# DÉDICACE

Nous dédions ce travail à nos parents.

P a g e

# REMERCIEMENTS

Nous adressons nos sincères remerciements :

* A notre Directeur de mémoire, le Dr KOSSINGOU Gislain qui, tout au long de cette période de travail, a été attentif et patient en notre égard, tout en nous accompagnons jusqu’au bout ;
* Au Directeur de l’Institut Supérieur de Technologie, le Pr. M’BOLIGUIPA Jean pour les efforts fournis chaque année pour nos études ;
* Au Directeur des études, PhD M’BOUANA Landry Privace Noé, qui déploie de grands efforts chaque année pour garantir aux étudiants, une formation fiable et actualisée ;
* Au chef de département, Mr GOUMAPE Innocent, ainsi qu’à tout le corps professoral qui a contribué à notre formation au sein de l’Institut Supérieur de Technologie.

Nos remerciements vont également à l’endroit de l’IPB, ainsi nous tenons à remercier :

* Notre Chef de stage, le Dr YAMBIYO Brice qui a su nous encadrer et nous soutenir ;
* Le Pr Alexandre, le chef de notre service à l’épidémiologie ;
* Le DG de l’IPB, le Pr YAP BOUM II qui a accepté de nous accueillir au sein de l’Institut afin que nous effectuions notre stage ;
* Ainsi que tout le personnel du service d’épidémiologie.

Nous portons également nos plus profonds remerciements à nos très chers parents qui n’ont jamais cessé de nous accompagner dans nos études et ont su faire preuve d’une grande patience et d’un grand amour dans notre éducation.

Nous remercions également nos frères et nos sœurs ainsi que toutes les personnes qui ont contribué de près ou de loin à l’élaboration de ce mémoire.

P a g e

# AVANT-PROPOS

L’Institut Supérieur de Technologie (IST), appelée auparavant Institut Universitaire Technologique des Mines et Géologies (IUTMG), a été fondé en 1973. Dix-neuf ans après (en 1992), celle-ci prit le nom de l’Institut Supérieur de Technologie. L’IST, est un établissement de l’Université de Bangui qui a pour but d’offrir des formations professionnelles à ses étudiants. L’entrée dans cette enceinte réservée aux élites, est sanctionnée par un test ; ce qui a pour objectif de sélectionner les meilleurs afin de leur faire bénéficier d’une formation de haute qualité.

L’institut Supérieur de Technologie offre des formations dans plusieurs domaines. Celle-ci est répartie en cinq grands départements dont :

* Le département de Génie Civil : ce département a pour objectif d’apporter aux étudiants une formation dans le cadre de l’Ingénierie en Bâtiment et Travaux Publiques ;
* Le département de Génie Informatique : les deux options qui composent ce département, c’est-à-dire l’option de Réseaux et Systèmes Informatiques et l’Informatique de Gestion, permettent aux étudiants de suivre une formation dans le domaine de l’IT et des gestions ;

* Le département de Génie Industriel : dont l’objectif est de former de futurs ingénieurs dans le cadre des Travaux en Télécommunication et en électromécanique ;
* Le département de Génie Minier et Géologie : a pour objectif de donner à ses étudiants une formation dans l’ingénierie des mines exploitation et mine environnementale ;

* Le département de Génie Pétrolier : au sein duquel sont formés de futurs ingénieurs en travaux de pétrole ;

A la fin d’un Cycle de trois ans (03) de formation, les étudiants de l’Institut Supérieur de Technologie bénéficient d’une Licence professionnelle à condition que ceux-ci passent un stage dans une entreprise et soutiennent devant un jury composé de trois membres.

P a g e

# SOMMAIRE

[DÉDICACE i](#_Toc88535)

[REMERCIEMENTS ii](#_Toc88536)

[AVANT-PROPOS iii](#_Toc88537)

[SOMMAIRE iv](#_Toc88538)

[Liste des sigles et abréviations v](#_Toc88539)

[Liste des tableaux vi](#_Toc88540)

[Liste des figures vii](#_Toc88541)

[INTRODUCTION GÉNÉRALE 1](#_Toc88542)

[PARTIE 1 : PRÉSENTATION GÉNÉRALE 1](#_Toc88543)

[CHAPITRE I : Présentation de la structure d’accueil 2](#_Toc88544)

[CHAPITRE II : Analyse de l’existant 4](#_Toc88545)

[PARTIE 2 : ÉTUDES THÉORIQUES 7](#_Toc88546)

[Chapitre III : Étude des concepts et méthodes 8](#_Toc88547)

[Chapitre IV : Étude comparative et choix de solutions 31](#_Toc88548)

[Chapitre V : Conception et Modélisation de l’application 36](#_Toc88549)

[PARTIE 3 : IMPLÉMENTATION 52](#_Toc88550)

[Chapitre VI : Mise en œuvre 53](#_Toc88551)

[BIBLIOGRAPHIE I](#_Toc88552)

[WEBOGRAPHIE II](#_Toc88553)

[ANNEXES III](#_Toc88554)

[TABLES DES MATIERES VII](#_Toc88555)

P a g e

# Liste des sigles et abréviations

|  |  |
| --- | --- |
| SIGLES | SIGNIFICATIONS |
| API | Application Programming Interface |
| REST | Representational State Transfert |
| HTTP | Hyper Text Transfert Protocol |
| HTTPS | Hyper Text Transfert Protocol Secured |
| URL | Uniform Ressource Locator |
| SOAP | Simple Object Access Protocol |
| MVC | Modèle Vue Contrôleur |
| MVT | Modèle Vue Template |
| IA | Intelligence Artificielle |
| IPB | Institut Pasteur de Bangui |
| SQL | Structured Query Language |
| SGBD | Systèmes de Gestion de Bases de Données |
| SGBDR | Systèmes de Gestion de Bases de Données Relationnelles |
| WWW | World Wide Web |
| BDD | Base De Données |
| HTML | Hyper Text Markup Langage |
| DRY | Don’t Repeat Your Self |
| CSV | Comma Separated Values |
| ORM | Object Relational Mapping |
| SGBDOO | Système de Gestion de Base de données Orienté Object |
| SqlLite | Structured Query Language Lite |
| XML | Extensible Markup Language |
| URI | Uniform Resource Identifier |
| CSS | Cascading Style Sheet |
| JSON | Javascript Object Notation |
| UML | Unified Modeling Language |
| UP | Unified Process |

P a g e

# Liste des tableaux

Tableau 1- Coût théorique du projet .................................................................................................... 6

Tableau 2- Etude comparative UML/MERISE .................................................................................. 31

Tableau 3- Etude comparative Django/Flask ..................................................................................... 32

Tableau 4- Etude comparative API REST/ API SOAP ...................................................................... 33

Tableau 5- Description détaillée du cas d'utilisation : gérer utilisateurs ............................................ 40

Tableau 6- Description détaillée de cas d'utilisation : Gérer projet ................................................... 41

Tableau 7- Description détaillée du cas d'utilisation : Gérer équipe .................................................. 42

Tableau 8- Description détaillée du cas d'utilisation : Recevoir notifications ................................... 45

Tableau 9- Description détaillée du cas d'utilisation : S'authentifier ................................................. 46

Tableau 10- Mesures de sécurité intégrées à Django ......................................................................... 60

Tableau 11- Installation de quelques bibliothèques ........................................................................... IV

P a g e

# Liste des figures

Figure 1- Communication entre client et serveur................................................................................. 9

Figure 2- Communication client et serveur avancée .......................................................................... 11

Figure 3- Fonctionnement d'un site web statique .............................................................................. 12

Figure 4- Fonctionnement d'un site web dynamique ......................................................................... 13

Figure 5- Architecture MVC .............................................................................................................. 18

Figure 6- Architecture MVT .............................................................................................................. 19

Figure 7- Fonctionnement général d'une API .................................................................................... 23

Figure 8- Fonctionnement d'une API REST ...................................................................................... 26

Figure 9- Diagramme de cas d'utilisation : Gérer utilisateurs ............................................................ 41

Figure 10- Diagramme de cas d'utilisation : Gérer projet .................................................................. 42

Figure 11- Diagramme de cas d'utilisation : Gérer projet (Chef de projet) ....................................... 43

Figure 12- Cas d'utilisation Gérer équipe du chef de d’équipe .......................................................... 44

Figure 13- Cas d'utilisation consulter informations des membres ..................................................... 44

Figure 14- Cas d'utilisation Consulter les informations par informations par le DG ........................ 45

Figure 15- Diagramme de classe ........................................................................................................ 47

Figure 16- Séquence du cas d'utilisation : S'authentifier ................................................................... 48

Figure 17- Séquence du cas d'utilisation : Ajouter Utilisateur ........................................................... 49

Figure 18- Séquence du cas d'utilisation : Modifier Equipe .............................................................. 50

Figure 19- Séquence du cas d'utilisation : Recevoir Notification ...................................................... 51

Figure 20- Diagramme d'activité Modifier Equipe ............................................................................ 52

Figure 21- Page de Login de l'espace d'Administration..................................................................... 62

Figure 22- Tableau de bord partie Administration ............................................................................. 62

Figure 23- Gestion des utilisateurs..................................................................................................... 63

Figure 24- Ajout d'un compte utilisateur ........................................................................................... 63

Figure 25- Gestion des permissions ................................................................................................... 64

Figure 26- Interface de gestion de l'API ............................................................................................ 64

Figure 27- Espace de gestion des projets ........................................................................................... 65

Figure 28- Formulaire d'ajout de projet ............................................................................................. 66

Figure 29- Espace de gestion des données des projets ....................................................................... 66

Figure 30- Statistique des données récoltées ..................................................................................... 67

Figure 31- Espace de visualisation des KPI d'un projet ..................................................................... 67

Figure 32- Page d'authentification des membres du projet ................................................................ 68

Figure 33- Tableau de bord des membres du projet ........................................................................... 68

Figure 34- Formulaire du projet ......................................................................................................... 69

Figure 35- Organigramme de l'IPB .................................................................................................... III

Figure 36- Code source des modèles .................................................................................................. V

Figure 37- Code source des vues ........................................................................................................ V

Figure 38- Code source des Template ................................................................................................ VI

P a g e

# INTRODUCTION GÉNÉRALE

A l’ère technologique florissante dans laquelle nous vivons, la gestion de projet n’a jamais été aussi dynamique et essentielle. Les projets deviennent de plus en plus complexes et collaboratifs, ce qui rend certains outils traditionnels inefficaces et insuffisants pour la gestion. Les entreprises font alors face à des difficultés qui les obligent à vouloir des solutions plus flexibles et évolutives, qui seraient capables de s’adapter à leurs besoins spécifiques.

Au cœur de l’Institut Pasteur de Bangui, où la recherche et l’innovation sont au rendez-vous au quotidien, l’efficacité et la fluidité de la gestion de projet sont primordiales. Une application de gestion de projet RESTFul est une application Web qui utilise l’architecture REST (Representational State Transfert) pour gérer les données des projets. Cette architecture permet aux utilisateurs d’accéder aux données des projets de manière uniforme et indépendante de la plateforme ou du langage de programmation utilisé. Les applications de gestion de projet RESTFul offrent de nombreux avantages, notamment en termes de flexibilité, d’évolutivité et de facilité d’utilisation.

L’objectif principal de ce travail est de développer une application de gestion de projet RESTFul qui soit non seulement fonctionnelle et efficace, mais aussi agréable à utiliser et facile à intégrer dans le flux de travail de l’institut Pasteur de Bangui. Elle permettra donc de faire la gestion des projets, mais aussi la collecte des données. Cette application doit donc s’intégrer facilement aux solutions déjà existantes.

A travers les objectifs que nous nous sommes fixés, la question qui se pose est de savoir comment nous allons nous y prendre pour développer une telle application ? Nous nous engagerons donc dans un travail dynamique qui sera structuré de la manière suivante : La première partie abordera une présentation générale de l’Institut Pasteur de Bangui et d’une analyse d’existant de la solution. La deuxième partie se concentrera principalement sur une étude théorique ; et enfin, la dernière partie concernera l’implémentation avec la mise en œuvre concrète de l’application.

P a g e 1

Développement d’une application RESTful de gestion de projet : cas du service d’épidémiologie de l’Institut Pasteur de Bangui

# PARTIE 1 : PRÉSENTATION GÉNÉRALE

Mémoire de fin de cycle Réalisé par BOGUIN Axel Tony

# CHAPITRE I : Présentation de la structure d’accueil

## I-1. Historique

Créé en 1961, l’Institut Pasteur de Bangui (IPB) s’est imposé comme un acteur majeur de la santé publique en Centrafrique. Son histoire est riche et ponctuée de défis et de réussites, contribuant à améliorer la vie des populations locales.

L’IPB naît dans un contexte marqué par la prévalence des maladies tropicales en Centrafrique. Le Professeur Jean-Paul Coulaud, figure pionnière de médecine tropicale, joue un rôle crucial dans sa création. Sa vision : un centre d’excellence dédié à la recherche, aux soins et à la formation dans le domaine des maladies tropicales.

## I-2. Domaines d’activités

L’Institut Pasteur de Bangui est un grand centre de recherche et de santé publique de premier plan en République Centrafricaine. Ses principaux domaines d’activités comprennent :

Recherche scientifique :

* Maladies infectieuses : VIH/SIDA, tuberculose, paludisme, maladies tropicales négligées ;
* Surveillance épidémiologique : suivi et analyse des épidémies de maladies infectieuses ;
* Développement de vaccins et traitements : recherche et développement de nouveaux vaccins et traitement pour les maladies infectieuses ;
* Santé publique et vétérinaires : recherches sur les maladies animales transmissibles à l’Homme (zoonose).

Services de santé publique :

* Diagnostique et traitement des maladies infectieuses : prestation de services de diagnostic et de traitement pour les maladies infectieuses, y compris le VIH/SIDA, la tuberculose et le paludisme ;
* Vaccination : campagnes de vaccination contre les maladies évitables par vaccination ;
* Surveillance de la résistance aux antimicrobiens : suivi de la résistance des bactéries aux antibiotiques ;
* Formation du personnel de santé : formation des professionnels de la santé sur les maladies infectieuses, la santé publique et les techniques de laboratoire ; Autres activités :
* Partenariats et collaborations : collaboration avec des organisations nationales et internationales pour renforcer les systèmes de santé publique ;
* Sensibilisation et éducation : sensibilisation du public sur les maladies infectieuses et mesures préventives.

Soutien aux politiques de santé : fourniture de données scientifiques et de conseils aux décideurs politiques pour améliorer les politiques de santé.

## I-3. Mission

L’Institut Pasteur de Bangui a pour mission :

* Contribuer à l’amélioration de la santé des populations en RCA : l’IPB mène des recherches scientifiques et fournit des services de santé publique pour lutter contre les maladies infectieuses et améliorer la santé globale ;
* Renforcer les capacités nationales en matière de recherche et de santé publique : il forme le personnel de santé, fournit des équipements et des infrastructures et collabore avec les organisations locales pour renforcer les systèmes de santé publique ;
* Promouvoir la coopération internationale : l’Institut fait partie du réseau international des Instituts Pasteur et collabore avec des partenaires internationaux pour partager les connaissances et les meilleures pratiques.

