autres, modifier le contenu de la balise <central> avec la page HTML définie dans chaque cas.

Par défaut, on arrivera sur la page d'accueil :

GESPRJ
Accueil - Chef de projet - Chef de projet 2 Prénom 2

Mon compte
Mes alertes
Recherche
Deconnexion

Menu principal

Accueil
Projets
Créer un projet
Demandes
Validation de développements
Historique

Administration

Plannings
Clients
Développeurs
Créer un compte utilisateur

Liste des projets
Liste des demandes clients
Liste des développements à valider

Accéder au projet sélectionné

Rechercher :

Libelle	Clients	Chef de projet	Priorité	Date lancement	Date cloture souhaitée
TEST PROJET 1	Employé client 3, Employé client 4	Chef de projet 2	1	2015-12-21	2015-12-10
TEST PROJET 2	Employé client 3	Chef de projet 2	5	2015-12-21	2015-12-09
TEST PROJET 3	Employé client 4	Chef de projet 2	2	2015-12-21	2016-01-02
TEST PROJET 4	Employé client 3, Employé client 4	Chef de projet 2	3	2015-12-21	2016-01-02
TEST PROJET 5	Employé client 3	Chef de projet 2	3	2015-12-21	2016-01-02

Accéder au projet sélectionné

Il y a actuellement un problème avec les icônes de tri des colonnes, qui ne s'affiche pas dans la version de la bibliothèque Datatables, mais on peut quand même cliquer à côté du nom de la colonne pour trier.

Les informations de la page dépendent de l'utilisateur connecté (le menu, le header, les onglets ...), grâce à la balise `<ng-if>`.

EXEMPLE DE REALISATION N°2 : LA GESTION DE PROJET POUR UN CHEF DE PROJET

Je vais maintenant détailler la partie « gestion des projets » pour un chef de projet, en parlant du système d'onglet, des tableaux et des formulaires.

LE SYSTEME D'ONGLET

Pour les onglets, j'ai utilisé la bibliothèque « UI Bootstrap » créée pour Angular.

```
<div ng-controller="AccueilController">

  <!-- SI L'UTILISATEUR EST UN CHEF DE PROJET -->
  <div ng-if="isChefProjet">

    <uib-tabset>

      <!-- ONGLET : LISTE DES PROJETS -->
      <uib-tab heading="Liste des projets">
        <ng-include src="'./gesprj/modules/tableaux/tableauProjets.html'"></ng-include>
      </uib-tab>

      <!-- ONGLET : LISTE DES PROJETS -->
      <uib-tab heading="Liste des demandes clients">
        <ng-include src="'./gesprj/modules/tableaux/tableauDemandesClients.html'"></ng-include>
      </uib-tab>

      <!-- ONGLET : LISTE DES DEVELOPPEMENTS A VALIDER -->
      <uib-tab heading="Liste des developpements à valider">
        <ng-include src="'./gesprj/modules/tableaux/tableauDeveloppementsAValider.html'"></ng-include>
      </uib-tab>

    </uib-tabset>
  </div>

  <!-- SI L'UTILISATEUR EST UN DEVELOPPEUR -->
  <div ng-if="isDeveloppeur">
```

Chaque onglet est donc lié à une page HTML. Dans le cas qui nous intéresse ici, je vais parler du tableau de projets affichée à l'accueil pour un chef de projet.

