

DÉDICACE

À

mes parents

REMERCIEMENTS

TABLE DES MATIÈRES

Dédicace	i
Remerciements	ii
Liste des sigles et abréviations	v
Résumé	vi
Abstract	vii
Liste des tableaux	vii
Liste des figures	viii
1 CONTEXTE ET PROBLÉMATIQUE	3
2 GÉNÉRALITÉS	4
2.1 Rapid Application Development	4
2.2 L'outil Ruby On Rails	4
2.3 Le Langage Ruby	6
2.4 Architecture et Création d'un projet avec Ruby On Rails	6
2.4.1 Création des classes de modèle	7
2.5 Scaffolding	8
2.5.1 Running	8
2.6 Avantages de Rails	8
2.7 L'outil Git	9
2.7.1 Git	9
2.8 Le pattern MVC	9
2.8.1 Définition	9
2.8.2 Présentation du MVC	10
2.8.3 Principe du MVC	10
2.9 Constructeur d'un projet UML	10
2.10 Le Développement Agile	12
2.10.1 Les principes du Manifeste Agile	12
2.10.2 La modélisation agile (AM)	12

3	ANALYSE ET CONCEPTION	13
3.1	Cahier de Charges	13
3.1.1	Fonctionnalités attendues de l'application	13
3.1.2	Besoins de l'application	13
3.1.3	Besoins fonctionnels	14
3.1.4	Besoins non fonctionnels	20
3.1.5	Etude de faisabilité	21
4	IMPLÉMENTATION	22
4.1	Introduction	22
4.2	Langages de programmation et Environnement de développement	22
4.2.1	Langages de programmation et Framework	22
4.2.2	Outils de développement	23
4.3	Présentation du projet	24
4.4	Modèle Architectural	25
4.4.1	Synoptique de fonctionnement d'une application web	25
4.4.2	Synoptique de fonctionnement d'une application web en Ruby On Rails	25
4.5	Arborescence de l'application	26
5	RÉSULTATS ET COMMENTAIRES	27
	Conclusion	28

LISTE DES SIGLES ET ABRÉVIATIONS

RÉSUMÉ

Mots-clés :

ABSTRACT

Keywords :

LISTE DES TABLEAUX

2.1	Les ancêtres de Ruby.	6
3.1	Enregistrement d'un Utilisateur	14
3.2	Modifier un utilisateur	14
3.3	Suppression d'un Utilisateur	15
3.4	Enregistrer un compte utilisateur	15
3.5	Enregistrer un compte utilisateur	15
3.6	Rechercher un utilisateur	16
3.7	Enregistrer un courrier	16
3.8	Modifier un courrier	16
3.9	Supprimer un courrier	17
3.10	Enregistrer un projet	17
3.11	Modifier un projet	17
3.12	supprimer un projet	18
3.13	Enregistrer une phase du projet	18
3.14	Modifier un projet	18
3.15	supprimer une phase du projet	19
3.16	Enregistrer une archive	19
3.17	Modifier une archive	19
3.18	supprimer une archive	20
3.19	Importer les données	20
3.20	Exporter les données	20

TABLE DES FIGURES

2.1	Terminal	5
2.2	Arbre généalogique de divers langages de programmation, incluant Ruby	6
2.3	Contenu d'un dossier de projet Rails	7
2.4	Première page d'un projet sur rails	8
2.5	Cycle de vie des états d'un fichier dans Git.	9
2.6	Schématisation du MVC	10
4.1	Arborescence du projet dans l'IDE RubyMine	24
4.2	Arborescence du projet dans l'explorateur windows	25
4.3	Synoptique de l'application web	25
4.4	Synoptique d'une application web en Ruby On Rails	25

INTRODUCTION GÉNÉRALE

Il se dégage aujourd'hui un consensus quant aux possibilités ouvertes par les technologies de l'information et de la communication (TIC) qui se développent rapidement dans tous les domaines de l'entreprise et plus largement de la société. Elles permettent de manipuler de l'information pour la stocker, la convertir, la gérer, la transmettre et la retrouver.

L'Internet est considéré comme étant un moyen idéal de communication, d'échange de données ou encore d'apprentissage, et est également un outil efficace pour avoir des informations sur un service ou un produit. L'apprentissage est l'un des services privilégiés qu'Internet offre aux visiteurs. Dans cette optique, de nombreuses applications et sites Web dynamiques ont vu le jour.

Les institutions publiques gouvernementales sont parmi les établissements qui ont besoin d'un système informatique pour bien conduire leur travail en évitant la perte de temps. Ces institutions dépendent de plus en plus de l'informatique pour réaliser leurs objectifs, elles sont donc plus sensibles à la qualité des services informatiques fournis aux différentes catégories d'utilisateurs et sont à la recherche de moyens et des ressources pour améliorer leurs services. Ainsi, restaurer le plus rapidement possible le fonctionnement normal des services afin de minimiser l'impact négatif de celui-ci sur les activités métiers, et s'assurer que les meilleurs niveaux de qualité de service et de disponibilité sont maintenus, seront des atouts pour chaque entreprise.

L'objectif de notre travail est de concevoir et mettre en œuvre une application web de suivi du projet, du courrier et d'archivage numérique au sein **CARPA** dans le but de résoudre les problèmes liés au suivi des projets et à la gestion, le suivi et l'archivage du courrier.

Le présent mémoire s'articule autour de cinq chapitres donc le premier est intitulé contexte et problématique dans lequel nous allons vous présenter le **CARPA** ou nous avons effectué notre stage et vous présenter le contexte dans lequel nous avons ressortis la problématique, le second chapitre intitulé les généralités qui présenteront quelques techniques et outils utilisés, le troisième chapitre intitulé analyse et conception consistera à établir les différents besoins de l'application aussi fonctionnels que non fonctionnels, et de faire des modélisations pour mieux comprendre les scénarios de notre application, le quatrième intitulé implémentation présentera la mise en œuvre de la plateforme proprement dite, elle consiste en l'établissement des meilleures méthodes, technologies et outils nécessaires au développement de l'application, ainsi que les dispositions architecturales de cette dernière. Enfin le dernier chapitre portera sur les résultats et commentaires qui consistera à présenter les résultats obtenus.

CHAPITRE 1

CONTEXTE ET PROBLÉMATIQUE

Introduction

Dans l'accomplissement d'un travail scientifique, on s'appuie « toujours » sur des résultats d'autres recherches. Il devient donc très indispensable, dans un tel projet, de présenter les outils et techniques qui ont été utilisés pour parvenir à une bonne fin.

Dans ce chapitre des généralités, nous allons présenter la technique Rapid Application Development, puis l'outil Ruby On Rails qui la met en valeur. Dans un souci de respect des principes du génie logiciel, pour éviter des erreurs de régression et pour mieux gérer les versions, nous utiliserons l'outil Git que nous présentons également dans ce chapitre. L'architecture MVC, le langage UML et la méthode de développement Agile feront l'objet des dernières présentations.

2.1 Rapid Application Development

Le Rapid Application Development (RAD), francisé en « Robot Automatique de Développement », est une méthode de développement rapide de logiciels. Il se fait généralement au moyen des outils tels que des AGL (Atelier de Génie Logiciel) ou des L4G (Langages de Quatrième Génération).