Objectifs spécifiques :

* Mener des recherches scientifiques de pointe : identifier les agents pathogènes, étudier les mécanismes de transmission de maladies et développer de nouveaux outils de diagnostic, de traitement et de prévention ;
* Fournir des services de santé publique de qualité : diagnostiquer et traiter les maladies infectieuses, vacciner les populations, surveiller les épidémies et fournir une éducation sanitaire ;
* Former les professionnels de la santé : former les médecins, les infirmières, les techniciens de laboratoire et autres professionnels de la santé aux techniques de diagnostic, de traitement et de prévention des maladies infectieuses ;
* Promouvoir la sensibilisation et l’éducation : sensibiliser le public aux maladies infectieuses, aux mesures préventives et aux services de santé disponibles.

I-4. Organigramme de l’Institut Pasteur de Bangui (Voir annexe)

# CHAPITRE II : Analyse de l’existant

## II-1. Étude de l’existant

L’institut Pasteur de Bangui est un grand centre de recherche publiquement reconnu pour son utilité dans le domaine de la recherche sur la santé. Il dispose de bons équipements informatiques permettant au personnel de travailler de manière optimale, et d’une bonne connexion Internet. En ce qui concerne l’outil utilisé pour la gestion de leurs projets, l’IPB n’en possède pas. L’Institut utilise plutôt d’autres outils afin de mener à bien ses projets. Parmi ces outils, on retrouve :

* Redcap : qui est un outil servant à faire la récolte des données sur le terrain. La plupart des projets de l’Institut Pasteur passe par une étape de récolte de données, et c’est cet outil qui est le plus utilisé dans cette situation ;
* Microsoft Access : pour stocker les données dédiées à chaque projet, l’IPB utilise des bases de données Access, ce qui lui permet d’effectuer un suivi de projet simplifié ;
* Gmail : utilisé par les membres de l’institut participant aux projets, pour la collaboration, c’est-à-dire pour l’échange de documents et de messages ;
* Microsoft Excel : pour l’analyse des données récoltées.

Il est important, après avoir examiné les outils utilisés par l’IPB pour la gestion de ses projets, de porter un regard critique sur leur efficacité pour identifier les domaines d’amélioration possibles.

## II-2. Critique de l’existant

Bien que les outils présentés plus haut soient des outils assez puissants, leur utilisation à l’Institut Pasteur de Bangui présente plusieurs limites :

* La spécialisation : ces outils ne sont pas spécialisés dans la gestion de projets. Ils sont conçus pour des tâches plus générales de collecte de données ou encore de gestion de base de données, ce qui peut limiter leurs utilisations dans le contexte spécifique de la gestion de projets ;
* Personnalisation : le manque d’outils personnalisés pour la gestion de projets signifie que les processus ne sont pas optimisés pour les besoins spécifiques de l’institut, ce qui peut entraîner des inefficacités et des difficultés d’adaptation ;
* L’Intégration : il peut y avoir des défis d’intégration avec d’autres systèmes ou logiciels utilisés par l’institut, ce qui entrave le flux de travail et la communication entre les équipes ;
* Fonctionnalités de gestion de projets : les fonctionnalités spécifiques à la gestion de projets, telles que le suivi de tâches, la planification des ressources, et la collaboration en équipe, peuvent être limitées ou absentes dans ces outils ;
* Maintenance et support : dépendre de logiciels tiers pour des fonctions critiques peut poser des problèmes de maintenance et de support, surtout si ces outils ne sont plus mis à jour ou si le support devient insuffisant.

Bien que l’utilisation de ces outils ait permis à l’Institut de gérer ses projets jusqu’à présent, le développement d’une application Web de gestion de projet dédiée pourrait offrir une solution plus adaptée, personnalisée et intégrée pour répondre aux besoins spécifiques de l’Institut en matière de gestion de projet.

## II-3. Objectifs et propositions

Après avoir analysé l’existant et ses limites, il est impératif de définir clairement les objectifs de la nouvelle solution à développer pour l’Institut Pasteur de Bangui. L’objectif principal est de concevoir une application Web de gestion de projet qui soit à la fois efficace, personnalisée et adaptée aux besoins spécifiques de l’Institut.

Objectifs spécifiques :

* Personnalisation : créer une application qui reflète les processus et le flux de travail unique de l’IPB ;
* Intégration : assurer une intégration fluide avec les autres systèmes et logiciels déjà en place
* Accessibilité : offrir une interface utilisateur intuitive et accessible pour les membres de l’Institut ;
* Sécurité : mettre en œuvre des mesures de sécurité robustes pour protéger les données sensibles à la recherche ;
* Collaboration : faciliter la collaboration entre les chercheurs et les gestionnaires de projets ;
* Evolutivité : construire une application qui peut évoluer avec les besoins de l’Institut et s’adapter aux changements futurs.

Pour atteindre ces objectifs, nous proposons le développement d’une application Web basée sur Django avec API REST. Cette solution offrira :

* Une interface personnalisée : conçue spécifiquement pour l’IPB, avec des modules adaptés à la gestion de projets de recherche ;
* Gestion des utilisateurs : des rôles et des permissions définis pour sécuriser l’accès et les opérations au sein de l’application ;
* Tableaux de bord : des vues d’ensemble et des rapports détaillés pour le suivi des projets et des ressources ;
* API REST : une interface de programmation qui permettra une intégration avec d’autres applications et facilitera l’échange des données ;
* Fonction de collecte et d’analyse des données ;
* Support et maintenance : un plan pour la maintenance continue et le support technique de l’application.

En mettant en œuvre ces propositions, l’IPB disposera d’un outil puissant et sur mesure pour la gestion de ses projets, améliorant l’efficacité et la productivité de ses activités de recherche.

## II-4. Coût du projet

L’institut Pasteur de Bangui possède une bonne infrastructure informatique permettant d’accueillir tous les projets de développement. Cependant, puisque le développement de cette solution se fait dans le cadre d’un stage, le coût du projet est donc élaboré de manière théorique, de sorte à être applicable à une société externe.

### Tableau 1- Coût théorique du projet

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Phase du projet | | Description | Heures estimées | Coût horaire  (FCFA) | Coût  (FCFA) | total |
| Conception |  | Définition des  besoins, spécifications  techniques et conception de  l’architecture | 80 | 10.000 | 800.000 |  |
| Modélisation la base  données | de de | Création de la base de données | 40 | 10.000 | 400.000 |  |
| Développement | | Programmation | 200 | 10.000 | 2.000.000 |  |
| Backend | | des  fonctionnalités  côté serveurs,  API REST |  |  |  | |
| Développement  Frontend | | Interface utilisateur,  intégration avec le backend | 150 | 10.000 | 1.500.000 | |
| Tests | | Tests unitaires, tests d’intégration et tests  d’acceptation | 100 | 10.000 | 1.000.000 | |
| Déploiement | | Configuration des serveurs,  déploiement de  l’application | 20 | 10.000 | 200.000 | |
| Maintenance | | Correction de  bugs, mises à jour | 50 (par an) | 10.000 | 500.000 (par an) | |
| Total | |  | 640 |  | 6.400.000 | |

Développement d’une application RESTful de gestion de projet : cas du service d’épidémiologie de l’Institut Pasteur de Bangui

# PARTIE 2 : ÉTUDES THÉORIQUES

Mémoire de fin de cycle Réalisé par BOGUIN Axel Tony

# Chapitre III : Étude des concepts et méthodes

De nos jours, l’utilisation du Web est devenue une méthode courante pour se connecter au monde grâce à Internet, et ce, avec moins de ressources nécessaires. Ses multiples bénéfices favorisent une communication en temps réel plus efficace dans divers secteurs tels que l’éducation, la santé, la sécurité, l’agriculture etc. Une demande nous a été faite pour développer une application de gestion de projet qui simplifierait la gestion des ressources et des données liées à l’IPB, en utilisant différentes fonctionnalités et concepts du web tels que les API REST. C’est ainsi que dans ce chapitre nous allons présenter le Web ainsi que tous les concepts qui nous ont permis d’aboutir au résultat définitif de l’application de gestion de projet.

## III-1. Présentation du Web

### III.1.1. Introduction au web

Inventé entre 1989 et 1990 par l’Informaticien Britannique Tim Bernes Lee, suivi de Robert Cailliaud, le WWW est une application d’Internet basée sur un système d’informations réparti sur un ensemble de sites et d’applications Web. L’objectif de cette innovation était de développer un système permettant aux chercheurs de communiquer plus facilement via un réseau unique. Ces sites et applications Web sont reliés entre eux par des liens hypertextes qui mènent vers d’autres adresses Web. Le Web est donc un système permettant d’accéder à un ensemble d’informations ou de documents distants, en utilisant un navigateur.

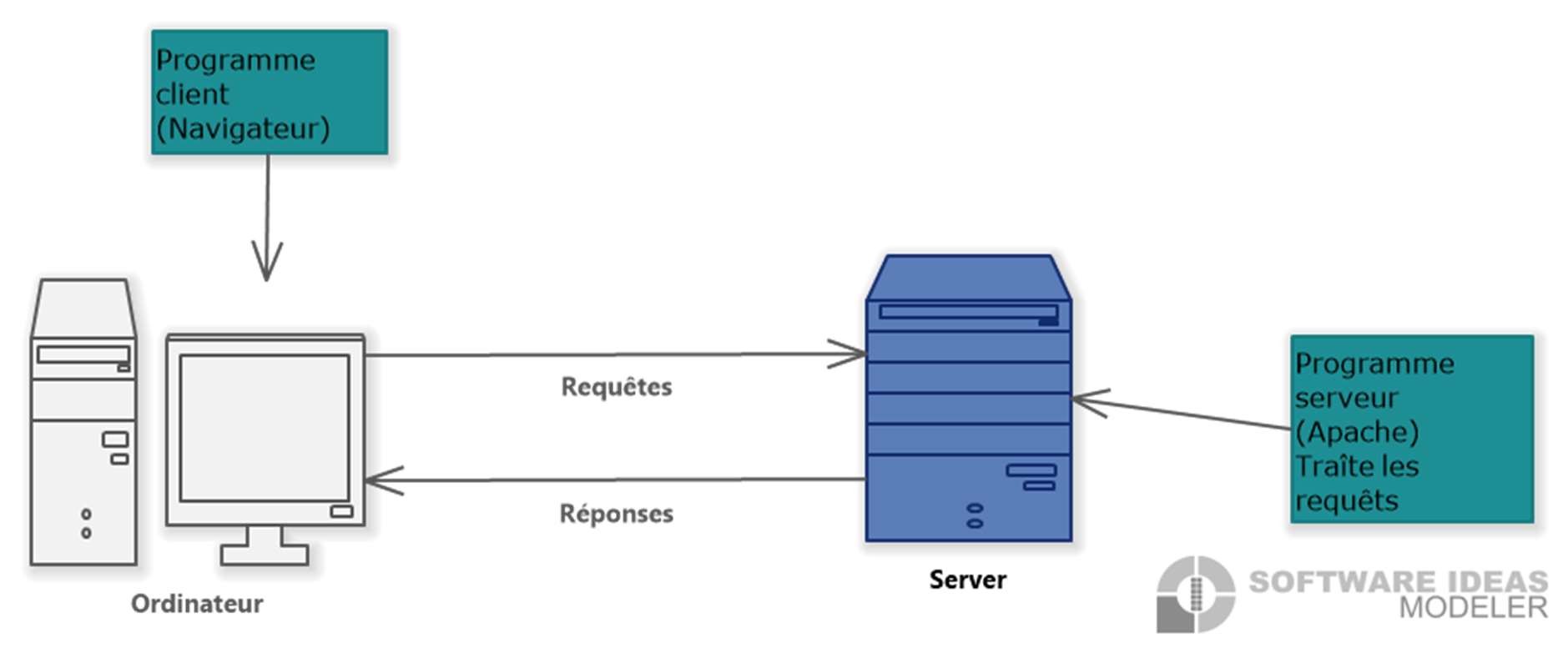
### III.1.2. Les composants et fonctionnement du web

Ce système, qu’est le Web, relie un ensemble d’informations entre elles par le réseau Internet. Il est constitué essentiellement de documents hypertextes (images, vidéos, textes, audio, etc). Chaque application Web dispose d’une quantité d’informations différentes. Ces informations sont transmises sur le réseau grâce aux fonctionnalités d’un serveur (serveur Web et autres). Cependant, pour avoir accès à ces ressources mises sur le réseau par les fonctionnalités d’un serveur Web, il faudra disposer d’une application cliente. Comme nous l’avons souligné au départ, l’utilisation du Web requiert l’Internet, qui est le réseau mondial interconnectant tous équipements informatiques distants. Principalement, le client et le serveur arrivent à communiquer grâce à une architecture appelée architecture client-serveur :

P a g e

* Le client : il peut être considéré comme tout type d’ordinateur doté d’un OS client qu’utilise une personne, ou un système. Ces ordinateurs clients utilisent des applications clients telles que les navigateurs Web pour envoyer des requêtes aux serveurs.
* Le serveur : Un serveur en informatique est un système informatique, ou un logiciel, qui fournit des services à d’autres systèmes informatiques (et leurs utilisateurs) dans le même réseau ou dans des réseaux différents. Il fait partie du modèle client-serveur, où les serveurs répondent aux demandes des clients.

Il existe donc une communication entre le client et le serveur qui est mise en place grâce à Internet :



#### Figure 1- Communication entre client et serveur

Toutefois, il faut noter que la communication entre le client et le serveur fait appel à certains concepts du web qui sont tout aussi importants les uns que les autres. On retrouve donc :

* Le HTTP : le HTTP pour Hyper Text Transfert Protocol, est un protocole de transfert de données qui est à la base de tout échange sur le Web. En effet, il est un protocole qui a pour rôle de récupérer les ressources (telles que le documents HTML) sur un serveur ou encore de les envoyer. Il est aussi désigné comme étant un protocole de type client-serveur, ce qui signifie que ses requêtes viennent d’un destinataire tel qu’un navigateur Web. Le HTTP utilise plusieurs méthodes pour établir la communication client-serveur, à savoir :
  + GET : cette méthode du HTTP est utilisée pour récupérer des données sur un serveur ;
  + POST : est une méthode utilisée pour envoyer des données au serveur de manière sécurisée ;