LE TABLEAU DES PROJETS D'UN CHEF DE PROJET

Les tableaux sont gérés avec la bibliothèque « Datatables » qui est une bibliothèque JQuery.

```
<div ng-controller="TableauProjetsController">

  <!-- SI L'UTILISATEUR EST UN CHEF DE PROJET -->
  <div ng-if="isChefProjet">
    <div class="boutonsTableaux">
      <button type="button" class="btn btn-primary" ng-click="accesProjet()" ng-disabled="!isARowSelected">
        Acceder au projet sélectionné
      </button>
    </div>

    <table id="tableauProjets" class="table table-striped table-bordered" cellspacing="0" width="100%">
      <thead>
        <tr>
          <th>projet_id</th>
          <th>chefProjet_id</th>
          <th>Libelle</th>
          <th>Clients</th>
          <th>Chef de projet</th>
          <th>Priorité</th>
          <th>Date lancement</th>
          <th>Date cloture souhaitée</th>
        </tr>
      </thead>
    </table>

    <div class="boutonsTableaux">
      <button type="button" class="btn btn-primary" ng-click="accesProjet()" ng-disabled="!isARowSelected">
        Acceder au projet sélectionné
      </button>
    </div>
  </div>

  <!-- SI L'UTILISATEUR EST UN DEVELOPPEUR -->
```

Ici, on peut remarquer quelques détails :

- L'utilisation de *ng-disabled*, qui rend indisponible à l'utilisateur la balise associée selon une condition, ici le fait qu'une ligne du tableau soit sélectionnée ou non.
- On définit uniquement les entêtes de colonnes pour le tableau, sans omettre de lui mettre un id (qui servira à le retrouver dans le contrôleur)

```
gesprj
.controller('TableauProjetsController', function ($scope, $http, $location, $rootScope) {

    $scope.isARowSelected = false;

    // on récupère l'id de l'utilisateur lié au tableau (mis à jour ici par accueilController.js)
    $http.post("/GES_PRJ/recuperationUtilisateur/" + $rootScope.utilisateurLieTableau.id).then
    (function (responseRecuperationUtilisateur) {
        if (responseRecuperationUtilisateur.data.success == true) {
            // On définit un tableau avec les méthodes de la bibliothèque Datatables
            var tableauProjets = $('#tableauProjets').DataTable({
                // On récupère via une requête ajax les données à afficher
                "ajax": {
                    "url": "/GES_PRJ/recuperationProjets/"
                    + responseRecuperationUtilisateur.data.utilisateur.typeUti
                    + "/" + responseRecuperationUtilisateur.data.utilisateur.id,
                    "type": "POST"
                },
                // On définit toutes les colonnes
                "columns": [{ "data": "id" }, { "data": "chefProjet.id" }, { "data": "libelle" },
                    { "data": "libelleClientsNoms" }, { "data": "chefProjet.nom" }, { "data": "priorite" },
                    { "data": "dateLancement" }, { "data": "dateClotureSouhaitee" }
                ],
                // On définit les colonnes que l'on veut mettre invisible
                "columnDefs": [{ "targets": [0, 1], "visible": false }
                ],
                // On autorise pas la multi-sélection
                select: { style: "single" },
                // On ne met pas de système de pagination
                "paging": false, "info": false,
                // On modifie les textes affichés
                language: { search: "Rechercher : " }
            });

            // Quand on désélectionne une ligne du tableau
            tableauProjets.on('deselect', function (e, dt, type, indexes) {
                $scope.isARowSelected = false;
                $scope.$apply();
            });

            // Quand on sélectionne une ligne du tableau
            tableauProjets.on('select', function (e, dt, type, indexes) {
                $scope.idSelectedProjet = tableauProjets.cell('.selected', 0).data();
                $scope.isARowSelected = true;
                $scope.$apply();
            });
        }
    });
});
```