Partant du résultat de la modélisation, ils doivent pouvoir générer une application avec des fonctionnalités basiques telles que l'ajout, la modification et la suppression d'une entité ; également l'affichage des données par liste. On parle généralement de CRUD (Create, Read, Update, Delete).

Ces outils sont des ensembles de programmes permettant la conception de programmes, d'applications ou de systèmes parfois très complexes. Ils sont généralement formés par des langages puissants et évolués, accompagnés d'utilitaires de création d'interfaces graphiques des programmes générés. Exemple : Ruby On Rails, Sinatra, Visual Basic, Django.

Avec un L4G, on programme vite et c'est simple, mais le code généré est souvent lourd et très lent, sans véritable optimisation. De plus, on n'a que rarement accès aux entrailles de son programme, et s'il ne fonctionne pas, on peut mettre beaucoup de temps à diagnostiquer.

Il est donc nécessaire pour le programmeur de trouver l'outil qui répondra à ses attentes. Un outil qui soit compatible au langage utilisé, qui soit le plus « open source » possible et qui procure une architecture familière. Ruby On Rails et Sinatra sont des outils destinés à la plateforme Ruby et qui utilisent des architectures en MVC. Ruby On Rails a particulièrement attiré notre attention.

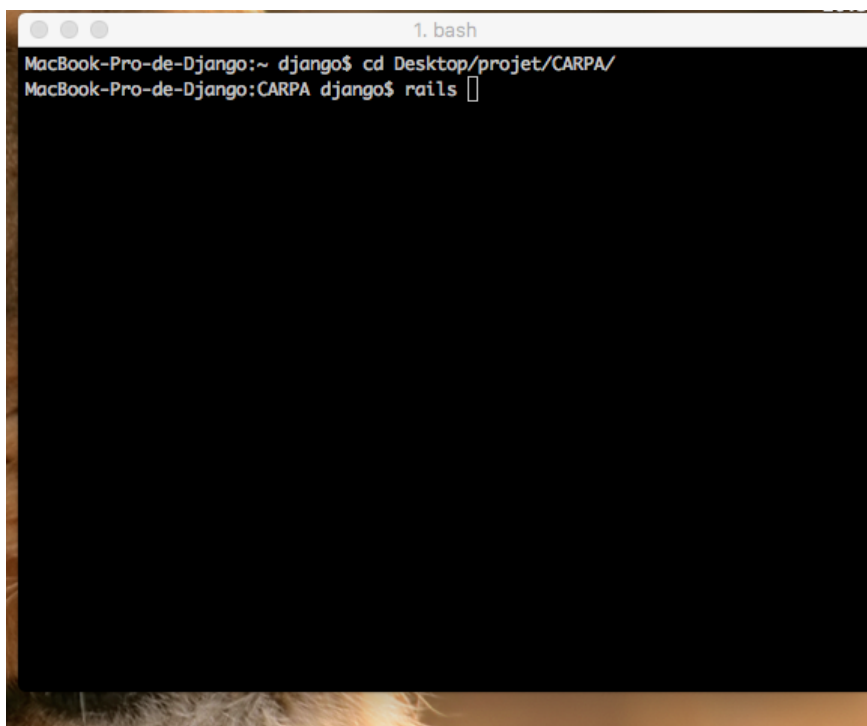
2.2 L'outil Ruby On Rails

Dans le monde Ruby, ROR, c'est-à-dire Ruby On Rails, apporte une réponse à la concurrence des frameworks de développement rapide tels que Groovy on rails ou Django. Il permet de créer rapidement des

applications WEB. Il a été initié en 2005 par David Heinemeier Hansson c'est un framework web open source écrit en Ruby sous licence MIT. Il suit le modèle vue-contrôleur(MVC). Il propose une structure qui permet de développer rapidement et intuitivement(reference : wikipedia.org/wiki/RubyOnRails).

Il s'installe d'une manière très particulière notamment en ligne de commande après avoir installé Ruby. On peut l'utiliser en ligne de commande ou via des IDE qui l'intègrent.

Sa commande principale est **Rails**. L'exécution de cette commande dans une invite de commandes Mac nous donne l'aperçu suivant :



```
1. bash
MacBook-Pro-de-Django:~ django$ cd Desktop/projet/CARPA/
MacBook-Pro-de-Django:CARPA django$ rails
```

FIGURE 2.1 – Terminal

✘ **Les modèles** sont des classes servant à modéliser les données et à établir les relations entre elles. Cela permet d'établir le mapping entre les Objets et la base de données, grâce aux outils tels que **Active Record**.

Il est possible d'effectuer des requêtes de base sur les modèles. On trouve sur chaque modèles des méthodes `save()`, `create()`, `find_by_name()`, `find()` etc... Ces méthodes n'ont jamais été définies par le développeur, mais elles existent grâce au framework.

Il est également possible de contrôler la validation des formulaires depuis le fichier qui définit le domaine.

✘ **Les Contrôleurs**. Un contrôleur est une classe qui reçoit la requête de l'utilisateur et qui, en fonction de l'action demandée, va effectuer le traitement.

Pour gérer quelle est l'action et quel contrôleur est appelé, Rails se base sur le formatage de l'URL : : **http ://<...>/controller/action/**.

✘ **Les Vues** sont représentées par un moteur de template **erb**, ou du **Haml**, on peut insérer du code Ruby.

On constate dans l'arborescence qu'il y'a une vue dédiée à chaque contrôleur et une vue dédiée à une action du contrôleur (Voir Annexe).

2.3 Le Langage Ruby

Le langage Ruby a été conçu, au milieu des années 90, par Yukihiro Matsumoto, un programmeur Japonais. Son objectif était d'avoir un langage qui soit « plaisant » à utiliser :

Ruby is **“made for developer happiness”**! (reference).

C'est un Langage Orienté Objet. la figure.. présente un arbre généalogique de Ruby. Quant au tableau.. il représente les ancêtre de Ruby, avec les principales caractéristiques héritées de ces ancêtres.

Langage	Année	Caractéristiques
Lisp	1958	approche fonctionnelle métaprogrammation
CLU	1974	itérateurs
Smalltalk	1980	langage objet pur, blocs de code GUI, sUnit
Eiffel	1986	Uniform Access Principle
Perl	1987	expressions régulières et pattern matching
Ruby	1993	

TABLE 2.1 – Les ancêtres de Ruby.