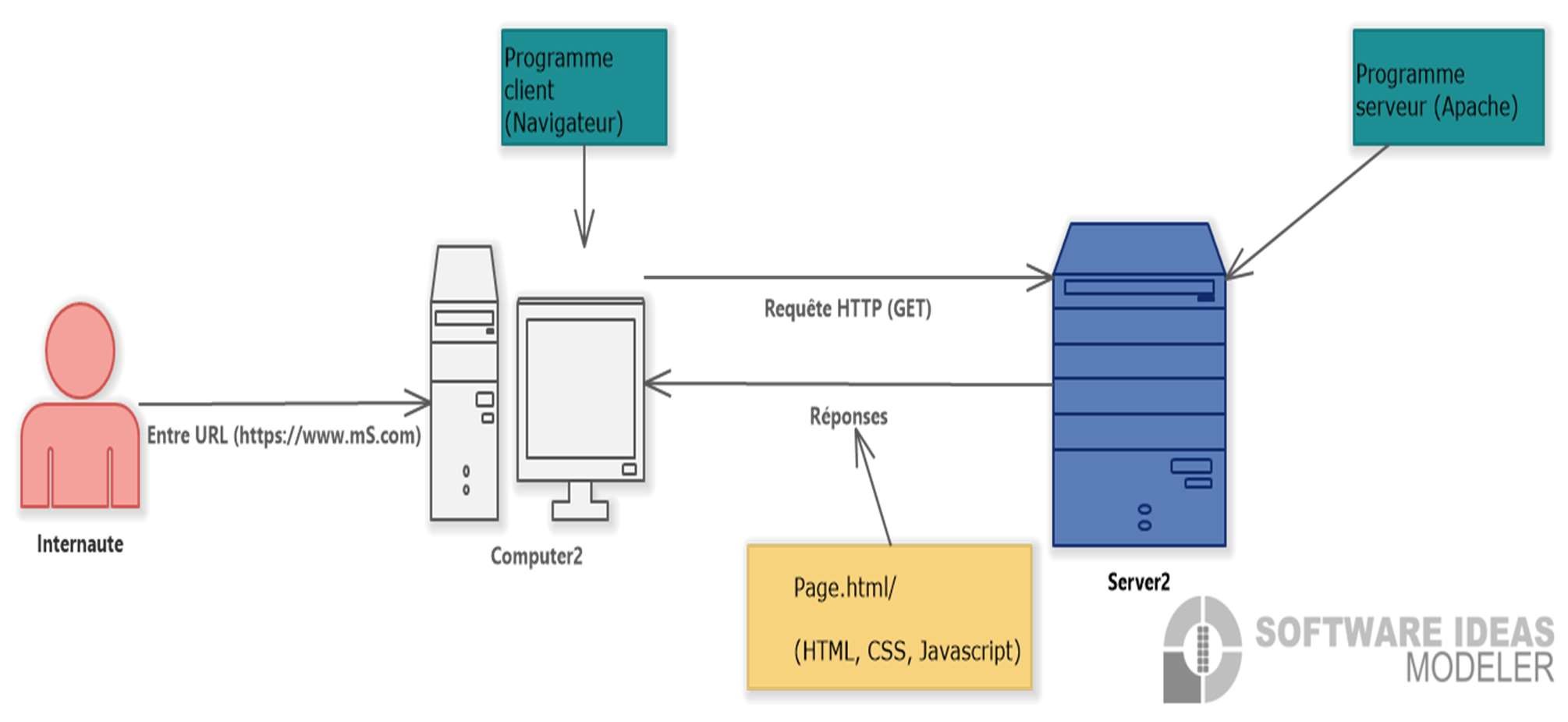
P a g e

* + PUT : elle permet de créer une nouvelle ressource ou de remplacer une autre représentation de cette ressource ciblée par le contenu de la requête ;
  + DELETE : la méthode HTTP DELETE, permet de supprimer une ressource indiquée dans la requête ;
  + Le HTTP utilise également d’autres requêtes telles que : HEAD, CONNECT, OPTION, TRACE et PATCH.

* L’URL : encore appelé adresse web, est une chaîne de caractère uniformisé pour tous les sites internet, qui permet d’identifier de manière unique une ressource du Web de par son emplacement et de préciser le protocole Internet qui permet de la récupérer. Ces ressources peuvent être de différentes natures à savoir : un fichier HTML, une vidéo, une image… etc. Puisque les URL sont uniformisées pour tous les sites web, il est important de savoir qu’elles sont décomposées en 3 parties à savoir : le protocole utilisé, le nom du domaine et l’extension du domaine. Si on devait prendre l’exemple de l’URL suivante : http://www.monSite.com/ alors cela correspondrait à :
* http://www : est le protocole de transfert utilisé, ici le HTTP ; et le sous domaine (www). Toutefois, pour le protocole, on peut utiliser le HTTPS qui est la version sécurisée du protocole HTTP ;
* monSite : cela correspond au nom du domaine. Ceci peut être décidé par l’utilisateur lorsque celui-ci possède le site Web ;
* .com : enfin, la dernière partie correspond à l’extension du nom de domaine. Les extensions qu’utilisent les sites ont des significations qu’il faut prendre en compte lors de la création d’un nom de domaine pour un site.

* Le HTML : c’est un langage de balisage qui a pour rôle d’organiser le contenu d’une page. Il est la base de tout site Internet. Lorsqu’une requête est envoyée au serveur pour demander une page, la réponse avec cette page est un fichier HTML.

Avec tous les concepts cités ci-dessus, une représentation plus avancée de la communication entre un client et un serveur ressemblerait à ceci :



#### Figure 2- Communication client et serveur avancée

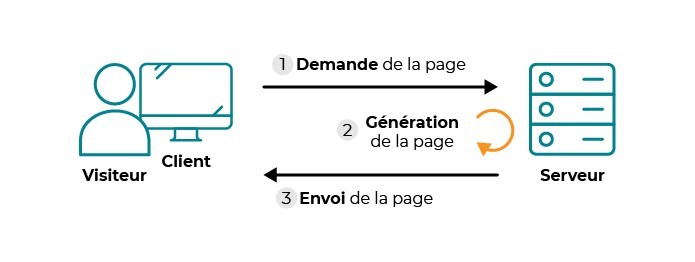
Toutefois, ceci reste encore un fonctionnement basique pour un site Web. Il peut arriver que la communication soit un peu plus complexe, étant donné que les sites Internet se répartissent en deux grandes catégories : les sites web statiques et les sites web dynamiques.

a. Les sites web statiques

Un site Web statique est un site dont le contenu ne change pas sans intervention directe du développeur dans les codes sources. Lorsqu’un utilisateur demande à recevoir la page, celle-ci est rendue comme elle a été créée, c’est-à-dire que les internautes qui demandent la page au même moment reçoivent le même contenu. À l’exemple de getbootstrap.com, qui est un site Web statique très connu dans le milieu du développement. Ces sites Web statiques, sont généralement créés en utilisant des langages tels que HTML et CSS.

Dans un site web statique, pour chaque page du site qu’on veut créer, on crée un document web statique (en HTML par exemple), correspondant selon les données stockées dans le serveur. On dit alors qu’une page est statique quand le serveur reste statique pour la création de la page Internet. Celui-ci ne fait rien pour réajuster la page, il ne fait que la rendre telle quelle.

Le fonctionnement d’une page web statique est très simple et nécessite moins de mécanismes pour les échanges d’informations entre le client et le serveur : le client fait une requête au serveur, ce dernier lui renvoie la réponse à travers une page Web statique.



#### Figure 3- Fonctionnement d'un site web statique (source : Open Classroom)

b. Les sites web dynamiques

Avec l’évolution de la NTIC, la quasi-totalité des applications hébergées sur le Web utilisent les technologies du Web dynamique.

Plus utilisés et plus nombreux de nos jours, les sites Web dynamiques désignent des sites sur lesquels certaines pages (ou voir toutes) sont dynamiques, c’est-à-dire des pages créées à la volée ou à la demande, par le serveur. Le contenu d’une page web dynamique varie selon certaines informations (localisation, fuseau horaire, données de recherche, etc), ce qui signifie qu’il s’adapte à la demande de l’internaute qui la consulte. On peut par exemple citer, parmi les sites web dynamiques, OpenClassroom pour les cours en ligne, Amazon pour les achats en ligne et Netflix pour le streaming.

Contrairement aux sites web statiques qui n’utilisent que des langages tels que HTML, CSS ou encore Javascript côté client, les sites web dynamiques utilisent des langages script côté serveur tels que Python, PHP, NodeJS (JavaScript côté serveur). Ces langages permettent d’établir une connexion avec des bases de données afin d’offrir une fonction interactive et de modifier le contenu du site. À noter aussi que lorsqu’on parle de site web dynamique, on se réfère à deux types différents en fonction du langage script utilisé :

* Script côté client : modifie un contenu en réponse aux actions d’un utilisateur ; par exemple, lorsque celui-ci clique sur un bouton. Ce type de script est chargé sur la machine client (navigateur) de l’utilisateur et non directement sur le serveur. Les scripts côté client les plus utilisés sont JavaScript et VBScript.
* Script côté serveur : ce type de script indique au serveur comment afficher et assembler la page demandée avec les informations recherchées dans la base de données. C’est grâce à ces scripts que les utilisateurs peuvent saisir des données, rechercher des informations ou encore télécharger des fichiers.

Le fonctionnement d’un site web dynamique est un peu plus complexe que celui d’un site web statique : lorsqu’un utilisateur demande à avoir une page à partir de son navigateur, il envoie une requête au serveur, alors le serveur interroge la BDD (Base De Données) afin de vérifier si les ressources sont existantes, la BDD envoie les informations, puis le serveur les compile et renvoie enfin le HTML/CSS au navigateur de l’internaute.



#### Figure 4- Fonctionnement d'un site web dynamique (source : Portfolio Marc)

Dans le cadre de notre projet, nous utilisons une application Web, ce qui peut présenter plusieurs avantages.

III.1.3. Les avantages du Web

Utiliser une application Web au lieu d’une application Desktop peut présenter plusieurs avantages, qu’ils soient en général pour tous les types d’applications, ou en particulier pour une application de gestion de projet. Parmi ces avantages, on peut retrouver :

* L’accessibilité : en effet, l’un des plus grands avantages que peuvent présenter les sites web est l’accessibilité. Les applications Web sont accessibles de n’importe où et sur n’importe quel appareil utilisant un navigateur Web ;
* La maintenance : les mises à jour et les corrections des imperfections sont plus faciles à appliquer aux applications Web, car elles sont déployées sur le serveur et sont immédiatement disponibles pour tous les utilisateurs ;
* L’intégration : il est plus facile d’intégrer d’autres applications basées sur le Web lorsqu’on utilise une application web ;
* La compatibilité : les applications web sont conçues de manière à être compatibles avec tous les systèmes d’exploitation ;
* La collaboration : les équipes pourront facilement collaborer en échangeant des fichiers. Cela permet d’éliminer le facteur de distanciation ;
* Mise en service des données : grâce aux données centralisées, toutes les équipes ainsi que les responsables peuvent facilement avoir accès aux informations des projets. De plus, les mises à jour sont également centralisées, alors les utilisateurs n’auront pas besoin d’une intervention directe de la part des administrateurs pour faire une quelconque mise à niveau.

Bien qu’il puisse s’agir de considérables avantages, il faut noter que pour développer de telles applications, il faut avoir à maîtriser certains outils avancés qui permettront de garantir de nombreuses fonctionnalités de l’application. Parmi ces outils, nous en avons un de très puissant nommé Framework.

## III-2. Les Framework

### III.2.1. Présentation

Littéralement traduit de l’anglais, un Framework signifie « cadre de travail ». C’est un ensemble d’outils qui facilitent grandement la tâche à un développeur. C’est un kit de composants logiciels structurels qui permet aux développeurs d’être plus efficaces dans la conception de leurs applications, tout en offrant une architecture et des composants logiciels prêts à l’emploi et réutilisables. Il apporte les bases communes à la majorité des programmes ou des sites Web. Cellesci étant souvent identiques, donc il serait plus simple pour un développeur de réutiliser du code déjà existant. On qualifie donc les Framework par le concept du DRY (Don’t Repeat Your Self). Dans un Framework, il existe généralement différentes parties qui sont déjà définies et prêtes à être utilisées par le développeur, telles que : les outils d’authentification, les outils pour la gestion de base de données, d’interface d’administration ou encore de gestion des utilisateurs. Les Framework fonctionnent généralement en utilisant des langages de programmation et permettent donc de développer sur tous les types de plateformes : que ce soit des applications mobiles, des logiciels de bureau ou encore des jeux vidéo.

Ce qui fait des Framework d’excellentes options pour le développement d’applications web, c’est qu’ils présentent un certain nombre d’avantages.

### III.2.2 Les avantages d’un Framework

Le Framework est un ensemble d’outils qui, en quelque sorte, oblige tous les développeurs à suivre une certaine ligne de conduite. Les Framework peuvent donc présenter les avantages suivants :

* Le gain de temps : lorsqu’on utilise un Framework, on gagne considérablement en temps, puisque les Framework évitent aux développeurs de développer une application depuis le début jusqu’à la fin. Cela permet donc aux développeurs de rester concentrés sur les spécificités fonctionnelles de l’application ;
* L’organisation : l’utilisation d’un Framework permet de garder un code clairement organisé, puisque celui-ci offre une hiérarchie de fichiers et de dossiers hautement optimisée. De plus, avec tout cela, on obtient une structure cohérente au code, ce qui en améliore sa qualité ;
* L’efficacité et l’optimisation du travail en équipe : puisque les Framework mettent à disposition des architectures bien formalisées, cela améliore le travail en équipe. En effet, cette organisation permet de répartir efficacement les tâches, étant donné que chacun possède sa zone de travail bien définie.
* La maintenance et l’évolutivité : les Framework sont développés par des groupes de développeurs et des organismes privés, ce qui fait que la maintenance et l’évolution de ces derniers sont optimales, et les mises à jour plus régulières. Grâce à cette communauté bâtie autour de ces outils, le développeur n’a plus à se soucier des compatibilités et des dernières tendances avec le web, car tout est mis à jour à partir du Framework ;
* La sécurité : l’un des plus gros avantages d’un Framework est que ce dernier est conçu pour prémunir des failles de sécurité que l’on rencontre le plus souvent sur le Web, telles que les failles XSS, les injections SQL, etc.
* La communauté : bien qu’il existe des développeurs assez expérimentés, cela n’enlève pas le fait que nous soyons tous le plus souvent confrontés à des difficultés. Les communautés organisées autour des Framework sont immenses, ce qui permet à un développeur de trouver facilement la solution à la plupart des problèmes qu’il peut rencontrer lors du développement de son application.

### III.2.3. Les différents types de Framework

Les Framework peuvent être divisés en deux grandes catégories, à savoir : les Framework Frontend et les Framework Backend.

III.2.3.1. Les Framework Frontend

Un Framework Frontend sert à définir tout ce qui représente l’interface graphique d’une application. Plus précisément, c’est un ensemble d’outils et de bibliothèques conçus pour faciliter le développement d’interfaces utilisateurs interactives et réactives pour les applications Web. Ces types de Framework permettent aux développeurs de créer des interfaces utilisateurs complexes plus rapidement et plus efficacement en fournissant une structure et des fonctionnalités prêtes à l’emploi. Ceci revêt une grande importance pour les développeurs se trouvant en RCA, par exemple, où la connexion Internet est plus chère. Cela leur permettra également de travailler hors ligne. On peut retrouver parmi les Framework Frontend les plus populaires et les plus utilisés :

React, Angular, Vue.js et Bootstrap.

III.2.3.2. Les Framework Backends

Contrairement aux Framework Frontend, les Framework Backend définissent la partie logique de l’application web. Cette partie logique est celle que les utilisateurs ne peuvent pas voir. Ces Framework offrent des fonctionnalités qui permettent de gérer les requêtes, les réponses, l’accès aux bases de données, la sécurité, l’authentification des utilisateurs, etc. Ils utilisent des langages de programmation tels que Python, PHP, Java, JavaScript côté serveur, Spring ou encore Ruby. Les Framework Backend les plus connus et les plus utilisés sont : Django, Larabel, NodeJs et Symphoni.

Les Framework, généralement, fonctionnent en suivant certains modèles. Ces modèles définissent comment le code doit être organisé et comment doit fonctionner l’application. Parmi ces modèles, principalement deux d’entre eux sont utilisés. Nous allons, dans la prochaine section, présenter les deux modèles les plus utilisés dans les Framework.