Comme ce tableau va afficher tous les projets pour l'utilisateur connecté, on envoie au serveur une requête AJAX de récupération des projets liés à l'utilisateur en fonction de son type et de son id, traitée comme d'habitude par *l'index.php* :

```
// POUR TOUS LES PROJETS D'UN UTILISATEUR
// -----
$app->post('/:typeUti/:idUti', function ($typeUti, $idUti) use ($app, $env) {
    $out = null;

    if (isset($env['user'])) {
        try {
            $listeProjets = null;

            // On récupère la liste de projets en fonction du type d'utilisateur
            switch ($typeUti) {
                case Utilisateur::TYPE_EST_CHEFDEPROJET:
                    $listeProjets = ProjetController::getByChefProjet(new ChefProjet($idUti), true, true);
                    break;
                case Utilisateur::TYPE_EST_DEVELOPPEUR:
                    $listeProjets = ProjetController::getByDeveloppeur(new Developpeur($idUti), true, true);
                    break;
                case Utilisateur::TYPE_EST_EMPLOYECLIENT:
                    $listeProjets = ProjetController::getByEmployeClient(new EmployeClient($idUti), true, true);
                    break;
            }

            // On remplit un tableau formaté comme voulu (en tableau et non en objet, pour pouvoir
            // le récupérer côté client via la réponse à la requête
            $tabProjets = array();
            foreach ($listeProjets as $projet) {
                array_push($tabProjets, Projet::projetToArray($projet));
            }

            // On positionne les variables attendues par la bibliothèque Datatables
            $out["data"] = $tabProjets;
            $out["draw"] = 1;
            $out["recordsTotal"] = count($tabProjets);
            $out["recordsFiltered"] = count($tabProjets);
            $out["success"] = true;

        } catch (Exception $e) {
            $out["success"] = false;
            $out["message"] = $e->getMessage();
        }
    } else {
        $app->redirect("./SessionExpiree.php");
    }

    $app->response()->header("Content-Type", "application/json");
    $app->response()->setBody(json_encode($out));
});
```

On appelle donc la méthode statique *getByChefProjet* du contrôleur *ProjetController* :

```

/**
 * Récupère une liste de projets via l'id du chef de projet
 * @param $utilisateur
 * @param bool|false $recupChefProjet
 * @param bool $recupClients
 * @return array|null
 * @throws Exception
 */
static function getByChefProjet($utilisateur, $recupChefProjet = false, $recupClients = false)
{
    $listeProjets = ProjetDAL::selectByChefProjet($utilisateur);

    if ($recupChefProjet) {
        /** @var Projet $projet */
        foreach ($listeProjets as $projet) {
            $projet->setChefProjet(UtilisateurController::getById($projet->getChefProjet()));
        }
    }

    if ($recupClients) {
        /** @var Projet $projet */
        foreach ($listeProjets as $projet) {
            $projet->setListeClients(UtilisateurController::getClientsByIdProjet($projet));
        }
    }

    return $listeProjets;
}

```

Celle-ci va appeler la DAL pour récupérer les données, et si voulu, va renseigner toutes les informations du chef de projet ou des clients liés à ce projet.

```

* recupere en base tous les projets d'un chef de projet
* @param Utilisateur $utilisateur
* @return array|null
* @throws Exception
*/
static function selectByChefProjet(Utilisateur $utilisateur)
{
    $idUtilisateur = $utilisateur->getId();
    $retourListeProjets = array();
    try {
        // Je récupère tous les projets dont le chef de projet est l'id utilisateur
        $query = MySqlFunctions::query("SELECT id,CHEF_PROJET_id,date_lancement " .
            "FROM PROJET WHERE CHEF_PROJET_id = '$idUtilisateur'");
        if (isset($query[0])) {
            foreach ($query as $key => $value) {
                // je construit l'objet projet
                $projet = new Projet();
                $idEntite = $value['id'];
                $projet->setId($idEntite);
                $projet->setDateLancement($value['date_lancement']);
                // je vais mettre un objet ChefProjet avec seulement son id
                $chefProjet = new ChefProjet();
                $chefProjet->setId($value['CHEF_PROJET_id']);
                $projet->setChefProjet($chefProjet);
                // je vais voir l'entité generale avec le même id
                $queryEntiteGenerale = MySqlFunctions::query("SELECT libelle,description,dateCreation FROM ENTITE_GENERALE WHERE id = '$idEntite'");
                if (isset($queryEntiteGenerale[0])) {
                    $projet->setLibelle($queryEntiteGenerale[0]['libelle']);
                    $projet->setDescription($queryEntiteGenerale[0]['description']);
                    $projet->setDateCreation($queryEntiteGenerale[0]['dateCreation']);
                }
                // je vais voir l'entité validee avec le même id
                $queryEntiteValidee = MySqlFunctions::query("SELECT entite_mere_id,date_cloture_souhaitee,
                    date_cloture_effective,date_validation,
                    date_abandon,priorite FROM ENTITE_VALIDEE WHERE id = '$idEntite'");
                if (isset($queryEntiteValidee[0])) {
                    $projet->setDateClotureSouhaitee($queryEntiteValidee[0]['date_cloture_souhaitee']);
                    $projet->setDateClotureEffective($queryEntiteValidee[0]['date_cloture_effective']);
                    $projet->setDateValidation($queryEntiteValidee[0]['date_validation']);
                    $projet->setDateAbandon($queryEntiteValidee[0]['date_abandon']);
                    $projet->setPriorite($queryEntiteValidee[0]['priorite']);
                }
                array_push($retourListeProjets, $projet);
            }
        } else {
            $retourListeProjets = null;
        }
    } catch (Exception $e) {
        throw new Exception($e->getMessage());
    }
    return $retourListeProjets;
}