La syntaxe de Ruby est faite pour apporter plus de flexibilité au framework Ruby On Rails

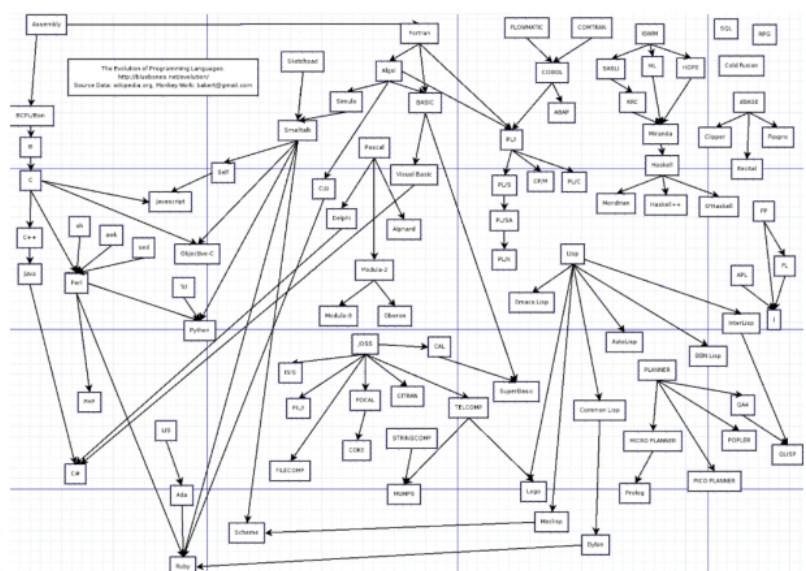


FIGURE 2.2 – Arbre généalogique de divers langages de programmation, incluant Ruby

2.4 Architecture et Création d'un projet avec Ruby On Rails

La création d'un projet ROR se fait en trois étapes :

- ✂ Création des classes de modèle ;
- ✂ Scaffolding ;
- ✂ Running.

2.4.1 Création des classes de modèle

Les «Migrations» sont des définitions du modèle. C'est la première étape dans la création d'un projet Rails.

Nous avons utilisé dans ce projet la version **5.1.6**

La commande **rails new nomProjet** permet de créer un projet nommé **nomProjet**. Rails va construire un **nomProjet** avec un contenu tel que représenté par la figure.

Dès que le projet est créé, on se positionne à l'intérieur en invite de commande (cd nomProjet).

La commande **rails generate modele Etudiant** va créer un fichier **etudiant.rb** dans le dossier **app/models**

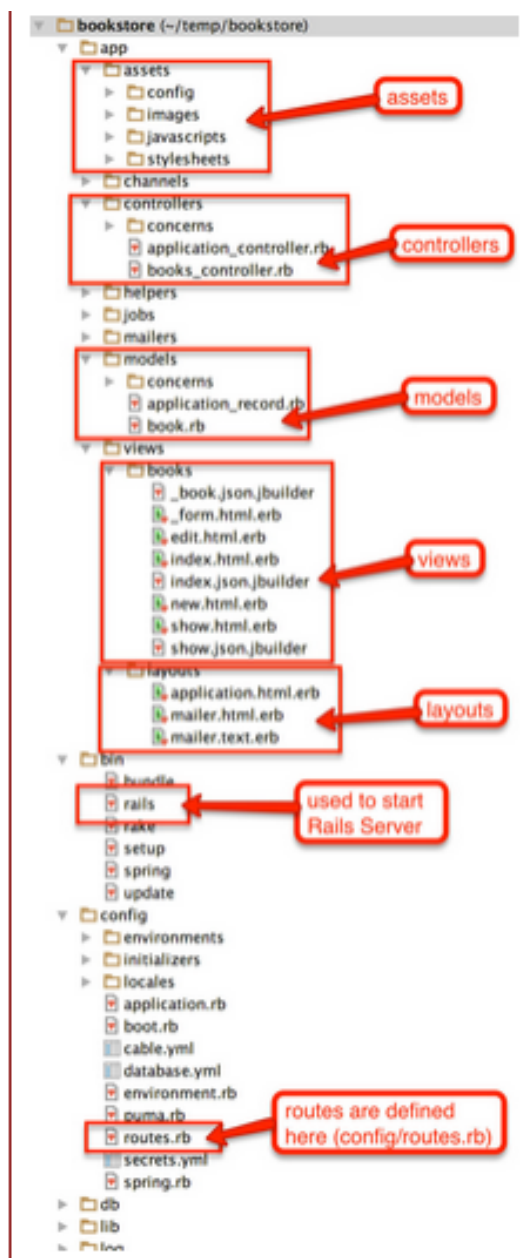


FIGURE 2.3 – Contenu d'un dossier de projet Rails

2.5 Scaffolding

Le scaffolding est une action de mise sur pied d'un échafaudage. C'est là que repose la génération des futures fonctionnalités.

Il se gère au niveau des controllers et permet de spécifier les actions du contrôleur.

La commande **rails** qui permet de créer un contrôleur pour le domaine Etudiant est : **rails generate controller Etududiants**.

Un fichier **EtudiantsController** sera créé dans le dossier **app/controllers**.

Le **scaffold** prend un domaine pour lequel le contrôleur définira les fonctionnalités CRUD. Ainsi, on pourra créer un **Etudiant**, modifier ses champs et le supprimer ; également, l'affichage de la liste des instances d'**Etudiant** créées sera disponible.

2.5.1 Running

L'exécution d'un projet **rails** se fait par la commande **rails server**. On verra sur la console, le message Server Puma Booting **http ://localhost :3000**.

Rails nous invite ainsi à exécuter l'application sur un navigateur à l'adresse indiquée. Le navigateur nous présentera l'interface suivante :

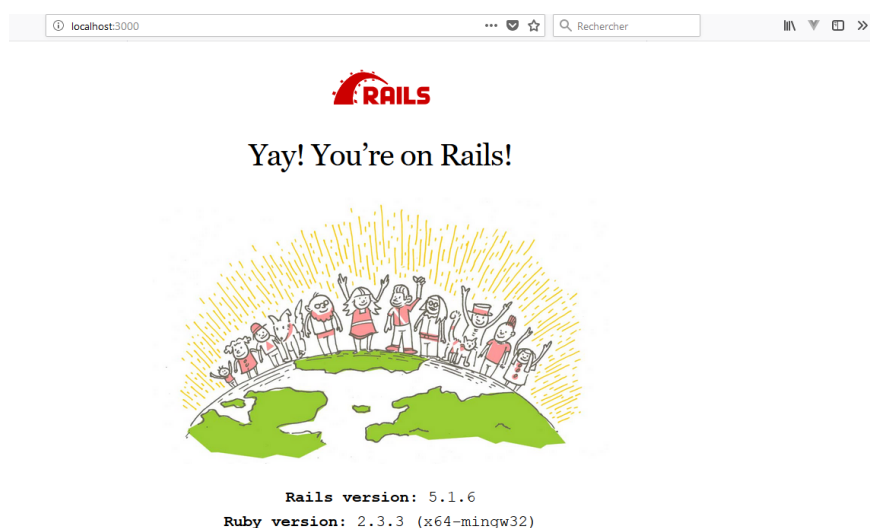


FIGURE 2.4 – Première page d'un projet sur rails

On peut constater que son serveur embarqué utilise par défaut le port 3000. Mais, il peut être modifié si un autre serveur (tomcat par exemple) utilise déjà ce port. Dans ce cas, la commande d'exécution, pour le port 5050 par exemple, sera **rails server.port=5050**.

2.6 Avantages de Rails

Les avantages que l'on peut avoir dans l'utilisation de Grails comme RAD, peuvent être les suivants :

- Web MVC : Facile à utiliser ;
- ERB : Langage de Template simple et complet ;
- Serveur embarqué ;
- ActiveRecord : Modélisation et accès aux données ;
- Base de données simulée (développement) ;
- Internationalisation ;
- Tests & Tests unitaires ;

- Documentation très riche.