## III-3. Les modèles

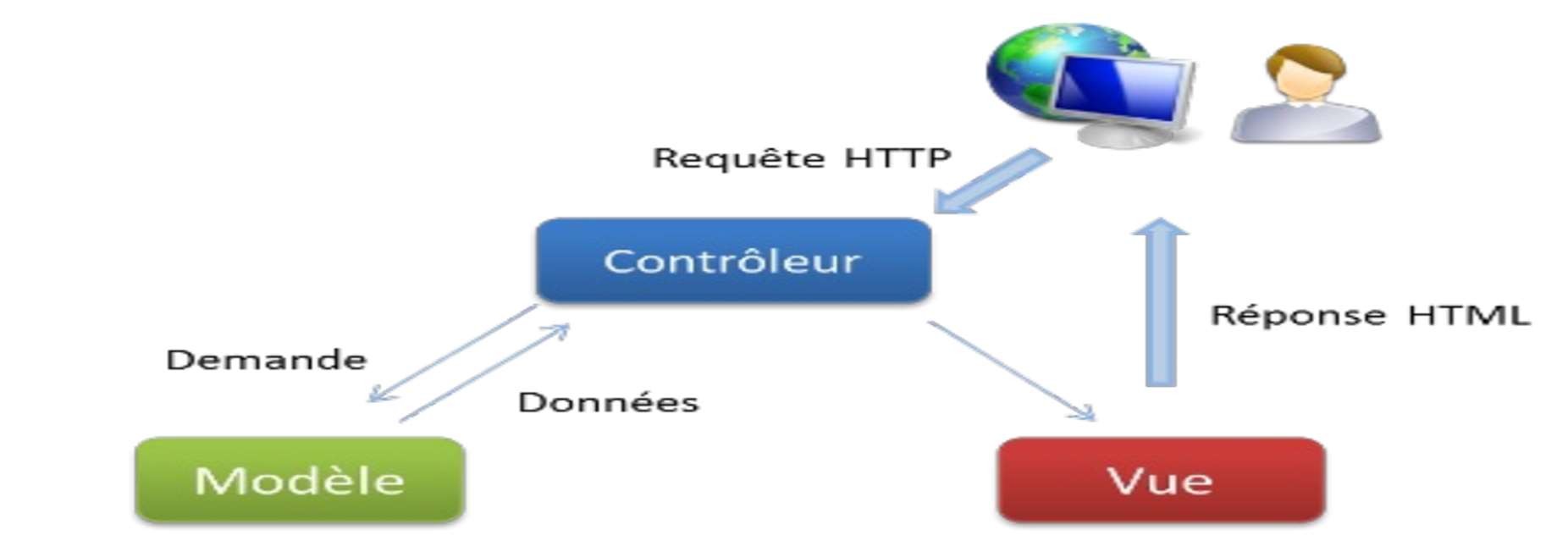
III.3.1. Le modèle MVC

Lorsque nous parlons de Framework qui fournissent une interface graphique à un utilisateur, nous parlons souvent de l’architecture MVC, « Model-View-Controller » ou « Modèle-Vue-Contrôleur » en Français. Le MVC est un pattern d’architecture logicielle très répandu. Il a été créé dans les années 1980 par Trygve Reensckaug au XEROX PARC. Le principal but de cette architecture est d’isoler les données. Ainsi, pour fonctionner, il se base sur trois entités distinctes, chacune ayant son rôle bien défini :

* Le Modèle : le modèle correspond aux données. Il représente une information enregistrée quelque part, le plus souvent dans une base de données, mais aussi dans des fichiers texte, XML, CSV, etc. Il permet d’accéder à une information, de la modifier, de vérifier l’intégrité de celle-ci, de la mettre à jour ou encore de la supprimer. Il s’agit d’une interface supplémentaire entre le code et la base de données ;
* La vue : la vue qui est, comme son nom l’indique, la visualisation des données. Elle représente l’interface utilisateur, c’est la seule chose que l’utilisateur peut voir. Non seulement elle sert à représenter une donnée, mais elle permet aussi de recueillir une éventuelle action de l’utilisateur telle qu’un clic sur un lien ou un bouton, ou encore la soumission d’un formulaire, etc. La vue correspond à un fichier HTLM, donc typiquement, un exemple de vue est une page web ni plus ni moins ;
* Le contrôleur : le contrôleur compte à lui, prend en charge tous les évènements de l’utilisateur, c’est-à-dire l’accès à une page, la soumission d’un formulaire, etc. Il se charge d’intercepter les requêtes de l’utilisateur, d’appeler le modèle puis de rediriger vers la page adéquate. Le contrôleur ne fait aucun traitement, il ne fait que l’interception et la redirection des pages.

Dans le modèle MVC, le Contrôleur reçoit les requêtes de l’utilisateur, interagit avec le Modèle pour récupérer ou mettre à jour les données, puis choisit la vue appropriée pour afficher les résultats à l’utilisateur. Lorsqu’un utilisateur interagit avec une application basée sur le modèle MVC, la requête est d’abord reçue par le Contrôleur. Le Contrôleur utilise ensuite les informations de la requête pour interagir avec le Modèle, en récupérant ou en modifiant les données nécessaires. Une fois que le Modèle a traité ces opérations, il retourne les résultats au Contrôleur. Ensuite, le Contrôleur sélectionne la Vue appropriée en fonction des résultats obtenus du Modèle. La Vue sera donc chargée de présenter ces résultats à l’utilisateur de manière appropriée, que ce soit sous forme de page Web, d’interface graphique ou autre.

Ainsi, dans le pattern MVC, le Contrôleur orchestre la communication entre le Modèle et la Vue, en s’assurant que les données sont récupérées et présentées de manière adéquate à l’utilisateur.



### Figure 5- Architecture MVC (source: Open Class room)

Les Framework basés sur le pattern MVC présentent de très grands avantages, parmi lesquels on peut citer :

* La séparation des responsabilités : le MVC permet une séparation claire des responsabilités entre le Modèle, la Vue et le Contrôleur, ce qui facilite la maintenance et l’évolutivité du code ;
* Réutilisation du code : en séparant les différentes parties de l’application, le code peut être réutilisé plus facilement, ce qui peut conduire à un développement plus efficace.

III.3.2. Le modèle MVT

Le MVT, « Model-View-Template », ou en Français « Modèle-Vue-Template » est un modèle spécial qu’utilisent certains Framework tels que Django ou Flask. Cette architecture est légèrement différente de l’architecture MVC classique. En effet, la particularité des Framework qui l’utilise, réside dans le fait qu’ils gèrent eux-mêmes la partie Contrôleur, c’est-à-dire la gestion des requêtes du client, les droits sur les actions, etc.

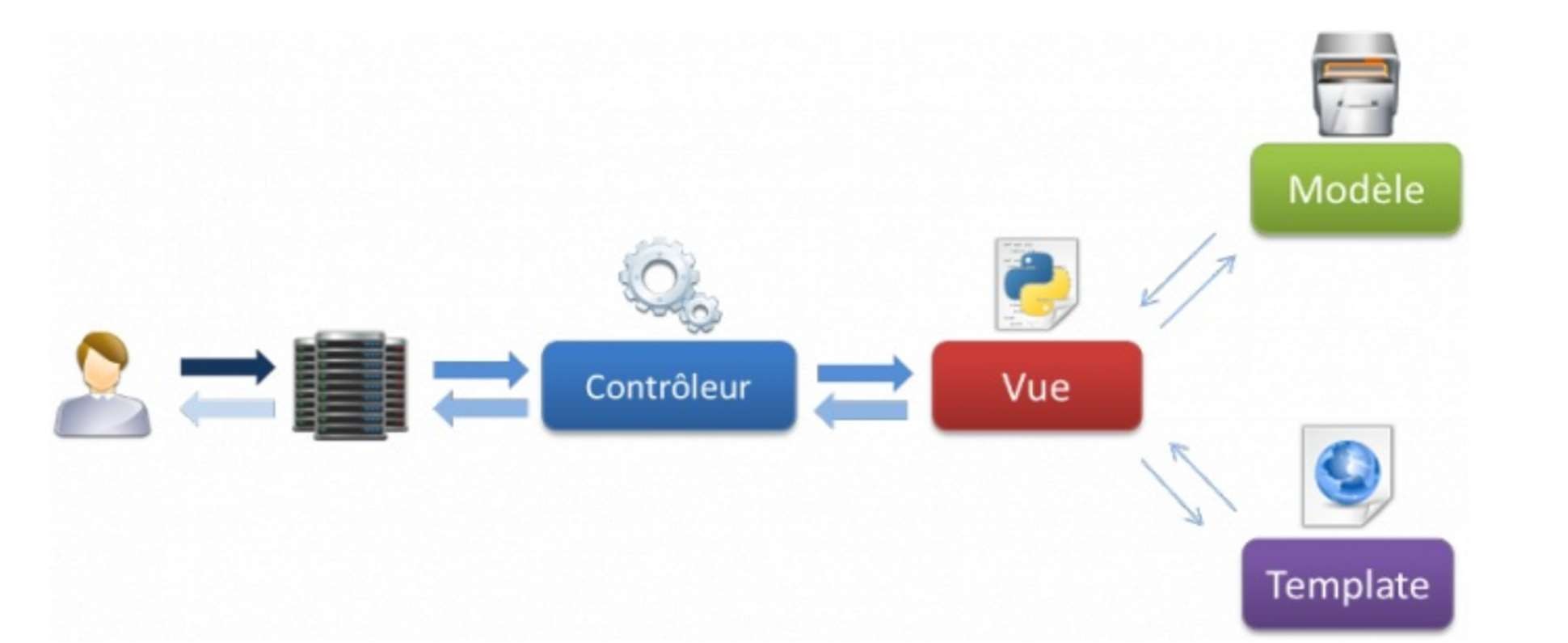
Dans cette architecture, les mêmes définitions concernant le Modèle et la Vue sont reprises ; mais toutefois, on introduit un nouveau concept, celui du « Template ». Aussi appelé gabarit en Français, le Template n’est rien d’autre qu’un fichier HTML. Ce fichier HTML sera récupéré par la Vue et envoyé au visiteur, mais avec une petite particularité : il sera analysé et exécuté par le Framework comme s’il s’agissait d’un fichier avec du code logique. Cela est rendu possible grâce au moteur de Template qu’utilise ce Framework. Ce moteur permet, dans le code HTML, d’afficher des variables ou encore d’utiliser des structures conditionnelles telles que « if/else » ou encore des boucles comme « for ».

Dans le MVT, comme dit plus haut, le Modèle gère toujours les données et la logique métier, mais la Vue agit davantage comme le Contrôleur dans le pattern MVC, en traitant les requêtes et en coordonnant l’interaction entre le Modèle et le Template.

Lorsqu’une requête est reçue par une application utilisant le MVT, elle est d’abord traitée par une Vue. La Vue interagit ensuite avec le Modèle pour récupérer ou mettre à jour les données nécessaires. Une fois que cette interaction avec le Modèle est terminée, la Vue sélectionne le Template approprié pour présenter les résultats à l’utilisateur.

Le Template, quant à lui, gère la présentation des données récupérées par la Vue, en générant la représentation finale qui sera affichée à l’utilisateur, que ce soit une page web HTML, un e-mail formaté ou tout autre type de contenu.

Ainsi, dans le pattern MVT, la Vue joue un rôle central en orchestrant l’interaction entre le Modèle et le Template, tout en conservant la séparation claire des responsabilités entre ces trois composants.



### Figure 6- Architecture MVT (source: Open class room)

Une telle architecture présente plusieurs avantages, elle reprend certains avantages que peut présenter le pattern MVC, tels que la structure claire. L’un des avantages principaux que peut présenter le MVT est :

 La productivité : en effet, grâce à sa structure prédéfinie et à ses fonctionnalités intégrées, celui-ci permet aux développeurs d’aller plus vite lors de la création des applications Web.

La conception de notre application de gestion de projet s’est en effet, basée sur ce modèle. Le MVT et le MVC définissent une architecture lisible et cohérente pour les applications qui utilisent des Framework basés sur ceux-ci. Il faut noter que certains Framework intègrent un autre outil très puissant qui leur permet d’améliorer efficacement la gestion des données du modèle. Cet outil est appelé ORM ; et nous allons l’aborder dans la prochaine section.

## III-4. L’ORM

Un ORM, ou Object Relational Mapping, est un outil de programmation qui permet de mapper des objets de type classe à des tables dans une base de données relationnelle. Plus précisément, c’est un programme qui se place en interface entre un programme applicatif et une base de données relationnelle (SGBDR) pour simuler une base de données orientée objet (SGBDOO). Le mapping objet-relationnel consiste à déclarer une association entre une (ou plusieurs) classe et une table, et chaque attribut de la classe avec un champ de la table associé. En utilisant un ORM, les développeurs peuvent interagir avec la base de données en utilisant des objets et des méthodes familiers au lieu d’écrire des requêtes SQL directement.

Dans le cas spécifique de Django, son ORM est particulièrement puissant, et cela est dû à plusieurs raisons :

* Une syntaxe expressive : l’ORM de Django, utilise une syntaxe expressive et intuitive qui permet aux développeurs d’interagir avec la base de données en utilisant du code Python clair et concis. En effet, grâce à l’ORM de Django, dans un Modèle, pour représenter une table dans la base de données, il suffit de créer une classe, et les champs de ces tables seront représentés par les attributs de la classe ;
* La portabilité : l’ORM de Django est conçu pour être indépendant du type de base de données, ce qui signifie qu’il peut être utilisé avec différents SGBD sans nécessiter de modification majeure dans la base de données. Donc on peut facilement passer d’une base de données telle que SQLite, à une base de données MySQL sans même modifier un seul bout de code ;
* La gestion automatique des schémas : Django offre une gestion automatique des schémas de base de données, ce qui signifie que les développeurs n’ont pas besoin d’écrire manuellement des Scripts SQL pour créer ou mettre à jour la structure de la base de données.

III.4.1. Les avantages des ORM

Les avantages pour des Framework qui utilisent cet outil puissant peuvent être multiples, à savoir :

* L’abstraction de la base de données : un ORM permet aux développeurs de travailler avec des objets et des méthodes familiers plutôt que de devoir écrire des requêtes SQL directement, ce qui facilite le développement et la maintenance du code ;
* La sécurité : les ORM offrent souvent des mécanismes intégrés pour prévenir les attaques par injection SQL, ce qui en renforce la sécurité des applications ;
* La syntaxe : les ORM ont généralement des syntaxes très expressives qui permettent d’écrire des requêtes SQL avec un langage de programmation, ce qui le rend plus clair et facile à comprendre.

Dans le développement d’une application de gestion de projet, nous nous servons principalement des ORM pour faciliter l’écriture des requêtes et accélérer le développement.

Après avoir exploré les subtilités des ORM et leurs capacités à structurer les modèles, il est essentiel de se pencher sur les mécanismes qui permettent aux applications de communiquer efficacement et de manière sécurisée avec d’autres applications et le monde extérieur. C’est ici que les API, et plus précisément les API REST vont nous être utile. Dans la section suivante nous verrons ce que c’est qu’une API, et comment les API REST enrichissent notre projet en offrant une interface flexible et évolutive pour l’échange des données.

III-5. Les API

III.5.1. Présentation

Une API, « Application Programming Interface » ou « Interface de Programmation d’Application » en Français, est un ensemble de règles et de protocoles qui permettent à différents logiciels de communiquer entre eux de manière efficace, ce qui facilite la création et l’intégration des applications. Les méthodes et les données qu'un développeur peut utiliser pour intégrer des fonctionnalités d'une application dans une autre sont définies par une API. En d'autres mots, une API offre la possibilité à deux programmes informatiques de communiquer et d'échanger des informations de façon organisée. Elle simplifie l'incorporation de diverses applications et services, ce qui revêt une importance capitale dans la création de logiciels plus modernes.