```

On remplit donc l'objet projet avec toutes les informations de la base, en prenant d'abord celles de la table PROJET, puis ENTITE_GENERALE, puis ENTITE_VALIDEE.

Une fois récupérées, je les mets dans un tableau de projets que je retourne au contrôleur, puis à l'*index.php*, qui les renvoie à la vue.

Comme indiqué précédemment, on peut sélectionner une ligne et cliquer sur un bouton « Afficher le projet sélectionné », ce qui amène au formulaire du projet. Comme on est chef de projet, on accède directement à la page en mode modification.

LE FORMULAIRE PROJET

(voir page suivante)

[Menu principal](#)

[Accueil](#)

Projets

[Créer un projet](#)

Demands

Validation de développements

Historique

Administration

Plannings

Clients

Développeurs

[Créer un compte utilisateur](#)

Informations principales

GED

Documentation

Cloturer le projet

Abandonner le projet

Informations générales

Libelle

TEST PROJET 1

Priorite

1

Date création

09/12/2015

Date lancement

21/12/2015

Date cloture souhaitee

10/12/2015

Valider les modifications

Développeurs

Clients

Nom	Prenom	
Employé client 3	Prenom 3	<button>Supprimer</button>
Employé client 4	Prenom 4	<button>Supprimer</button>

Description

H1 H2 H3 H4 H5 H6 P pre **B** *I* U ~~S~~

Je suis un projet 1

Valider la description

Le design n'est pas finalisé, je n'ai pas insisté sur ce point pour le moment.

Comme pour la page d'accueil du chef de projet, on a un système d'onglet, ce qui me permet donc de fractionner le code HTML en plusieurs fichiers pour faciliter le développement et la gestion des contrôleurs Angular.

Les modules GED et Documentation n'étant pas encore réalisés entièrement, je ne parlerais ici que de l'onglet du formulaire.

La page HTML étant conséquente, je vais la diviser en quelques exemples de code :

```
<!-- BOUTONS DE GESTION DU PROJET : CLOTURE/ABANDON -->
<div class="btn-group-vertical pull-right">
  <input type="button" class="btn btn-primary" ng-click="cloturerProjet()" value="Cloturer le projet"
  ng-show="(!projet.estCloture || !projet.estAbandonne) && utilisateurConnecte.isChefProjet"/>
  <input type="button" class="btn btn-primary" ng-click="abandonnerProjet()" value="Abandonner le projet"
  ng-show="(!projet.estCloture || !projet.estAbandonne) && utilisateurConnecte.isChefProjet"/>
</div>

<h2>Informations générales</h2>

<!-- FORMULAIRE -->
<form class="form-horizontal" role="form" data-ng-submit="validerModificationProjet()" novalidate>

  <!-- LIBELLE DU PROJET -->
  <div class="form-horizontal">
    <div class="form-group">
      <label for="libelle">Libelle</label>
      <input type="text" class="form-control" id="libelle"
      data-ng-model="projet.libelle" ng-disabled="projet.estCloture || projet.estAbandonne || !utilisateurConnecte.isChefProjet">
    </div>
  </div>
```