2.7 L'outil Git

Il n'est pas toujours évident de parvenir rapidement au but que l'on se fixe dans la réalisation programmes. Pour ne pas rendre la tâche encore plus complexe, il peut être nécessaire d'utiliser des méthodes et outils permettant de ne pas s'éloigner du but. L'outil, Git, que nous présentons dans cette partie nous permet de détecter les erreurs de régression et de gérer les versions, respectivement.

2.7.1 Git

Toujours dans un souci d'efficacité et de rapidité, le programmeur ne doit pas se permettre une mauvaise gestion des versions. Une version est tout simplement l'état du projet à un moment donné. On peut toujours avoir besoin, dans un projet, de retrouver un état antérieur. Git est un outil qui permet de sauvegarder et de restaurer des états.

Lorsque **Git** est bien installé, pour commencer à suivre un projet existant dans **Git**, il suffit de se positionner dans le répertoire du projet et exécuter la commande **git init**. Cela va créer un sous-dossier nommé **.git**. La commande **git add** fichier permet d'indexer les fichiers du projet qui seront suivis en version.

Pour cloner un projet existant, on fait **git clone urlProjet**. La commande **git commit** permet de valider les modifications effectuées sur les fichiers indexés.

Le cycle de vie des états d'un fichier **Git** peut être illustré par la figure ci-dessous :

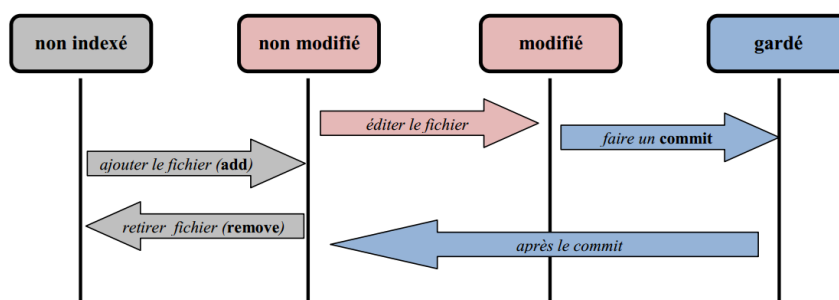


FIGURE 2.5 – Cycle de vie des états d'un fichier dans Git.

2.8 Le pattern MVC

2.8.1 Définition

Un pattern est « une solution à un problème dans un contexte bien défini ». Ses quatre éléments d'un pattern sont :

- ✓ son nom ;
- ✓ son but ;
- ✓ comment il résout le problème ;
- ✓ les contraintes à considérer dans la solution.

Les patterns nous permettent de réutiliser les solutions qui ont marché pour les autres ; « pourquoi réinventer la roue ? ». Ils nous permettent également d'avoir un vocabulaire commun.

2.8.2 Présentation du MVC

MVC (Modèle – Vue – Contrôleur) est un design de conception d'interface utilisateur permettant de découpler le modèle (logique métier et accès aux données) des vues (interfaces utilisateur).

Des modifications de l'un n'auront ainsi, idéalement, aucune conséquence sur l'autre ce qui facilitera grandement la maintenance.

2.8.3 Principe du MVC

Modèle : gère les données et reprend la logique métier (le modèle lui même peut être décomposé en plusieurs couches mais cette décomposition n'intervient pas au niveau de MVC). Le modèle ne prend en compte aucun élément de présentation !

Vue : elle affiche les données, provenant exclusivement du modèle, pour l'utilisateur et/ou reçoit ses actions. Aucun traitement, autre que la gestion de présentation, n'y est réalisé.

Contrôleur : son rôle est de traiter les événements en provenance de l'interface utilisateur et les transmet au modèle pour le faire évoluer ou à la vue pour modifier son aspect visuel (pas de modification des données affichées mais des modifications de présentation).

Le contrôleur « connaît » la (les) vue(s) qu'il contrôle ainsi que le modèle.

Il pourra appeler des méthodes du modèle pour réagir à des événements, il pourra faire modifier à la vue son aspect visuel. Il pourra aussi instancier de nouvelles vues. Pour faire cela, le contrôleur sera à l'écoute d'événements survenant sur les vues.

La vue observera le modèle qui l'avertira du fait qu'une modification est survenue. Dans ce cas, la vue interrogera le modèle pour obtenir son nouvel « état ».

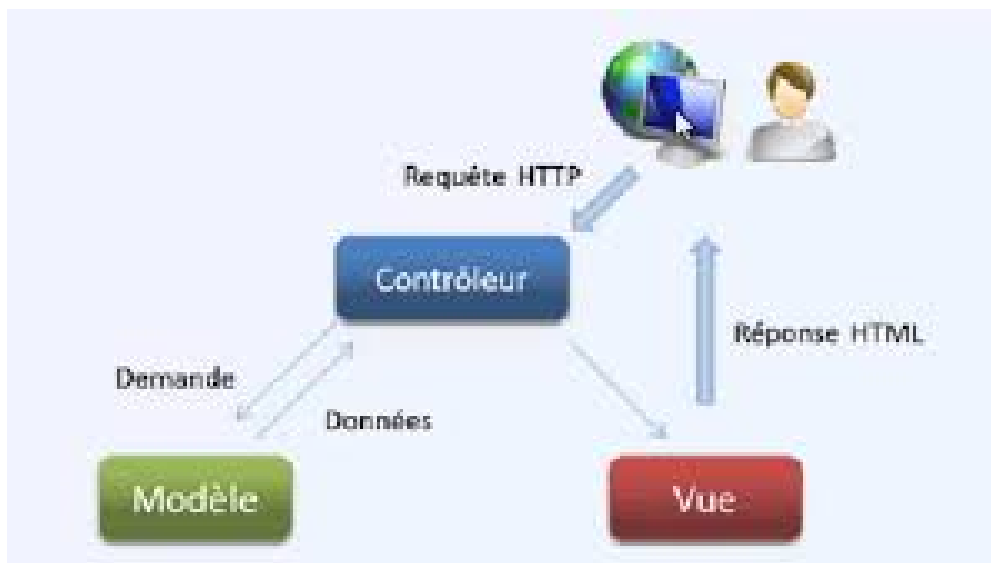


FIGURE 2.6 – Schématisation du MVC

2.9 Constructeur d'un projet UML

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solu-

tions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

UML unifie également les notations nécessaires aux différentes activités d'un processus de développement et offre, par ce biais, le moyen d'établir le suivi des décisions prises, depuis l'expression de besoin jusqu'au codage.

Le fil tendu entre les différentes étapes de construction permet alors de remonter du code aux besoins et d'en comprendre les tenants et les aboutissants. En d'autres termes, on peut retrouver la nécessité d'un bloc de code en se référant à son origine dans le modèle des besoins.

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux grands groupes :

✦ diagrammes structurels :

- Diagramme de classes – Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
- Diagramme d'objets - Il montre les instances des éléments structurels et leurs liens à l'exécution.
- Diagramme de packages - Il montre l'organisation logique du modèle et les relations entre packages.
- Diagramme de structure composite – Il montre l'organisation interne d'un élément statique complexe.
- Diagramme de composants – Il montre des structures complexes, avec leurs interfaces fournies et requises.
- Diagramme de déploiement – Il montre le déploiement physique des « artefacts » sur les ressources matérielles.