Il est important de noter que lorsqu’on aborde les API, on peut les considérer sous deux angles principaux : les types d’API basées sur l’accès et les types d’API basées sur la technologie.

Parmi les types d’API basées sur l’accès, on peut citer :

* Les API privées : aussi appelées API internes, ces API sont plus utilisées par les entreprises et donc ne laissent pas paraître les données à d’autres organismes. En adoptant une telle approche, on a la possibilité de garder un contrôle total sur ce dernier. C’est pour ce type d’API que nous avons opté pour notre projet ;
* Les API publiques : contrairement aux API privées, ces API sont conçues pour être ouvertes à tous les développeurs et personnes tierces voulant les exploiter. Elles donnent l’opportunité à d’autres applications de les utiliser ;
* Les API partenaires : elles sont conçues dans le but d’être partagées avec certains partenaires, en offrant toutefois des accès contrôlés aux données.

Parmi les types d’API basées sur la technologie, on peut citer :

* Les API SOAP : c’est une API qui est basée sur le protocole SOAP pour échanger les informations ;
* Les API GraphQL : elles ont la capacité de permettre aux clients de récupérer les données exactes dont ils ont besoin, par conséquent, cela peut avoir comme impact la réduction de la quantité de données transférées ;
* Les API REST : plus simples et efficaces à manier, ces API fonctionnent en suivant le principe de l’architecture REST. Leur fonctionnement est basé sur des requêtes HTTP.

Les API sont utilisées dans certains types d’applications, où elles sont très utiles. Cette utilité se fait remarquer dans différents domaines – dont la liste n’est pas exhaustive - à savoir :

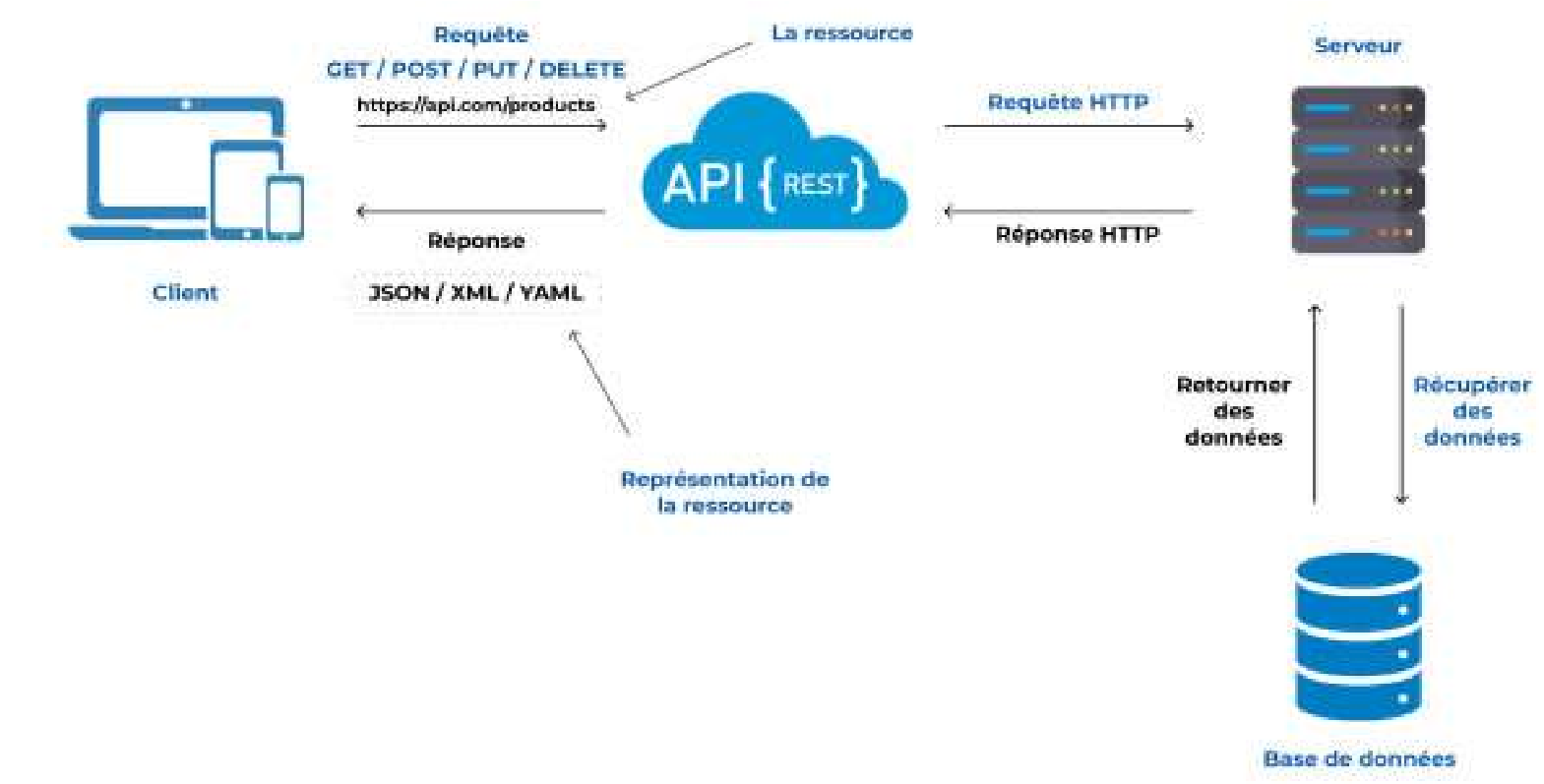
* Le commerce en ligne : dans le cas des applications qui permettent de faire du commerce en ligne, les API leur donnent l’opportunité de communiquer avec des logiciels de livraison ou encore des applications client et des systèmes de paiement (tels que Orange Money) ;
* Les réseaux sociaux : dans ce cadre-là, les API sont utilisées pour examiner un compte ;
* Les inscriptions sur certains sites : lorsqu’on veut parfois s’inscrire sur un site pour par exemple participer à un cours en ligne, ceux-ci nous demandent de choisir de nous inscrire

soit avec notre compte Google, soit avec un compte Facebook. Cette inscription à partir de Google ou Facebook est rendue possible grâce aux API qu’ils offrent.

Quand une application utilise une API pour interagir avec une autre, elle envoie une demande qui précise ses intentions, comme la récupération de données ou l’exécution d’une action spécifique. Cette demande est transmise de manière précise, généralement en utilisant des protocoles tels que HTTP ou REST, et renferme des renseignements sur les exigences de l’application.

La demande sera ensuite interprétée par l’application qui reçoit la requête via son API, qui prendra les mesures nécessaires et enverra une réponse à l’application émettrice. Les informations demandées sont incluses dans une réponse, ce qui confirme que l’action demandée a été réalisée avec succès.

Quant aux bases de données, les API sont fréquemment employées afin de permettre à une application d’accéder et de manipuler les informations qui y sont stockées. Les API jouent le rôle d’un lien entre l’application et la base de données, en autorisant l’application à envoyer des demandes pour récupérer, insérer, mettre à jour ou supprimer des informations dans celle-ci.



### Figure 7- Fonctionnement général d'une API (source : Hubvisory)

Après avoir exploré le contexte général des API et leur rôle essentiel dans la communication entre les applications, nous allons maintenant nous pencher sur un type spécifique d’API qui a révolutionné le développement Web : il s’agit de l’API REST. Pour notre projet, nous avons choisi ce type d’API pour plusieurs raisons que nous verrons.

### III.5.2. Les API REST

Autrefois, pour communiquer, les services Web utilisaient principalement le protocole SOAP, qui est un protocole de messagerie capable d’envoyer des documents au format XML en utilisant des requêtes HTTP. Ce protocole qu’utilisait l’API SOAP était plus complexe à exploiter et à manipuler. C’est en 2000 que dans sa thèse, Roy Fielding présente officiellement l’API qui va révolutionner le Web, l’API REST.

Une API REST est une architecture logicielle qui définit un ensemble de contraintes pour la création de services web. Ce type d’API respecte les contraintes imposées par l’architecture REST et, par conséquent, permet d’interagir avec les services Web RESTful. Contrairement aux autres styles d’API, les API REST utilisent les opérations standard du protocole HTTP (GET, POST, PUT, DELETE) pour effectuer les actions sur les ressources.

Pour fonctionner, les API REST se basent sur certains principes de conception.

III.5.2.1. Les principes de conception REST

Une API REST a été développée dans le but d'être intuitive, flexible et compréhensible. Elle encourage l'emploi des fonctions courantes du protocole HTTP, ce qui la rend compatible avec de nombreuses applications Web et mobiles. Cependant, les API REST reposent sur des principes spécifiques. La conception REST repose sur l'idée de considérer les ressources comme des entités distinctes et d'utiliser des opérations standardisées pour interagir avec elles. Ces principes sont au nombre de 6 :

* Architecture client-serveur : cette architecture permet la séparation entre le client et le serveur. Dans la création d’une API REST, il est essentiel que les applications client et serveur soient entièrement autonomes les unes des autres. L’unique donnée que l’application client doit avoir est l’URI de la ressource requise ; sans quoi, elle ne pourra pas interagir avec l’application serveur. De même, il est impératif que l’application serveur ne modifie pas l’application client en lui transmettant les données demandées, si ce n’est par le biais de HTTP ;
* Mise en cache : les ressources doivent toujours être mises en cache du côté client ou du côté serveur dans la mesure où cela est possible. Les réponses doivent être explicitement ou implicitement marquées comme étant cacheables ou non-cacheables. Cela permet d’améliorer les performances et l’efficacité du système ;
* Interface uniforme : il est essentiel que toutes les demandes d’API provenant d’une même ressource aient une apparence identique, peu importe l’origine de la demande. Il est essentiel pour l’API REST de s’assurer que le même élément de données, comme le nom de l’adresse électronique d’un utilisateur, ne soit attribué qu’à un unique identifiant URI. Il est important que les ressources ne soient pas trop volumineuses, mais qu’elles renferment toutes les informations nécessaires pour le client. Les ressources sont manipulées à travers une interface uniforme, qui simplifie et standardise les interactions entre le client et le serveur ;
* Stateless (Sans état) : chaque requête du client au serveur doit contenir toutes les informations nécessaires pour comprendre et traiter la requête. Le serveur ne conserve pas l’état de la session du client entre les requêtes ;
* Systèmes en couches : il faut savoir que dans une API REST, les appels effectués entre le client et le serveur passent par différentes couches. Cette architecture fait référence à la possibilité d’ajouter des intermédiaires entre le client et le serveur, sans que ni le client ni le serveur soient conscients de la présence de ces intermédiaires. Ces intermédiaires peuvent être des proxys, des passerelles et des pares-feux, ce qui a pour conséquence d’améliorer la sécurité, l’évolutivité et la gestion du système ;
* Code à la demande : celui-ci est facultatif. Généralement, les ressources envoyées par les API REST sont statiques (au format JSON par exemple) ; mais dans certains cas, ces réponses peuvent contenir du code ; et dans cette situation, le code ne devra être exécuté qu’à la demande.

C’est en suivant ainsi, le principe du REST, que les API REST fonctionnent. Outre le fait que ces API fonctionnent en suivant ces principes, leur application en est encore plus simple.

III.5.2.2. Fonctionnement des API REST

Dans le fonctionnement des API REST, ce qui est à la base du commencement du processus de fonctionnement sont les URI (Uniform Resources Identifier). En effet, dans ce type d’API, une entité symbolise un objet que l’on peut repérer, manipuler et conserver. En général, ces entités sont associées à des concepts ou des éléments liés au domaine de l’application, tels que les utilisateurs, des commandes, des produits et bien d’autres. Ces entités ne sont accessibles que par les URI. Les URI sont des chaînes de caractères utilisées pour identifier de manière unique une source particulière.

Dans une API REST, toutes les ressources sont accessibles à partir d’une URI en suivant un ensemble de conventions et de méthodes HTTP standard. La gestion de ces ressources fonctionne de la manière suivante :

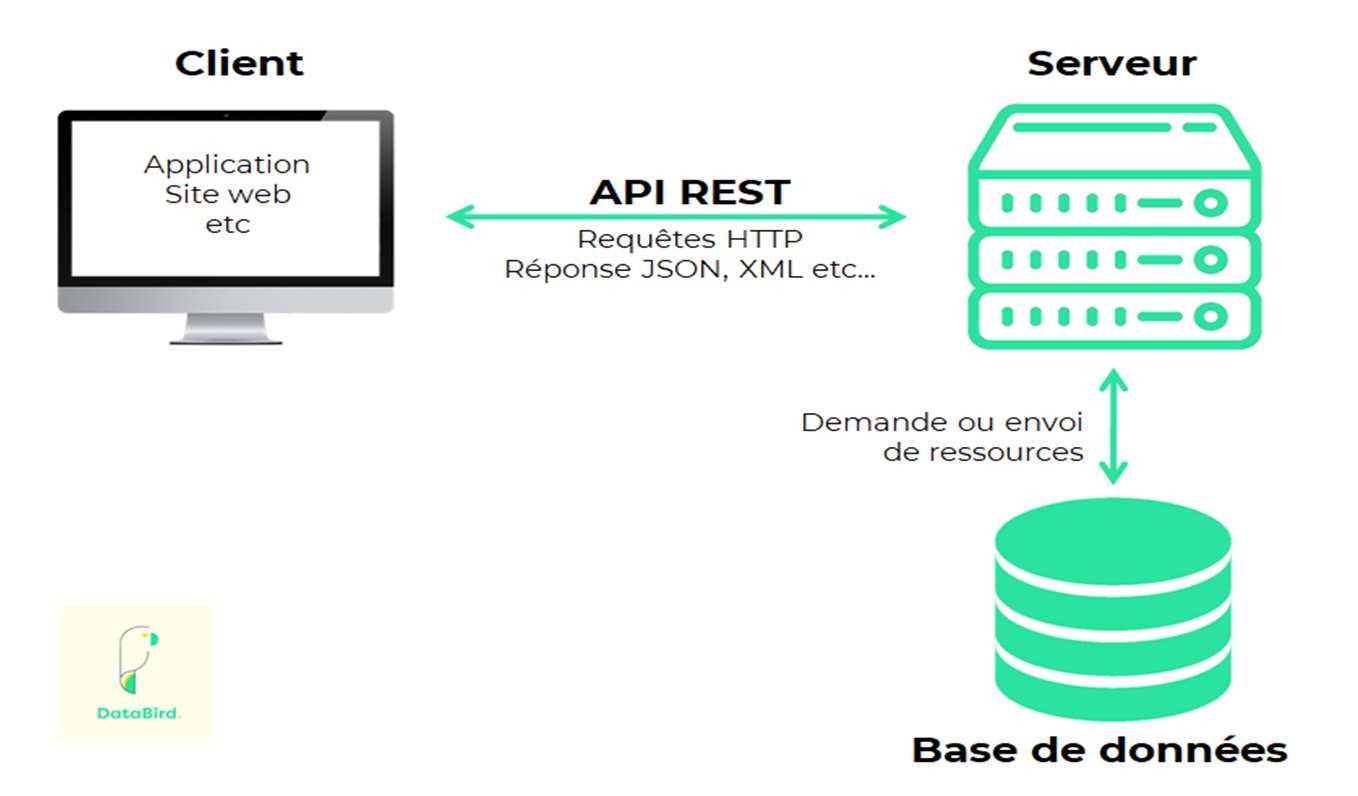
 Chaque ressource est identifiée par une URI unique qui fournit un moyen d’accéder à la ressource sur le serveur. Pour illustrer ceci, nous pouvons prendre l’exemple de l’URI

« /projets » pour accéder à l’ensemble des projets dans une application ;

 Ensuite, les requêtes HTTP vont définir une action à effectuer sur la ressource ciblée en utilisant les opérations standardisées. Par exemple, pour chaque action :

* GET : utilisé pour récupérer une ressource ou une série de ressources. À titre d’exemple, nous avons « /projet » qui permet de récupérer tous les projets, et « /projets/2 » permettra de récupérer un projet spécifique avec ses informations détaillées grâce à son ID (2) précisé ;
* POST : on crée une nouvelle ressource. Si, par exemple on envoie une requête avec cette méthode sur l’URI « /projet », avec les données nécessaires au projet, on va créer un nouveau projet ;
* PUT : remplace une ressource déjà existante dans la base de données. Lorsqu’on envoie une telle requête, par exemple à « /projet/2 » avec les nouvelles données du projet, cela remplacera les données de l’ancienne ressource se trouvant à l’identifiant 2 ;
* PATCH : permet de faire une mise à jour partielle sur la ressource. Envoyer une telle requête avec « /projet/2 » avec certaines données mises à jour modifie uniquement les champs spécifiés ;
* DELETE : enfin, la dernière opération qu’est la requête DELETE, permet de supprimer une ressource en spécifiant son identifiant, par exemple « /projet/2 », supprimera la ressource se trouvant à l’identifiant 2 ;

 Après la demande d’une ressource par un client, le serveur envoie une réponse avec une représentation de cette ressource. La réponse envoyée par le serveur se présentera généralement sous deux formats, à savoir : le format JSON et le format XML. Cette représentation qui apparaît sous l’un de ses deux formats est facilement lue et manipulée par le client.