Là, on peut remarquer qu'on affiche/rend indisponible des composants selon si l'utilisateur connecté est un chef de projet ou si le projet est clôturé/abandonné.

Un autre morceau de code intéressant est le tableau des clients :

```
<table id="tableauClientsProjet" class="table table-striped table-bordered" cellspacing="0" width="100%">
  <tr>
    <th>Nom</th>
    <th>Prenom</th>
    <th></th>
  </tr>
  <tr ng-repeat="client in projet.listeClients">
    <td>{{ client.nom }}</td>
    <td>{{ client.prenom }}</td>
    <td>
      <button class="btn btn-danger" ng-click="supprimerClientProjet(client.id)"
      ng-show="!projet.estCloture && !projet.estAbandonne && utilisateurConnecte.isChefProjet">
        Supprimer
      </button>
    </td>
  </tr>
</table>
<div ng-hide="projet.listeClients.length"><em>Aucun client pour ce projet</em></div>
```

La directive *ng-repeat* permet de boucler sur les itérations d'un objet du scope, ce qui me permet ici de n'afficher que les informations souhaitées, c'est-à-dire le nom et le prénom des clients de la liste de clients du projet.

Autre directive intéressante, le *ng-hide*, qui permet de cacher le contenu de la balise pour une condition donnée, ici si la liste de clients n'est pas vide.

Comme la page HTML du formulaire projet est incluse dans la page projet globale, c'est donc le *pageProjetController* qui fait office de contrôleur :

```

gesprj
.controller('PageProjetController', function ($scope, $http, $location, $rootScope) {

    // On positionne la variable du contenu de l'editeur HTML
    $scope.htmlcontent = "";

    // On envoie une requête en POST pour récupérer toutes les informations du projet
    // dont l'id est dans la variable du rootScope projetSelectionne
    $http.post("/GES_PRJ/recuperationProjets/" + $rootScope.projetSelectionne.id).then
    (function (responseRecuperationProjet) {
        if (responseRecuperationProjet.data.success == true) {
            $scope.projet = responseRecuperationProjet.data.projet;

            // on construit des objet Date() en javascript pour pouvoir les afficher facilement
            $scope.projet.dateCreation = new Date($scope.projet.dateCreation);
            $scope.projet.dateLancement = new Date($scope.projet.dateLancement);
            $scope.projet.dateClotureSouhaitee = new Date($scope.projet.dateClotureSouhaitee);
            // On va positionner la variable du scope estCloture si on a une date de cloture effective
            if ($scope.projet.dateClotureEffective != null) {
                $scope.projet.dateClotureEffective = new Date($scope.projet.dateClotureEffective);
                $scope.projet.estCloture = true;
            } else {
                $scope.projet.estCloture = false;
            }
        }
    })
};