✦ Sept diagrammes comportementaux :

- Diagramme de cas d'utilisation - Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude. opérations, généralisations, etc.
- Diagramme de vue d'ensemble des interactions - Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
- Diagramme de séquence - Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
- Diagramme de communication - Il montre la communication entre objets dans le plan au sein d'une interaction
- Diagramme de temps – Il fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
- Diagramme d'activité - Il montre l'enchaînement des actions et décisions au sein d'une activité.
- Diagramme d'états – Il montre les différents états et transitions possibles des objets d'une classe.

2.10 Le Développement Agile

2.10.1 Les principes du Manifeste Agile

La notion de méthode agile est née à travers un manifeste signé en 2001 par 17 personnalités du développement logiciel.

Ce manifeste prône quatre valeurs fondamentales :

- « Personnes et interactions plutôt que processus et outils » : dans l'optique agile, l'équipe est bien plus importante que les moyens matériels ou les procédures. Il est préférable d'avoir une équipe soudée et qui communique, composée de développeurs moyens, plutôt qu'une équipe composée d'individualistes, même brillants. La communication est une notion fondamentale.
- « Logiciel fonctionnel plutôt que documentation complète » : il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est secondaire, même si une documentation succincte et précise est utile comme moyen de communication. La documentation représente une charge de travail importante et peut être néfaste si elle n'est pas à jour. Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).
- « Collaboration avec le client plutôt que négociation de contrat » : le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un feedback continu sur l'adaptation du logiciel à ses attentes.
- « Réagir au changement plutôt que suivre un plan » : la planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premiers releases du logiciel vont souvent provoquer des demandes d'évolution.

2.10.2 La modélisation agile (AM)

La « modélisation agile » prônée par Scott Ambler s'appuie sur des principes simples et de bon sens, parmi lesquels :

- ✦ Vous devriez avoir une grande palette de techniques à votre disposition et connaître les forces et les faiblesses de chacune de manière à pouvoir appliquer la meilleure au problème courant.
- ✦ N'hésitez pas à changer de diagramme quand vous sentez que vous n'avancez plus avec le modèle en cours. Le changement de perspective va vous permettre de voir le problème sous un autre angle et de mieux comprendre ce qui bloquait précédemment.
- ✦ Vous trouverez souvent que vous êtes plus productif si vous créez plusieurs modèles simultanément plutôt qu'en vous focalisant sur un seul type de diagramme.

Conclusion

Dans ce chapitre de généralités, il a été question de présenter certains outils utilisés dans le cadre de notre travail. Nous avons à cet effet présenté la technique du RAD, les outils Rails et Git, l'architecture MVC, le langage UML et la méthode de développement Agile. Notons également que tout le long du projet, nous avons adopté une attitude « Agile », telle que décrite ci-dessus.

CHAPITRE 3

ANALYSE ET CONCEPTION

Introduction

Pour mettre sur pied un logiciel au sein d'une entreprise on passe par la réalisation d'un document appelé cahier de charges.

L'IEEE 1233 (ref ici) définit un cahier de charge comme étant l'expression d'un besoin à satisfaire. Le cahier de charge n'indique pas la manière de réaliser le besoin, ni un produit à fournir, il est en amont de la conception. C'est donc le seul document qui décrit les besoins d'un utilisateur en termes de fonctions à assurer et d'objectifs atteindre.

Dans ce chapitre nous allons en première phase spécifier les besoins fonctionnels pour la mise en place d'une application du suivi de projet, du courrier et d'archivage numérique. Cette phase décrit les exigences auxquelles la solution à mettre en place devra répondre, en termes de fonctionnalités, de caractéristiques fonctionnelles attendues et de la faisabilité du logiciel. La deuxième phase constituera essentiellement la conception de notre application. Il s'agira de l'établissement du diagramme des cas d'utilisation, des diagrammes de séquences et des diagrammes de classes pour mieux comprendre son fonctionnement, afin de maîtriser sa complexité et d'assurer sa cohérence.

3.1 Cahier de Charges

3.1.1 Fonctionnalités attendues de l'application

Dans un début, il est important de recenser les propriétés fonctionnelles de notre application. A la fin de ce projet notre application devrait être capable de :

—

3.1.2 Besoins de l'application

La spécification des besoins décrit sans ambiguïté l'application sécurisée à développer. L'énoncé d'un besoin exprime un comportement ou une propriété que le système doit respecter. Chaque énoncé traduit la présence d'un comportement très spécifique.

1. Un identifiant unique ;
2. La catégorie du besoin qui décrit si celui-ci est fonctionnel ou non fonctionnel ;
3. La description ;

4. Une liste de termes le référant ;
5. La justification de la présence et de l'utilité du besoin ;
6. La priorité du besoin par l'une des appréciations suivantes : haute, moyenne, faible ;
7. La vérification qui décrit un moyen avec lequel le client peut se rassurer tel qu'implémenté, le besoin est remplie.

3.1.3 Besoins fonctionnels

Les besoins fonctionnels expriment des actions que doit effectuer l'application en réponse à une demande (sorties qui sont produites pour un ensemble donné d'entrées) . Pour mieux recenser ces besoins nous avons opté pour un découpage en module.

✂ Le module de gestion des utilisateurs

Dans ce module nous pouvons énumérer les besoins suivants :

- Créer un utilisateur

Identifiant	F1
Catégorie	fonctionnel
Description	Le système permettra l'ajout des données des utilisateurs qui seront les futurs utilis
Termes	Administration, données, utilisateur
Justification	Plusieurs responsables du CARPA utiliseront l'application pour cela, nous devons les identifier afin de leur c
Priorité	Haute
Vérification	Après enregistrement d'un utilisateur, on vérifie dans la liste des utilisateur si ce

TABLE 3.1 – Enregistrement d'un Utilisateur

- Modifier un utilisateur

Identifiant	F2
Catégorie	fonctionnel
Description	Le système permettra de modifier les données d'un utilisateur.
Termes	Administration, données, utilisateur, modification
Justification	L'administrateur peut mal saisir les données
Priorité	Moyenne
Vérification	Après la modification d'un utilisateur, on vérifie dans la liste des utilisateurs si la modification est effective.

TABLE 3.2 – Modifier un utilisateur

- Supprimer un utilisateur

Identifiant	F3
Catégorie	fonctionnel
Description	Supprimer un utilisateur
Termes	Administration, données, utilisateur, suppression
Justification	Le CARPA peut décider de rompre le contrat de un des responsable ou ce dernier peut être mis à la retraite
Priorité	Moyenne
Vérification	Après la suppression d'un utilisateur, on vérifie si l'utilisateur n'est plus dans la liste

TABLE 3.3 – Suppression d'un Utilisateur

- Créer un compte d'utilisateur

Identifiant	F4
Catégorie	fonctionnel
Description	Le système permettra l'ajout des comptes d'utilisateurs.
Termes	Administration, donnée, compte
Justification	Plusieurs utilisateurs sont dans le système et chacun doit avoir un compte pour se connecter à l'application
Priorité	Haute
Vérification	Après enregistrement d'un compte, on vérifie dans la liste des comptes si ce dernier existe.