#### Figure 8- Fonctionnement d'une API REST (source : Data Bird)

Ainsi fonctionne donc une API REST.

Les API REST sont aujourd’hui largement répandues et sont par conséquent les plus utilisées. Cette utilisation massive découle des nombreux avantages que cette API peut présenter.

#### III.5.2.3. Avantages des API REST

Les API REST sont devenues aujourd’hui, incontestablement les plus utilisées, soit un taux d’utilisation atteignant 93,4 %[[1]](#footnote-1). La plupart des entreprises et des développeurs les utilisent parce qu’elles présentent de nombreux avantages, parmi lesquels on peut citer :

* L’indépendance : utiliser une API REST a pour avantage de séparer les applications au niveau frontend et backend. C’est-à-dire que le client et le serveur sont séparés. Ceci aura pour conséquence de protéger le stockage des données puisqu’il n’y pas de traitement des règles de gestion directement côté client ;
* La flexibilité et la facilité d’utilisation : une API de qualité garantit la souplesse et la simplicité d’utilisation. Les développeurs ont la possibilité de se focaliser sur les caractéristiques de leur application plutôt que sur le fonctionnement du système. Ceci diminue les délais de création, d’intégration et de programmation ;
* La réutilisation : non seulement valable pour les API REST, mais une API en général permet la réutilisation des fonctionnalités spécifiques à un projet. Ceci évite aux développeurs la duplication du code et par conséquent, améliore l’efficacité du développement ;
* L’interopérabilité : les API REST ont l’avantage de ne dépendre d’aucun langage de programmation ou encore d’un système d’exploitation donné. De plus, une fois développées, elles peuvent être utilisées sur à peu près toutes les plateformes et applications ;
* L’évolutivité : plus flexibles que les autres types d’API, les API REST ont la capacité d’évoluer au fil du temps sans pour autant perturber les clients existants. Il est donc plus possible d’ajouter de nouvelles fonctionnalités qui n’auront aucun impact sur les applications existantes.

Les API REST sont donc les mieux adaptées pour la plupart des projets professionnels.

## III-6. UML

UML, pour « Unified Modeling Language », est un langage destiné à la modélisation des systèmes logiciels, permettant ainsi d’avoir une visualisation graphique de ces systèmes. Pour représenter chaque partie d’un système, UML utilise ce qu’on appelle des diagrammes. Un Diagramme permet d’avoir une visualisation des processus et des séquences dans un système logiciel. Il capture les éléments essentiels tels que les acteurs, les rôles, les actions et les artefacts. Pour représenter un système, ces diagrammes se subdivisent en 14 depuis la version 2.3 de UML, et sont repartis en deux (02) grandes catégories parmi lesquelles on peut citer :

* Les diagrammes de structures ou diagrammes statiques : ils représentent la partie statique d’un système. On retrouve dans cette catégorie de diagrammes :
  + Diagramme de classe : représente un système statique Orienté objet1. Il décrit les classes d’une application ainsi que leurs relations ;
  + Diagramme d’objet : il est similaire au diagramme de classe, à la seule différence qu’il représente des objets de la vie réelle. Il représente les objets ainsi que leur lien à un instant donné ;
  + Diagramme de composants : ce type de diagramme UML montre la relation structurelle entre chaque composant et sous-composant physique d'un système logiciel complexe. Il illustre la manière dont les composants sont combinés pour créer des composants plus volumineux ;
  + Diagramme de déploiement : il illustre la corrélation entre les systèmes matériels et logiciels d’un système. Il expose la configuration physique des nœuds au sein d’un système réparti ;
  + Diagramme de paquetage : les diagrammes de paquetage ont pour rôle de regrouper les classes en paquets. Les différentes dépendances et relations entre les paquets au sein d’un système sont illustrées par ces éléments ;
  + Diagramme de structure composite : il sert à représenter, sous forme de boîte blanche, des relations entre composants d’une classe. Ils permettent de visualiser la structure interne d’une classe ; décomposent le réseau d’une classe ; d’interface et de composants ;
  + Diagramme de profils : dernier diagramme à être intégré depuis UML 2.2, il fournit une représentation des concepts utilisés dans la définition des profils, tels que les packages, les stéréotypes et l’application de profil.
* Les diagrammes de comportement ou diagrammes dynamiques :
  + Diagramme de cas d’utilisation : ce diagramme représente les objets et les interactions, définit et organise les exigences fonctionnelles et précise le contexte et les exigences d’un système ;
  + Diagramme d’activité : il représente les flux d’une activité à une autre ;
  + Diagramme d’état transition : il représente les transitions entre divers objets. Il représente aussi des objets liés à un évènement dans un système interactif ;
  + Diagramme de séquence : ce diagramme sert à illustrer les cas d’utilisation avec les différents flux d’exécution d’un état à un autre d’un objet. Ils sont utilisés pour montrer le comportement global d’un automate ou le comportement d’un ensemble connexe d’automates ;
  + Diagramme de communication : ce diagramme est de très près similaire au diagramme de séquence, à la seule différence du diagramme de séquence qui met plus l’accent sur l’aspect temporel, celui-ci se concentre plus sur les relations entre les objets ;
  + Diagramme global d’interaction : utile pour visualiser une séquence d’activité, le diagramme global d’interaction sert à décomposer de multiples scénarios et activités en diagrammes d’interaction plus petits ;
  + Diagramme de temps : c’est un diagramme dédié aux contraintes temporelles prises en compte dans l’écriture de la structure d’un logiciel.

Ainsi, grâce à cette grande variété de diagrammes, toutes adaptées à un besoin spécifique, on peut modéliser facilement n’importe quel système.

## III-7. Merise

Créée en 1978 en France et plus utilisée dans les pays francophones, la Méthode d’Etude et de Réalisation des Systèmes d’informations est une méthodologie de conception de systèmes d’information. Elle est plus utilisée pour la conception des SI (Systèmes d’information) de petite et moyenne taille.

Elle est une méthodologie structurée qui s’articule principalement autour de trois niveaux d’abstraction :

* Le niveau conceptuel : décrit la première étape de la méthode Merise. Il s’agit d’une étape d’analyse qui consiste à définir les grandes lignes du SI sans se soucier des aspects techniques. Les modèles utilisés à ce niveau sont :
  + Le Modèle Conceptuel des données (MCD) : il représente les entités du domaine métier, leurs attributs et les relations entre les entités. Il permet de décrire la structure des données du SI.
  + Le Modèle Conceptuel de Traitement (MCT) : il représente les traitements qui seront effectués dans le SI, leurs entrées, leurs sorties et leurs enchaînements. Il permet de décrire les fonctionnalités du SI.
* Le niveau organisationnel : le niveau organisationnel décrit la deuxième étape de Merise. Il s’agit d’une étape de conception qui consiste à définir l’organisation du SI en tenant compte des contraintes techniques. L’objectif est de définir les modules du SI, les interfaces entre les modules et les flux de données entre eux. A ce niveau, deux modèles sont utilisés :
  + Le Modèle Organisationnel des données (MOD) : il représente l’organisation des données du SI en définissant les fichiers, les bases de données et les flux de données entre les fichiers et les bases de données.
  + Le Modèle Organisationnel de Traitement (MOT) : ce modèle représente l’organisation des traitements du SI en définissant les modules du SI ainsi que ses interfaces et les flux.
* Le niveau logique : ce niveau représente la troisième étape de la méthode Merise. Il s’agit d’une étape qui consiste à définir la logique du système d’information sans se soucier des contraintes physiques. L’objectif est de décrire les traitements du SI en détail. Il utilise principalement à ce niveau deux modèles :
* Le Modèle Logique des données (MLD) : il représente les données du SI dans une structure logique, en définissant les entités, les attributs, les relations entre entités et les contraintes d’intégrité des données.
* Le Modèle Logique de Traitements (MLT) : il représente les traitements du SI en détail, en définissant les processus, les sous processus, les tâches, les entrées, les sorties et les structures de données.

MERISE propose un ensemble de modèles et notations pour décrire les différents aspects d’un système d’information à chaque niveau d’abstraction.

# Chapitre IV : Étude comparative et choix de solutions

La conception d’une application de gestion de projet avec le Web requiert l’utilisation de plusieurs outils, qu’ils soient utilisés pour le développement, la gestion des données ou encore la modélisation de l’application elle-même. Ces outils sont nombreux et possèdent chacun des avantages et des inconvénients. Dans ce chapitre, nous allons faire une étude comparative et, par la suite, un choix, des principaux outils utilisés dans le cadre du développement d’une application de gestion de projet.

### IV.1. Études comparatives

#### IV.1.1. Étude comparative UML/MERISE

Lors du processus de développement d’une application, l’une des étapes les plus importantes est celle de la conception. Dans le domaine de la conception d’application, deux outils sont principalement utilisés : UML et MERISE.

##### Tableau comparatif d’UML/MERISE

###### Tableau 2- Etude comparative UML/MERISE

|  |  |  |
| --- | --- | --- |
| Nom | UML | MERISE |
| Objectifs | Principalement utilisé pour la modélisation et la conception de systèmes logiciels, en se concentrant sur les concepts orientés objet et les structures. | Utilisé pour la modélisation des systèmes  d’information avec une portée plus large, couvrant la modélisation des données, des processus et des aspects organisationnels. |
| Standardisation | Standardisé par le Object  Management Group (OMG) et largement accepté comme norme de l’industrie. | Bien qu’il soit largement utilisé, il n’est pas aussi universellement normalisé que l’UML. |
| Influence Géographique | Développé avec une | Développé en France et a eu |
|  | collaboration mondiale, sans lien spécifique avec une région géographique. | un impact significatif principalement dans les pays francophones. |
| Méthodologie | UML est un langage de modélisation | MERISE est une méthode, ou encore une méthodologie complète pour la modélisation et la conception. |
| Niveau d’abstraction | UML peut être appliqué à différents niveaux d’abstraction, du modèle conceptuel de haut niveau à la conception détaillée et à l’implémentation. | Merise est structuré en trois niveaux : conceptuel, organisationnel et logique, correspondant chacun à une phase spécifique du processus de développement. |

#### IV.1.2. Étude comparative Django/Flask

Concevoir une application de gestion de projet peut être une tâche assez laborieuse et complexe. Il est donc essentiel d’avoir des outils assez puissants pour le développement.

Django et Flask sont des Framework Web, tous deux basés sur le langage de programmation Python. Chacune de ces deux solutions peut avoir ses particularités.

##### Tableau comparatif Django/Flask

###### Tableau 3- Etude comparative Django/Flask

|  |  |  |
| --- | --- | --- |
| Nom | Django | Flask |
| Type de Framework | Framework full-stack, c’est-à-dire bien approprié pour les approches frontend et backend | Micro-framework léger, idéal pour les applications web simples et extensibles. |
| Complexité | Robuste et complet, avec de nombreuses fonctionnalités intégrées. | Flexible et minimaliste, nécessitant des extensions pour ajouter des fonctionnalités. |
| ORM | Utilise un ORM puissant pour | N’a pas d’ORM intégré, mais |
|  | interagir avec la base de données. | offre une grande liberté dans le choix des outils de base de données. |
| Interface  d’administration | Propose une interface d’administration prête à l’emploi. | N’a pas d’interface  d’administration intégrée, mais peut être ajoutée via des extensions. |
| Convenance | Projets complexes, applications riches en fonctionnalités, sites web avec de multiples aspects. | Petits projets, API REST, applications légères. |

#### IV.1.3. Étude comparative API REST/API SOAP

Comme nous l’avons présenté ci-dessus, les API peuvent avoir certains avantages pour les entreprises et les particuliers qui les utilisent. Les principales API utilisées dans le monde du développement sont les API REST et les API SOAP.

##### Tableau comparatif des API REST/API SOAP

###### Tableau 4- Etude comparative API REST/ API SOAP

|  |  |  |
| --- | --- | --- |
| Nom | API REST | API SOAP |
| Protocole | Utilise HTTP/HTTPS comme protocole de communication. | Utilise HTTP, SMTP, JMS ou d’autres protocoles pour la communication. |
| Format de données | Transmet les données au format JSON, XML, HTML, texte brut, etc. | Utilise exclusivement le format XML pour l’encodage des données. |
| Flexibilité | Plus flexible et moins structuré. | Plus strict et structuré avec des règles définies pour les messages. |
| Sécurité | Utilise des mécanismes de sécurité standard tels que OAuth, JWT, etc. | Propose des normes de  sécurité plus strictes, telles que WS-Security. |
| Performance | Léger et plus rapide en raison | Plus lourd en raison de |
|  | de la simplicité du format de données. | l’utilisation d’XML et des contraintes de traitement. |
| Interopérabilité | Indépendant du langage et de la plateforme. | Peut être complexe lors de l’intégration avec des systèmes non-SOAP. |
| Évolutivité | Évolutif et peut ajouter de nouvelles fonctionnalités sans perturber les clients existants. | Peut nécessiter des  modifications importantes lors de l’ajout de nouvelles fonctionnalités. |
| Documentation | Souvent bien documenté avec des exemples et des tutoriels. | La documentation peut être moins conviviale et nécessite une compréhension approfondie de SOAP. |
| Utilisation | Idéal pour les applications web, les API RESTful, les services web, etc. | Utilisé dans des domaines tels que les services web d’entreprise, les transactions financières, etc. |

### IV.2. Choix de solutions

Grâce aux études comparatives menées, nous avons pu remarquer que chacun des outils cités plus haut pouvait présenter un avantage particulier. De ce fait, nos choix se sont portés particulièrement sur ceux qui répondaient le plus à nos besoins.