```

On envoie donc une requête en POST pour aller récupérer toutes les informations d'un projet :

```

// ===== RECUPERATION DES PROJETS =====
$app->group('/recuperationProjets', $authenticate, function () use ($app, $env) {

    // POUR UN PROJET PRECIS
    // -----
    $app->post('/:idProjet', function ($idProjet) use ($app, $env) {
        $out = null;

        if (isset($env['user'])) {
            try {
                $projet = new Projet();
                $projet->setId($idProjet);

                // On va récupérer l'objet projet via le contrôleur
                $projet = ProjetController::getById($projet, true, true);

                // Si on a bien récupéré l'objet
                if (is_object($projet)) {
                    // On transforme l'objet projet en tableau
                    $out["projet"] = Projet::ProjetToArray($projet);
                    $out["success"] = true;
                }
            } catch (Exception $e) {
                $out["success"] = false;
                $out["message"] = $e->getMessage();
            }
        } else {
            $app->redirect("./SessionExpiree.php");
        }

        $app->response()->header("Content-Type", "application/json");
        $app->response()->setBody(json_encode($out));
    });
});

```

Le contrôleur va appeler la méthode *selectById* de la DAL :

Une fois qu'on a effectué des modifications et qu'on a cliqué sur le bouton « Valider les modifications », cela appelle la fonction *ValiderModificationProjet* :

```

/**
 * Méthode appelé sur le bouton de validation de modification du projet
 */
$scope.validerModificationProjet = function () {
    $http.post("/GES_PRJ/modificationProjet", $scope.projet).then(function (response) {
        if (response.data.success) {
            // TODO : Mettre une div affichant la réussite de la modification
            alert("ok validerModificationProjet");
        }
        else {
            // TODO : Mettre une div affichant l'echec de la modification
            alert("erreur modif projet");
        }
    })
};

```

Celle-ci va envoyer une requête en POST au serveur, traitée par l'*index.php* :

```

// ===== MODIFICATION D'UN PROJET =====
$app->post('/modificationProjet', $authenticate, function () use ($app, $env) {

    // On récupère les paramètres envoyés
    $params = $app->request()->getBody();
    $request = json_decode($params);

    // On crée un objet Projet qu'on va remplir avec les informations reçues
    /** @var Projet $projet */
    $projet = new Projet();
    $projet->setId($request->id);
    $projet->setLibelle($request->libelle);
    $projet->setPriorite($request->priorite);
    $projet->setDescription($request->description);
    $projet->setDateClotureSouhaitee($request->dateClotureSouhaitee);
    $projet->setDateClotureEffective($request->dateClotureEffective);
    $projet->setDateLancement($request->dateLancement);

    // On appelle le contrôleur de projet, qui renvoie un boolean selon la réussite de la mise à jour
    $okModif = ProjetController::modificationProjet($projet);

    $out = null;
    if ($okModif) {
        $out["success"] = true;
    } else {
        $out["success"] = false;
    }

    $app->response()->header("Content-Type", "application/json");
    $app->response()->setBody(json_encode($out));
});

```

On appelle le *ProjetController*, qui va juste appeler la méthode *updateProjet* de la DAL :

```

/**
 * Mise à jour en base d'un projet
 * @param Projet $projet
 * @return bool
 * @throws Exception
 */
public static function updateProjet(Projet $projet)
{
    $retour = null;
    $idProjet = $projet->getId();
    // TODO : Mettre un rollback sur la transaction

    try {
        // je met à jour l'ENTITE GENERALE
        $libelle = $projet->getLibelle();
        $description = $projet->getDescription();

        $okModif = MySQLFunctions::query("UPDATE entite_generale SET " .
            " libelle='{$libelle}'" .
            " description='{$description}'" .
            " WHERE id={$idProjet}");

        if ($okModif) {
            // je met à jour l'ENTITE VALIDEE
            // en convertissant les dates dans le format attendu par la base de données MySQL
            $dateClotureSouhaitee = $projet->getDateClotureSouhaitee() != null ? "" . time_phpToSql(strtotime($projet->getDateClotureSouhaitee())) .
            $dateClotureEffective = $projet->getDateClotureEffective() != null ? "" . time_phpToSql(strtotime($projet->getDateClotureEffective())) .
            $dateValidation = $projet->getDateValidation() != null ? "" . time_phpToSql(strtotime($projet->getDateValidation())) . "" : "NULL";
            $dateAbandon = $projet->getDateAbandon() != null ? "" . time_phpToSql(strtotime($projet->getDateAbandon())) . "" : "NULL";

            $priorite = $projet->getPriorite();

            $okModif = MySQLFunctions::query("UPDATE entite_validee SET " .
                " date_cloture_souhaitee={$dateClotureSouhaitee}" .
                " date_cloture_effective={$dateClotureEffective}" .
                " date_validation={$dateValidation}" .
                " date_abandon={$dateAbandon}" .
                " priorite='{$priorite}'" .
                " WHERE id={$idProjet}");
        }
    } catch (Exception $e) {
        throw new Exception($e->getMessage());
    }

    return $okModif;
}