TABLE 3.4 – Enregistrer un compte utilisateur

- Modifier un compte

Identifiant	F5
Catégorie	fonctionnel
Description	Le système permettra de modifier les données d'un compte.
Termes	Administration, donnée, compte, modification
Justification	L'administrateur peut mal saisir les données, ou alors le profil de l'utilisateur peut changer.
Priorité	Moyenne
Vérification	Après la modification d'un compte, on vérifie dans la liste des comptes si ce dernier existe.

TABLE 3.5 – Enregistrer un compte utilisateur

- Effectuer des recherches sur les utilisateurs

Identifiant	F6
Catégorie	fonctionnel
Description	Le système permettra de faire une recherche rapide sur les utilisateurs.
Termes	Administration, données, utilisateur, modification, recherche.
Justification	L'administrateur peut avoir besoin des informations sur une personne.
Priorité	Moyenne
Vérification	Si la personne existe dans le système alors elle sera affichée.

TABLE 3.6 – Rechercher un utilisateur

✦ Le module de gestion du courrier

Identifiant	F7
Catégorie	fonctionnel
Description	Le système permettra de faire une recherche rapide sur les courriers.
Termes	Administration, données, courrier.
Justification	L'administrateur ou l'utilisateur peut avoir besoin des informations sur un courrier.
Priorité	Haute
Vérification	Si le courrier existe dans le système alors il sera affiché.

TABLE 3.7 – Enregistrer un courrier

• Modifier un courrier

Identifiant	F8
Catégorie	fonctionnel
Description	Le système permettra de faire une modification sur les courriers.
Termes	Administration, données, courrier, modification.
Justification	L'administrateur ou l'utilisateur peut avoir besoin des informations sur un courrier.
Priorité	Moyenne
Vérification	Après la modification, on vérifie dans la liste des courriers si ce dernier est à jour.

TABLE 3.8 – Modifier un courrier

• Supprimer un courrier

Identifiant	F9
Catégorie	fonctionnel
Description	Le système permettra de faire la suppression d'un courrier.
Termes	Administration, données, courrier, suppression.
Justification	CARPA peut décider de ne plus avoir besoin des informations d'un courrier.
Priorité	Moyenne
Vérification	Après la suppression , on vérifie dans la liste des courriers si ce dernier a été supprimer.

TABLE 3.9 – Supprimer un courrier

✧ Le module suivi des projets

• Enregistrer un projet

Identifiant	F10
Catégorie	fonctionnel
Description	Le système permettra de faire l'enregistrement d'un projet.
Termes	Administration, données, projet.
Justification	L'administrateur a besoin des données qui permettrons de faire le suivi des projets
Priorité	Haute
Vérification	Après l'enregistrement , on se rassure que ce dernier se est bien dans la liste.

TABLE 3.10 – Enregistrer un projet

• Modification d'un projet

Identifiant	F11
Catégorie	fonctionnel
Description	Le système permettra de faire la modification d'un projet.
Termes	Administration, données, projet, modification.
Justification	L'administrateur ou l'utilisateur peut faire une erreur dans la saisie d'une d'un projet et souhaite faire une
Priorité	Moyenne
Vérification	Après la modification d'une phase du projet, on verifie s'il est dans la liste des phases du projet

TABLE 3.11 – Modifier un projet

• Supprimer un projet

Identifiant	F12
Catégorie	fonctionnel
Description	Le système permettra de faire la suppression d'un projet.
Termes	Administration, données, projet, suppression.
Justification	CARPA peut décider de ne plus avoir besoin des informations d'un projet
Priorité	Moyenne
Vérification	Après la suppression, on vérifie dans la liste des projets si ce dernier a disparu.

TABLE 3.12 – supprimer un projet

✕ Le module gestion des phase du projet

- Enregistrer une phase du projet

Identifiant	F13
Catégorie	fonctionnel
Description	Le système permettra de faire l'enregistrement d'une phase du projet.
Termes	Administration, données, phase projet.
Justification	L'administrateur ou l'utilisateur a besoin des données concernant les phase du projet pour le suivi des projets.
Priorité	Haute
Vérification	Après l'enregistrement , on vérifie dans la liste des phases du projet ce dernier est bien enregistrée .

TABLE 3.13 – Enregistrer une phase du projet

- Modifier une phase du projet

Identifiant	F14
Catégorie	fonctionnel
Description	Le système permettra de faire la modification d'une phase du projet.
Termes	Administration, données, phase du projet, modification.
Justification	L'administrateur ou l'utilisateur peut faire une erreur dans la saisie d'une phase du projet et souhaite une modification.
Priorité	Moyenne
Vérification	Après la modification d'une phase du projet, on vérifie s'il est dans la liste des phases du projet.

TABLE 3.14 – Modifier un projet

- Supprimer une phase du projet

Identifiant	F15
Catégorie	fonctionnel
Description	Le système permettra de faire la suppression d'une phase du projet.
Termes	Administration, données, phase du projet, suppression.
Justification	CARPA peut décider de ne plus avoir besoin des informations d'une phase du projet
Priorité	Moyenne
Vérification	Après la suppression, on vérifie dans la liste des phases du projet si ce dernier a disparu.

TABLE 3.15 – supprimer une phase du projet

✂ Le module archivage

- Enregistrer une Archive

Identifiant	F16
Catégorie	fonctionnel
Description	Le système permettra de faire l'enregistrement d'une archive.
Termes	Administration, données, archive.
Justification	L'administrateur ou l'utilisateur a besoin des données concernant les archives.
Priorité	Haute
Vérification	Après l'enregistrement, on vérifie dans la liste des archives si ce dernier est bien enregistré.

TABLE 3.16 – Enregistrer une archive

- Modifier une phase du projet

Identifiant	F17
Catégorie	fonctionnel
Description	Le système permettra de faire la modification d'une archive.
Termes	Administration, données, archive, modification.
Justification	L'administrateur ou l'utilisateur peut faire une erreur dans l'enregistrement d'une archive et souhaite une modification.
Priorité	Moyenne
Vérification	Après la modification d'une archive, on vérifie s'il est dans la liste des archives.

TABLE 3.17 – Modifier une archive

- Supprimer une archive

Identifiant	F15
Catégorie	fonctionnel
Description	Le système permettra de faire la suppression d'une archive
Termes	Administration, données, archive, suppression.
Justification	CARPA peut décider de ne plus avoir besoin des informations concernant une archive.
Priorité	Moyenne
Vérification	Après la suppression, on vérifie dans la liste des archives si ce dernier a disparu.

TABLE 3.18 – supprimer une archive

✱ Le module administration des données

- Importer les données sous format excel, pdf, doc etc

Identifiant	F15
Catégorie	fonctionnel
Description	Le système permettra aux utilisateurs d'importer les données sous format excel, pdf, doc etc.
Termes	Administration, données, importé.
Justification	Ceci facilite l'administration des données et la production des rapports.
Priorité	Moyenne
Vérification	Après importation des données, ces dernières apparaissent dans la liste.

TABLE 3.19 – Importer les données

- Exporter les données sous format excel, pdf, json, xml

Identifiant	F15
Catégorie	fonctionnel
Description	Le système permettra aux utilisateurs d'exporter les données sous format excel, pdf, json, xml .
Termes	Administration, données, exporté.
Justification	Ceci facilite la production des rapports.
Priorité	Moyenne
Vérification	Après l'exportation un document est produit en fonction du format choisi.