#### Choix de solution de conception

Pour la conception de notre application, nous avons choisi le langage de modélisation qu’est UML ; car il est plus large et mieux adapté pour des projets aussi exigeants.

#### Choix du Framework

Parmi les Framework présentés plus haut pour les études comparatives, notre choix s’est tourné vers Django pour les nombreux avantages qu’il peut offrir en développement. En d’autres termes, ce Framework possède :

* Une bibliothèque étendue : Django possède une vaste bibliothèque de paquets. Ces paquets autonomes offrent des fonctionnalités réutilisables adaptées à des projets de grande envergure ;
* Performance : il offre un temps de chargement rapide et une bonne gestion de l’affichage de contenus, raison pour laquelle de nombreux sites à forts trafic le choisissent ;
* Sécurité : Django intègre les mécanismes de sécurité contre les attaques Web les plus courantes ;
* Flexibilité : il peut prendre en charge n’importe quel type de site Web, et peut gérer du contenu sous différents formats ;
* Communauté : la communauté autour du Framework est très vaste, ce qui fait qu’il est plus facile de trouver des solutions aux problèmes courants.

#### Choix de l’API

En ce qui concerne les API, notre choix s’est porté sur les API REST, pour leur souplesse et leur flexibilité en termes de manipulation de données. Les aspects suivants nous ont donc poussé à opter pour ce type d’API :

* La simplicité : elles utilisent les méthodes du protocole HTTP pour la communication.
* La scalabilité : peut gérer un grand nombre de requêtes simultanément.
* La performance : Les API REST peuvent utiliser le cache pour améliorer les performances en stockant les réponses aux requêtes fréquemment utilisées.
* L’interopérabilité : peuvent être utilisées avec n’importe quel langage de programmation qui supporte HTTP, ce qui les rend très flexibles.

Suite à cette étude comparative, nous avons pu mener à bien un choix d’outils approprié pour la conception et le développement de l’application de gestion de projet. Dans le prochain chapitre, nous aborderons la question de la conception de notre application.

# Chapitre V : Conception et Modélisation de l’application

Dans le processus de développement d’une application, il existe plusieurs étapes par lesquelles passe un développeur pour aboutir à une application complète. Parmi ces étapes, l’une des plus importantes et des plus cruciales est celle de la conception. C’est pendant cette étape que nous déterminons comment l’application va fonctionner et à quoi elle va ressembler. Dans ce chapitre, nous aborderons donc les processus de conception et la modélisation de notre application.

### V.1. Processus de développement

Lors de la conception d’une application, le développeur est appelé à choisir un processus de développement adapté à ses besoins. UML étant un langage de modélisation, son processus le plus adapté au développement est le processus UP. Le processus UP, pour « Unified Process » ou « Processus Unifié » en Français, est un processus de développement de logiciels itératifs orientés objet. Ce processus met principalement en œuvre quatre (04) principes, à savoir :

V.1.1. Processus itératif

L’itération consiste à renouveler une séquence d’instructions ou une partie de programme à un nombre de fois prédéterminé à l’avance ou jusqu’à ce qu’une condition spécifiée soit remplie, dans le but de prendre un traitement sur des données différentes. On dit que UP est itératif, ce qui signifie qu’il répète des cycles de développement. Chaque itération prend en compte un certain nombre de cas d’utilisation et traite en priorité les risques majeurs.

V.1.2. Processus centré sur l’architecture

Le Processus Unifié est centré sur l’architecture. Ce qui signifie qu’il met l’accent sur la conception de l’architecture du système dès le début du processus de développement.

V.1.3. Processus piloté par le cas d’utilisation

Le principal objectif d’un système informatique est de répondre aux exigences du client. Ainsi, le développement sera accessible à l’utilisateur. Les cas d’utilisation illustrent ces exigences. Les besoins fonctionnels sont identifiés et décrits par l’utilisateur, et leur combinaison forme le modèle de cas d’utilisation qui définit les fonctionnalités complètes de l’application.

V.1.4. Processus orienté par la réduction des risques

Il est important, lors du développement d’un système, qu’une équipe fasse l’analyse des risques à tous les niveaux de ce système. Il est crucial d’évaluer de manière adéquate les risques des évolutions pour faciliter la prise de décision adéquate.

Maintenant que nous avons décrit le processus que nous allons suivre tout au long de notre développement, on détaillera ensuite les acteurs participants au fonctionnement de l’application.

### V.2. Identification des acteurs

Suite à l’étude du cahier de charge élaboré pendant l’identification des besoins de l’application, nous avons pu déterminer le nombre d’acteurs importants intervenant dans le système. Nous avons donc :

V.2.1. L’Administrateur

Cet acteur a le pouvoir absolu sur le système. Il est capable d’effectuer toutes les actions dans le système. Ainsi, parmi les interactions que l’administrateur peut avoir avec le système, nous pouvons citer :

* S’authentifier ;
* Gérer les projets :
  + Ajouter un projet
  + Modifier un projet  Supprimer un projet
* Gérer les utilisateurs :
  + Ajouter un utilisateur
  + Modifier un utilisateur  Supprimer un utilisateur
* Gérer données des projets :
  + Modifier une donnée  Supprimer une donnée
* Recevoir notifications.

V.2.2. Le responsable des projets

Il est, après l’administrateur, l’acteur le plus important interagissant avec le système. Il supervise et gère tous les projets, il est chargé de contrôler également leur progression. Cet acteur interagit avec le système comme suit :

* S’authentifier ;  Gérer projets :
  + Ajouter / Modifier / Supprimer / Suspendre / Relancer projet
  + Gérer équipe : Monter / Modifier / Supprimer équipe  Consulter données projets ;
* Recevoir notifications.

V.2.3. Le chef de projet

Contrairement au responsable des projets qui a le contrôle sur tous les projets, le chef de projet lui ne peut en diriger qu’un seul. Il a la responsabilité de conduire toutes les équipes de son projet jusqu’au complet aboutissement. Ainsi, il peut :

* S’authentifier ;
* Gérer les équipes d’un projet : Monter équipe / Modifier équipe / Supprimer une équipe ;
* Gérer les tâches : attribuer une tâche / Retirer une tâche ;
* Consulter les données du projet ;
* Recevoir notifications.

V.2.4. Le chef d’équipe

Dans chaque équipe de projet, on retrouve un chef. Le chef d’équipe est chargé de conduire efficacement son équipe jusqu’à la fin du projet. Il peut donc :

* S’authentifier ;
* Gérer les tâches : attribuer une tâche / Retirer une tâche ;
* Recevoir notifications.

V.2.5. Le membre (ou Data manager)

Il représente l’acteur qui a les interactions les plus limitées avec le système. Le membre peut :

* S’authentifier ;
* Ajouter données ;
* Consulter informations :
  + Consulter informations sur les données ;
  + Consulter informations sur le projet (nom et description).
* Consulter tâches ;
* Recevoir notifications.

V.2.6. Le Directeur Général

Il est l’acteur qui n’interagit pas avec le système en écriture. Son but est de voir l’évolution de tous les projets de l’IPB et d’être authentifié des évènements les plus importants. Le DG peut donc :

* S’authentifier ;
* Consulter les données des projets (en lecture seule) ;
* Accéder au tableau de bord ;
* Recevoir notifications.

### V-3. Conception de l’application

Après avoir repéré les différents acteurs de notre système plus haut, nous allons maintenant nous centrer sur l’aspect conceptuel de l’application. La conception est une étape par laquelle passe tout projet en informatique, et principalement dans le domaine du développement d’applications. Pour notre application, puisque le choix d’outil de modélisation s’est tourné vers UML en suivant le Processus UP, certains diagrammes interviendront pour montrer le fonctionnement du système.

V.3.1. Diagrammes de cas d’utilisation

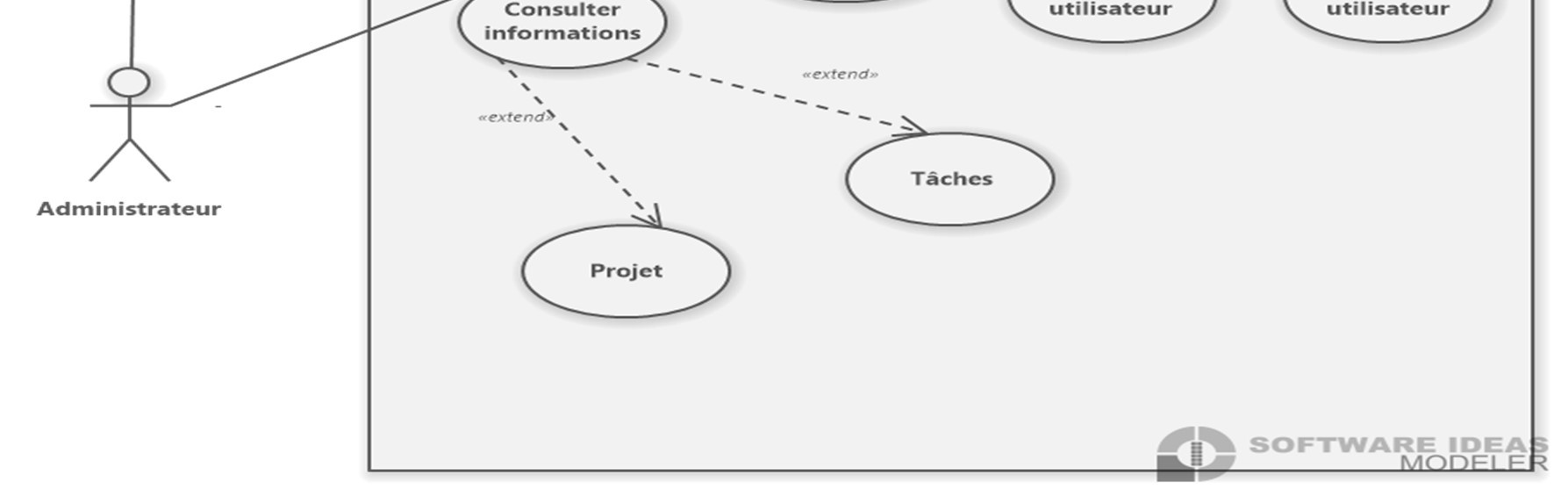
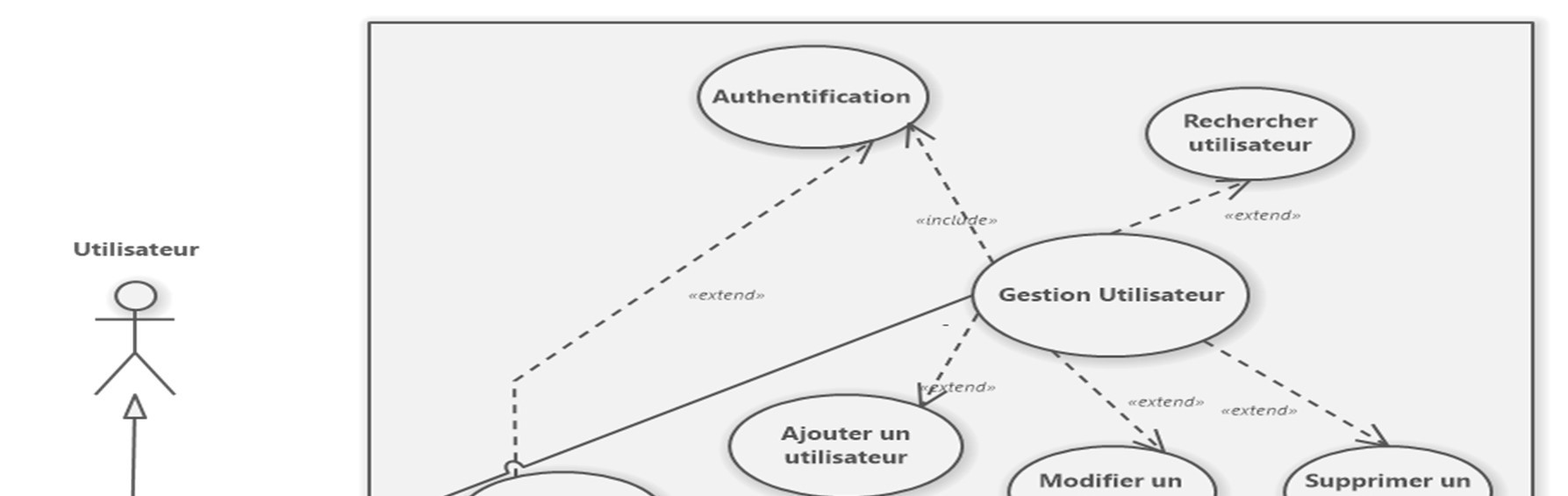
Il est utilisé pour représenter les différentes interactions entre le système et les utilisateurs. Dans notre cas, pour des questions de clarté, nous allons aborder chaque cas d’utilisation différemment selon chaque acteur. Les cas d’utilisation les plus importants seront détaillés dans un tableau avant d’être modélisés.

V.3.1.1. Diagramme de cas d’utilisation 1 : Administrateur

Décrit la gestion des utilisateurs par l’administrateur.