```

Pour rappel, selon les souhaits utilisateurs, chaque partie de la page projet doit être validée séparément (par exemple, si on valide les modifications sur la partie « Informations générales », la partie « Description » ne sera pas validée). En adéquation avec ce principe, la validation de modification n'entraîne pas un retour immédiat à la liste des projets mais on reste sur la page du projet.

La clôture ou l'abandon utilisant la même méthode *updateProjet*, il n'est pas utile de les mettre ici.

CONCLUSION

SUR LE RESPECT DU CAHIER DES CHARGES

Dans l'ensemble et compte tenu de l'avancée sur le projet, je considère avoir dans l'ensemble collé au cahier des charges que je m'étais fixé au début. Les fonctionnalités **sont ou seront là** à la fin du projet, ou au moins seront faciles à mettre en place.

Par contre, il a été plus **difficile** que prévu pour moi de **coller aux maquettes** sélectionnées par les utilisateurs. Utilisant par manque de temps Bootstrap, et sans formation de design web, je pense que le logiciel final sera forcément un peu différent graphiquement. Il faut donc que **je minimise les différences** pour ne pas que les utilisateurs pensent que leurs idées de design n'ont pas été respectées et qu'ils se fidélisent moins vite à l'application.

A signaler que l'idée d'avoir un clic droit sur les tableaux n'était pas bonne pour deux principales raisons :

- L'utilisateur n'est pas habitué à faire des clics droits sur une page internet
- Si un jour il est question de mobilité, il n'est pas possible de clic droit sur des appareils mobiles.

A la place, il a donc été préférable de passer par de simples boutons.

Sinon, les premiers retours utilisateurs ont été bons pour **l'ergonomie et la simplicité**, je suis donc sur la bonne voie pour les satisfaire à ce niveau-là (même si par rapport à l'existant, la tâche n'est pas très ardue).

SUR LA GESTION DE PROJET

Comme indiqué précédemment, je travaillais sur plusieurs autres dossiers en même temps, donc j'ai pu me dégager du temps pour accélérer ce projet quand j'avais besoin, et au contraire, le mettre un peu de côté quand j'avais de l'avance. J'ai, au final, respecté le planning prévisionnel, ayant dû stopper le projet après une **dizaine de jours de codage**.

J'aurais aimé pouvoir me consacrer entièrement à ce projet afin de définir un planning complet et de savoir précisément les points où j'ai pris du retard/de l'avance, de comprendre pourquoi afin **d'affiner mes plannings** vis-à-vis de mes compétences réelles pour la suite.

De plus, j'aurais souhaité avoir plus de temps pour finir le projet en entier, j'ai beaucoup apprécié le fait de m'occuper d'un projet de A à Z.

Dans l'ensemble, j'ai perdu du temps sur :

- **Le maquetage** : j'ai sous-estimé le nombre d'itérations de maquettes nécessaires afin de satisfaire les futurs utilisateurs.
- **L'apprentissage des technologies** : Entre le travail en entreprise (que je fais en Basic pur) et les cours à l'ENI, il m'a été plus difficile que prévu d'appréhender le php (que j'avais vu uniquement 2 semaines en cours à l'ENI) et Angular JS (sachant que j'avais une maîtrise moyenne de javascript).