TABLE 3.20 – Exporter les données

3.1.4 Besoins non fonctionnels

Les besoins non spécifiques sont des besoins qui font référence aux aspects généraux de l'interface utilisateur. ce sont des besoins en matière de performance, de type de matériel ou le type de conception.

✓ Ergonomie et convivialité

L'application devra disposer d'une interface conviviale, facile et agréable à utiliser et à comprendre, permettant ainsi une appropriation rapide et intuitive par ses utilisateurs. Le portail doit être simple et transparent, laissant les utilisateurs prendre le contrôle de l'application.

✓ Sécurité et fiabilité

L'application devra envoyer des notifications pour toutes tentatives d'usurpation des droits de privilèges ou lorsqu'un champs est mal rempli. L'application devra restreindre l'utilisation de ses fonctionnalités sur le principe de responsabilité des utilisateurs et des moindres privilèges. Le droit d'accès aux ressources doit se fonder sur le principe de moindres privilèges et de la responsabilité des utilisateurs.

✓ **Documentation**

L'application devra être documentée :

- ✦ Au niveau du code proprement dit (commentaire des lignes de code), pour permettre en cas de besoin de comprendre, modifier ou changer le code ; ceci nous permet d'assurer la maintenance qui est l'une des bonnes qualités d'un logiciel.
- ✦ Au niveau de l'interface, considérée comme aide dans notre cas, elle permet à tout utilisateur de mieux s'accommoder aux portails afin d'assurer l'utilisabilité.
Les documents qui accompagneront la livraison de l'application sont :

- Document de conception ;
- Document de modélisation ;
- Guide de l'utilisation.

✓ **Evolution, maintenance et réutilisation**

Pour faciliter l'évolutivité, la réutilisation et la maintenance, l'application devra être développée suivant une architecture choisie sur la base des technologies utilisées pour son développement ; mais aussi, la séparation en module devra être prise en compte pour mieux cerner les fonctionnalités qui y sont implémentées. Et enfin les conventions de codage telles que l'utilisation des noms significatifs, majuscules pour les constantes etc. devront être respectées.

3.1.5 Etude de faisabilité

3.1.5.1 Justification du projet

La mise en place de cette plateforme permettra à l'entreprise de profiter au maximum des avantages suivants :

- ✓ **Gagner en temps et limiter les coûts et dépenses :** Tous les membres de l'entreprise signaleront à temps les différents problèmes rencontrés et ceux-ci seront pris en charge et résolus dans les brefs délais.
- ✓ **Augmenter la productivité :** il est clair que la résolution des problèmes à temps et dans les brefs délais augmente la productivité de l'entreprise.

3.1.5.2 Analyse des coûts et bénéfices

Les bénéfices de l'utilisation d'une telle application ont déjà été énumérés dans le paragraphe précédent. Cependant, en termes de coûts :

3.1.5.3 Planning prévisionnel

4.1 Introduction

Dans ce chapitre nous entamons la partie pratique, ou nous allons présenter l'environnement et les outils de développement utilisés, l'architecture de l'application et quelques interfaces de celui-ci. Vue la complexité de notre système après analyse, conception et modélisation il ne sera évident pour nous de l'implémenter comme cela a été prévu dans le cahier de charge tout en respectant les délais prescrit. En se basant sur l'étude de l'existant faite au **chapitre 2** dans les généralités, nous avons opté pour l'utilisation d'une solution open source qui couvre le maximum de fonctionnalités exigées, et la modification pour satisfaire aux besoins de l'entreprise.

4.2 Langages de programmation et Environnement de développement

4.2.1 Langages de programmation et Framework

1. Ruby

Le langage Ruby a été conçu, au milieu des années 90, par Yukihiro Matsumoto, un programmeur Japonais. Son objectif était d'avoir un langage qui soit « plaisant » à utiliser, (référence ici) c'est un langage open source dynamique et orienté objet avec une syntaxe flexible. La version utilisée est la **2.5.1**.

Le langage Ruby compte diverses manières d'utilisation (référence ici) :

- Pour une interface web : c'est l'utilisation la plus courante ;
- En ligne de commandes (CLI "Command Line Interface") ;
- Pour produire une interface desktop (GUI "Graphical User Interface ").

2. JavaScript

JavaScript est un langage de script orienté objet principalement utilisé dans les pages HTML. A l'opposé des langages serveurs (qui s'exécutent sur le site), Javascript est exécuté sur l'ordinateur de l'internaute par le navigateur lui-même. Ainsi, ce langage permet une interaction avec l'utilisateur en fonction de ses actions (lors du passage de la souris au dessus d'un élément, du redimensionnement de la page, etc). La version standardisée de Javascript est l'ECMAScript. (référence ici)

3. HTML/XHTML

Le HTML et sa variante plus stricte XHTML sont des langages de balisage des pages Web. Il n'y a pas si longtemps, le HTML servait à définir aussi bien la structure des pages que leur présentation

visuelle. Aujourd'hui, ces deux aspects doivent être bien distincts et le X/HTML est destiné uniquement à représenter la structure d'une page : titres, sous-titres, paragraphes, images, formulaires de saisie, liens hypertextes, etc.

4. CSS

CSS (Cascading Style Sheets, ou feuilles de styles en cascade) permet de modifier la présentation des éléments X/HTML : couleur, taille, police de caractères, mais aussi position sur la page, largeur, hauteur, empilement, bref tout ce qui touche à la mise en page d'un document X/HTML.

5. Bootstrap

Bootstrap est un framework CSS, mais pas seulement, puisqu'il embarque également des composants HTML et JavaScript. Il comporte un système de grille simple et efficace pour mettre en ordre l'aspect visuel d'une page web. Il apporte du style pour les boutons, les formulaires, la navigation...etc. Il permet ainsi de concevoir un site web rapidement et avec peu de lignes de code ajoutées. Le framework le plus proche de Bootstrap est sans doute Foundation qui est présenté comme « The most advanced responsive front-end framework in the world » (référence ici)

6. Ruby On Rails

Ruby On Rails est un framework libre distribué sous licence MIT destiné au prototypage et au développement d'application Web diverses et variées, il est multiplateforme . ROR (Ruby On Rails) est orienté RAD (Rapid Application Development)(référence ici), il reprend de nombreux principes communs aux méthodes "Agiles" et est basé sur l'architecture MVC et ceci se voit dès l'arborescence d'un projet ROR .

7. JQUERY

jQuery est un framework JavaScript libre et Open Source, implanté côté client, qui porte sur l'interaction entre le DOM(Document Object Model), JavaScript, AJAX et le Html. Cette librairie JavaScript a pour but de simplifier les commandes communes du JavaScript. La devise de jQuery est en effet, "Écrire moins pour faire plus" (write less do more).

jQuery, du moins à l'origine, est l'œuvre d'un seul homme : John Resig. Ce jeune surdoué de JavaScript développa la première version de jQuery en janvier 2006.

Les spécificités de jQuery sont nombreuses mais l'essentielle est assurément la souplesse qu'il apporte pour accéder à tous les éléments du document Html grâce à la multitude de sélecteurs mis en place. Cette caractéristique fut d'ailleurs retenue pour donner un nom à ce framework : j pour JavaScript et Query pour chercher ou accéder aux éléments.(référence ici)

8. XML

XML (eXtensible Markup Language) est un langage de balisage, qui se distingue de HTML par le fait qu'il permet de spécifier la structure du contenu d'un document plutôt que la façon de le présenter.