Et j'ai gagné du temps sur :

- La réalisation du **diagramme des classes** : malgré la complexité relative, je m'en suis sorti plus rapidement que je ne l'aurais pensé.
- **La création de la base de données** : Avant de commencer le projet, je ne connaissais pas l'outil MySQL Workbench, qui m'a grandement aidé pour schématiser la base.

SUR L'AVENIR DU PROJET

Les points essentiels pour passer le prochain « palier » d'acceptation de l'application sont :

- La mise en place de la **GED** et de la documentation
- Le système **d'alertes**
- Le **mailing**

Après ces points-là, **l'essentiel de l'application aura été développé** à part des fonctionnalités mineures.

Il ne restera donc plus que les **tests** et la **formation** du personnel d'ATLANTEDATA et des futurs clients utilisateurs.

ANNEXE

EXEMPLE DE COMPTE-RENDU DES REUNIONS D'ENQUETES

REUNION DU 04/08/2015 + REUNION DU 07/08/2015 : RECUEIL DES BESOINS UTILISATEURS

SUJET

Recueillir les besoins fonctionnels des développeurs d'ATLANTEDATA au sujet d'une application de suivi de projet.

PARTICIPANTS

- Vincent HERAUD
- Pascal RAUX
- Didier HEURTAUX
- Cédric FAUCHER
- Véronique BOUYER

COMPTE-RENDU

INTRODUCTION : CHOIX DES TERMES/ENTITES

En préambule, j'ai demandé à ce qu'on se mette d'accord sur les termes et les entités de l'application.

- Un PROJET est l'entité la plus globale.
- Un DOSSIER est une sous-entité de PROJET.
- Une REALISATION est un nouveau développement
- Un SAV est une correction/modification mineure
- Une DEMANDE est une demande faite par un client

PARTIE 1 : L'EXISTANT

Ce qui ressort au niveau de l'existant, c'est que le wiki a un gros manque de clarté et d'organisation de ces pages. Il est pour eux très difficile de faire un suivi de projet et de communiquer avec un client par ce biais, car il n'y a aucun lien « fort » entre les différents contenus d'un projet.

Le wiki n'est pas non plus simple au niveau de la création des pages et du contenu. En effet, il fonctionne avec un éditeur utilisant des balises visibles à l'écran lors de la modification /création de texte, ce qui est fastidieux pour écrire un contenu clair et ordonné.

Une partie d'entre eux (les plus anciens) reste attachée à l'utilisation combinée du téléphone et de l'email. Ils ont donc besoin de ne pas trop perdre cette rapidité et facilité de communication.

PARTIE 2 : LES BESOINS FONCTIONNELS

LES MODULES

Gestion de mailing : La première chose qui est ressortie est une gestion de mailing dans l'application. Par exemple, quand un développeur indiquera une évolution sur le suivi d'un projet, le client associé au dossier en sera informé.

Gestion électronique de documents : Chaque projet devra avoir sa propre Gestion Electronique de Documents (ou GED).

Gestion de documentation : Chaque projet devra avoir une partie documentation.

Outil de recherche : Il devra y avoir un outil de recherche assez complet.

Page principale d'un projet : Chaque entité doit avoir une page résumant rapidement ce qu'elle est. Par exemple, on doit rapidement voir les dossiers d'un projet.

Suivi de l'avancée : Un outil doit permettre de rendre compte facilement de l'avancée des entités.

Planning : Un outil simple de planning est souhaité (très global).

AUTRES

- L'outil de création de contenu doit être le plus simple et rapide possible.
- Les clients doivent pouvoir proposer des demandes.
- Ces demandes devront être acceptées par un « Chef de projet » qui les transformera en une entité choisie par ses soins
- Pour eux, il existe 3 rôles :
 - Client
 - Développeur
 - Chef de projet
- Que les entités aient une notion de priorité.