9. SQL SQL (Structured Query Language) est un langage d'interrogation de base de données très populaire. Il constitue aujourd'hui une norme implémentée par de nombreux SGBD (Systèmes de Gestion de Bases de Données).

4.2.2 Outils de développement

1. SQLITE

Sqlite est le système de gestion de base de données (SGBD) que nous avons utilisé pour gérer notre base de données car il respecte le modèle d'architecture 3-tiers. Il permet de créer facilement nos tables, il est sécurisé, rapide et léger, qui fonctionne sur de nombreux systèmes d'exploitation (dont Linux, Mac OS, Windows, Solaris, FreeBSD...) et qui est accessible en écriture par de nombreux

langages de programmation, incluant notamment Ruby, C, C++ ,PHP, Java, .NET, Python .

2. RubyMine

RubyMine est un environnement de développement intégré (IDE), qui est cross-plateform (utilisable sur n'importe quel système), il comprend toutes les caractéristiques d'un IDE moderne (éditeur couleur, projet multiplateforme et de page web) intègre le système de contrôle de version et bien d'autre outils à son sein, il est payant mais néanmoins donne une version d'essai de 30 jours et une licence pour étudiant.

3. ImageMagick

ImageMagick est un logiciel en ligne de commande très puissant de manipulation d'image dans pratiquement tous les formats existant. Sa licence est compatible avec la licence GPL (general public licence).

4. **StarUml** StarUml est un logiciel de modélisation UML, qui nous a permis de modéliser nos différents diagrammes

4.3 Présentation du projet

L'arborescence du projet représente ici l'empaquetage et l'organisation des différents fichiers du projet. Le projet Java que nous avons créé sur RubyMine est constitué en module, en fonction de leurs utilités. Les captures ci-dessous présentent cette arborescence, d'abord dans un explorateur Windows, et ensuite dans l'IDE RubyMine.

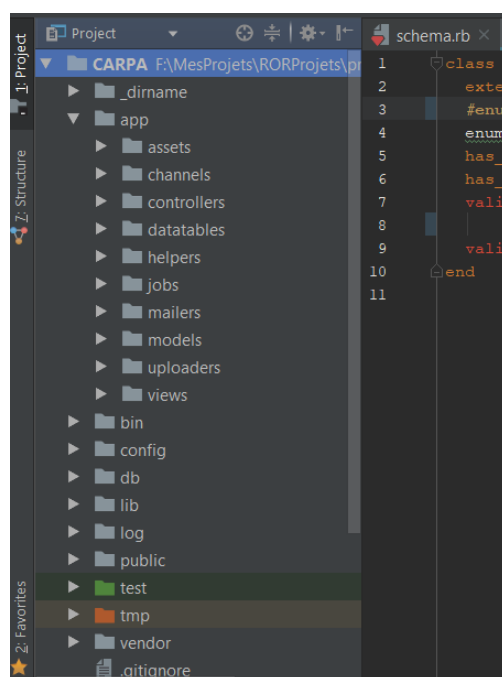


FIGURE 4.1 – Arborescence du projet dans l'IDE RubyMine

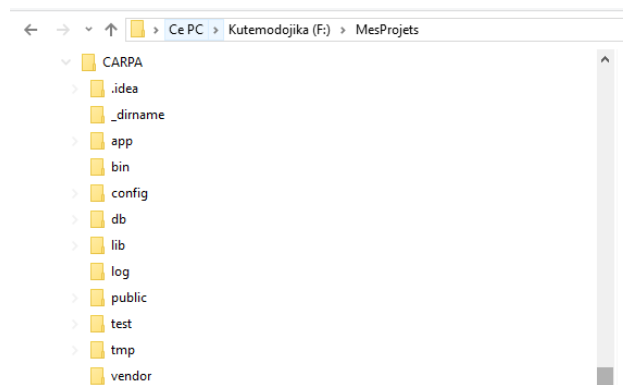


FIGURE 4.2 – Arborescence du projet dans l'explorateur windows

4.4 Modèle Architectural

4.4.1 Synoptique de fonctionnement d'une application web

La figure ci-dessous présente le principe de fonctionnement de notre application web. Dans cette architecture, nous avons d'un côté, un ordinateur ayant notre application web et de l'autre, un serveur qui communique avec notre ordinateur à travers des web services.

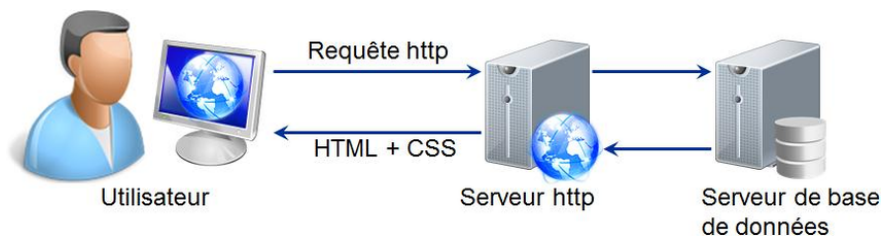


FIGURE 4.3 – Synoptique de l'application web

4.4.2 Synoptique de fonctionnement d'une application web en Ruby On Rails

La figure ci-dessous présente l'architecture générale du système en Ruby On Rails. On y ressort une ORM Active Record dans notre cas, qui permet de faire le mapping relationnel consistant à modéliser la base de données sous forme d'objets pour une manipulation plus simple à travers le code Ruby.

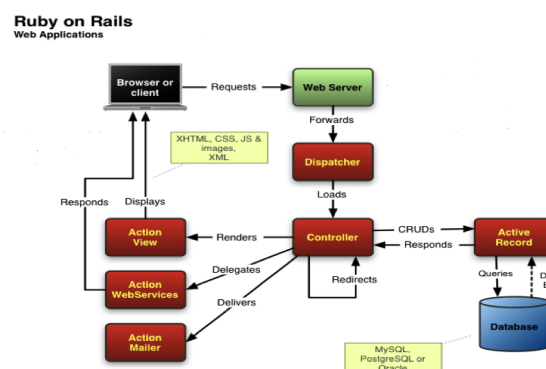


FIGURE 4.4 – Synoptique d'une application web en Ruby On Rails

Le contrôleur s'occupe de l'accès aux données. Nous implémentons aussi dans cette couche les algorithmes métier de notre système. Elle ne change pas si on modifie l'interface utilisateur ou la façon d'accéder aux données nécessaire au fonctionnement de l'application.

L'interface utilisateur est l'interface web qui permet à l'utilisateur de piloter l'application et d'en recevoir des informations.

4.5 Arborescence de l'application

Nous avons dans ce chapitre dévoilé ce qui a été utilisé pour résoudre la problématique générale du suivi des projets du courrier et d'archivage. Mais aussi, pour la satisfaction de chaque besoin émis par le cahier de charges. Il ne nous reste plus qu'à présenter le résultat de tout cela dans le chapitre qui suit.

CHAPITRE 5

RÉSULTATS ET COMMENTAIRES

CONCLUSION ET PERCEPTIVES