#### Tableau 5- Description détaillée du cas d'utilisation : gérer utilisateurs

|  |  |
| --- | --- |
| Nom du cas | Gérer utilisateurs |
| Acteur principal | Administrateur |
| Objectif | Assurer la gestion efficace des utilisateurs |
| Précondition | L’administrateur est connecté à l’application |
| Déclencheur | L’administrateur sélectionne l’option de gestion des utilisateurs |
| Scénario principal | * Il sélectionne l’option de gestion des utilisateurs * L’application affiche la liste des utilisateurs en proposant les options pour ajouter, modifier et supprimer un utilisateur * Il sélectionne une option pour la gestion (ajouter, modifier ou supprimer un utilisateur) * L’application enregistre les modifications |
| Post condition | Les modifications des utilisateurs sont enregistrées et mises à jour dans l’application |
| Exception | * Si l’administrateur tente de modifier un utilisateur qui n’existe pas, l’application affiche un message d’erreur * Si une erreur se produit lors de l’enregistrement des modifications, l’application affiche un message d’erreur. |



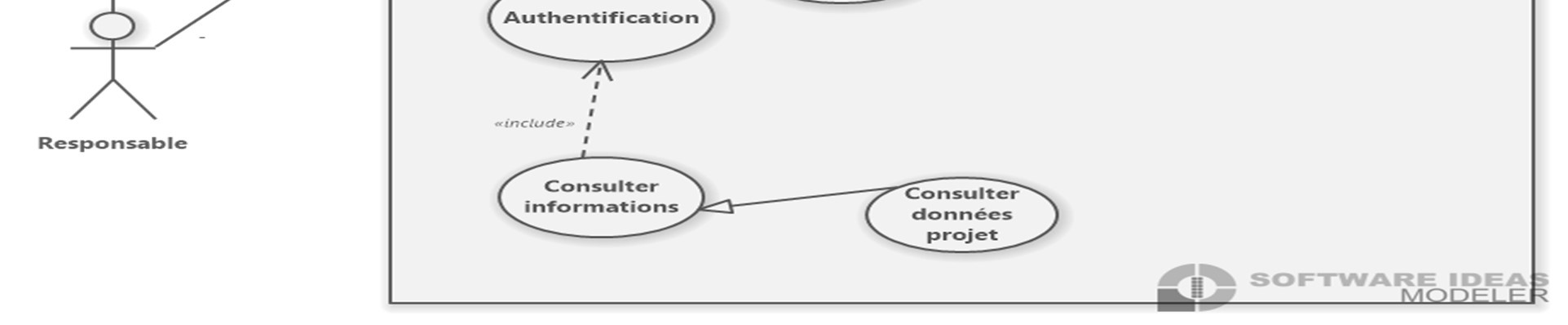
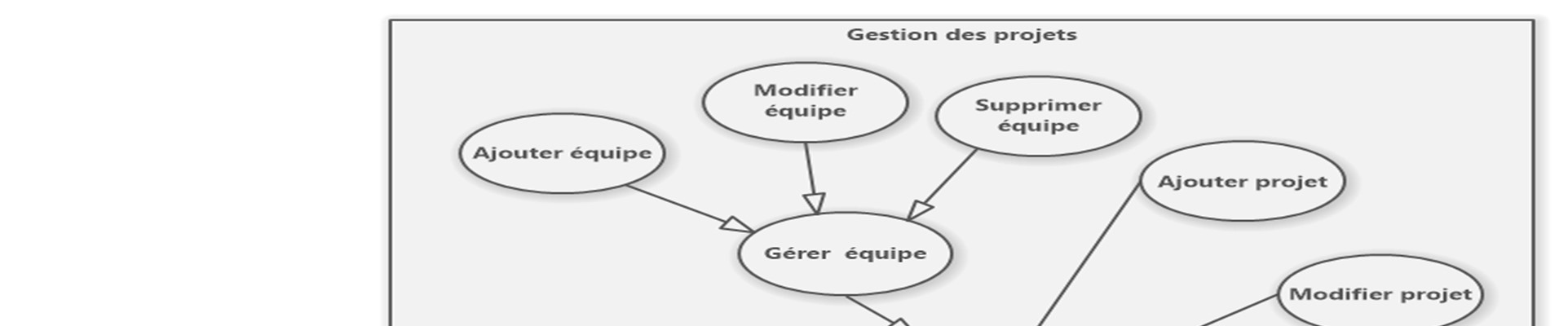
#### Figure 9- Diagramme de cas d'utilisation : Gérer utilisateurs

V.3.1.2. Cas d’utilisation 2 : Responsable des projets

Décrit la gestion des projets par un responsable chargés des projets.

#### Tableau 6- Description détaillée de cas d'utilisation : Gérer projet

|  |  |
| --- | --- |
| Nom du cas | Gérer projets |
| Acteur principal | Responsable (et administrateur) |
| Objectif | Assurer le contrôle et la gestion efficace et efficiente des projets |
| Précondition | Le responsable de projet est authentifié et connecté à l’application |
| Déclencheur | Le responsable sélectionne l’option de gestion des projets |
| Scénario principal | * L’application affiche une liste de tous les projets créés et propose des options pour la gestion (Ajouter, Modifier,   Supprimer, Lancer, Suspendre, Gérer une équipe,  Relancer un projet, assigner des tâches)   * Le responsable sélectionne une option de gestion * Une fois les modifications apportées, l’application enregistre |
| Post condition | Les modifications du projet sont enregistrées et visibles par tous les membres de l’équipe du projet |
| Exception |  Si une erreur se produit lors de l’enregistrement des modifications, l’application affiche un message d’erreur. |



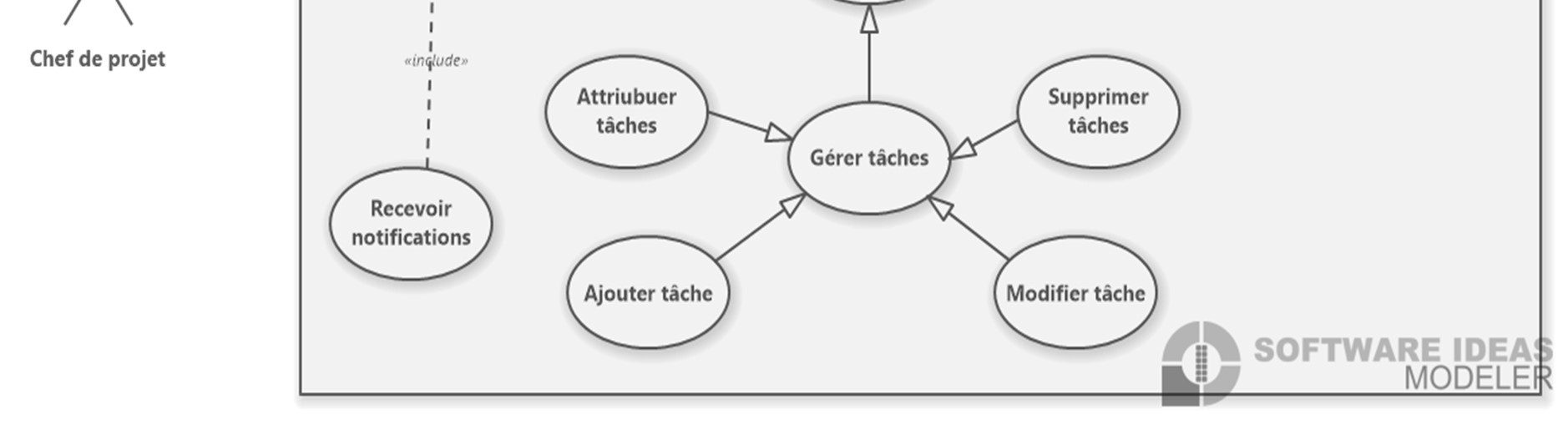
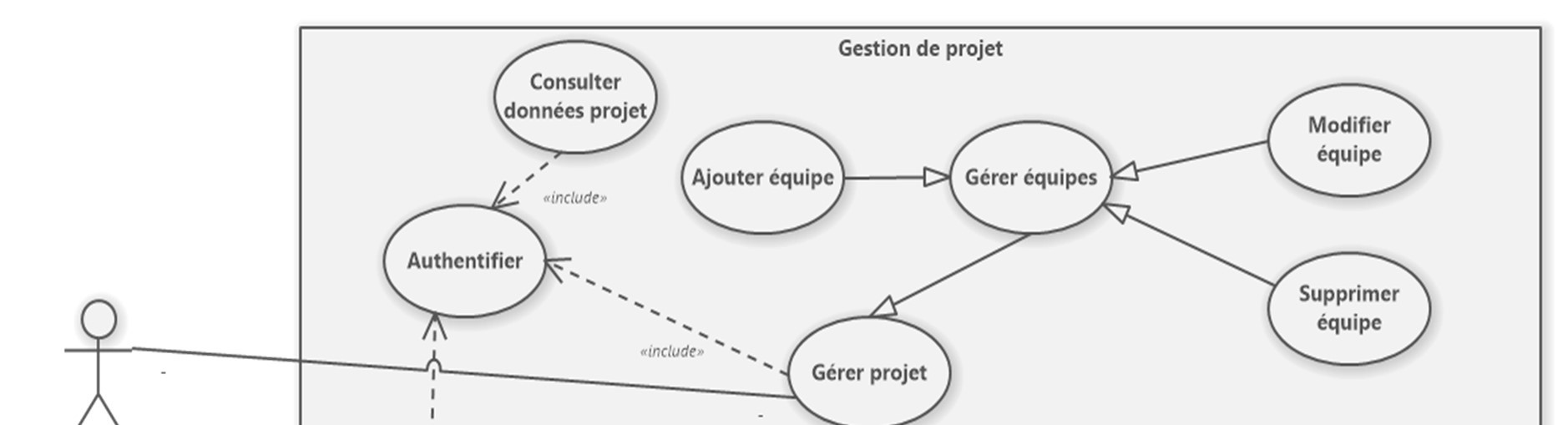
#### Figure 10- Diagramme de cas d'utilisation : Gérer projet

V.3.1.3. Cas d’utilisation 3 : Chef de projet

Décrit la gestion de projet par un chef de projet (un seul projet bien défini).

#### Tableau 7- Description détaillée du cas d'utilisation : Gérer équipe

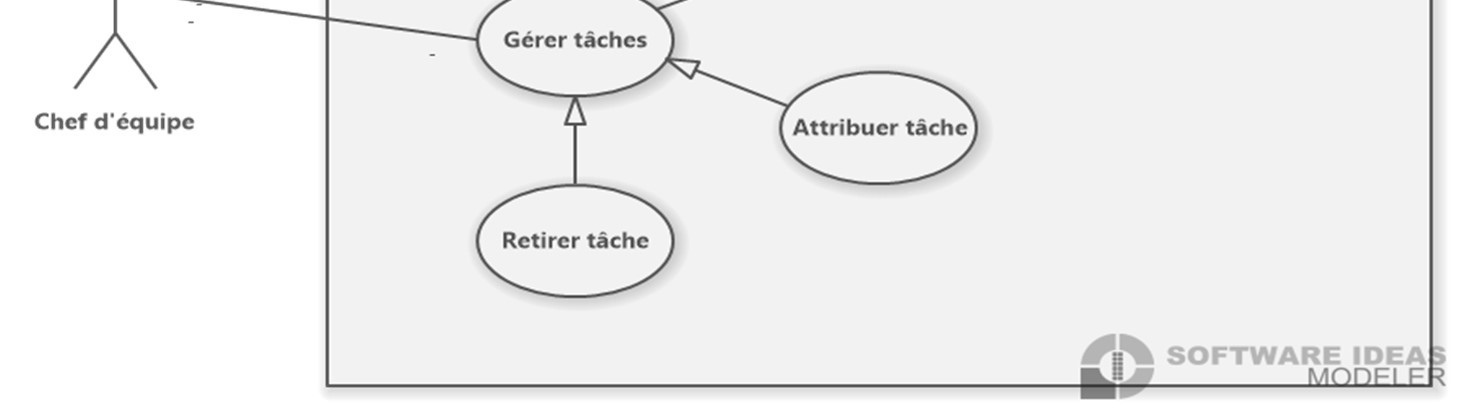
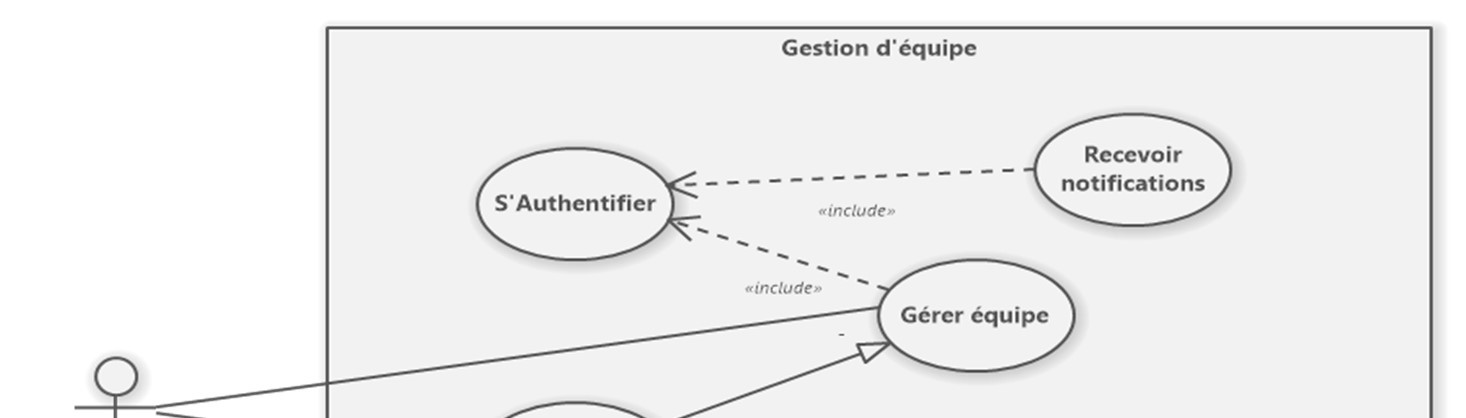
|  |  |
| --- | --- |
| Nom du cas | Gérer équipes |
| Acteur principal | Chef de projet |
| Objectif | Assurer une bonne de toutes les équipes du projet |
| Précondition | Le chef de projet est connecté à l’application |
| Déclencheur | Il sélectionne une équipe à gérer |
| Scénario principal | * Le chef de projet sélectionne l’option de gestion des équipes * L’application affiche une liste de toutes les équipes |
|  | qui participent au projet   * Le chef de projet apporte les modifications (ou ajoute / supprime) à l’équipe souhaitée * L’application enregistre les modifications apportées |
| Post condition | Les modifications apportées dans une équipe ou l’ajout d’une équipe sont enregistrés et visibles pour le responsable et l’administrateur |
| Exception |  Si le chef de projet tente d’apporter une modification à une équipe alors que le projet a été suspendu, une erreur se déclenchera. |



#### Figure 11- Diagramme de cas d'utilisation : Gérer projet (Chef de projet)

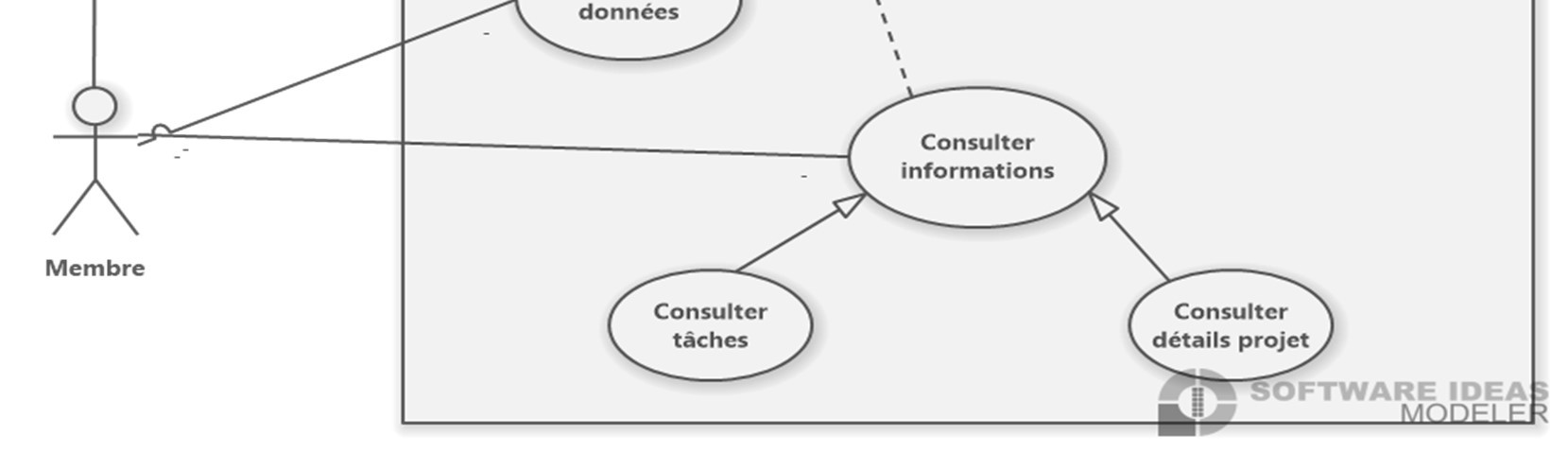
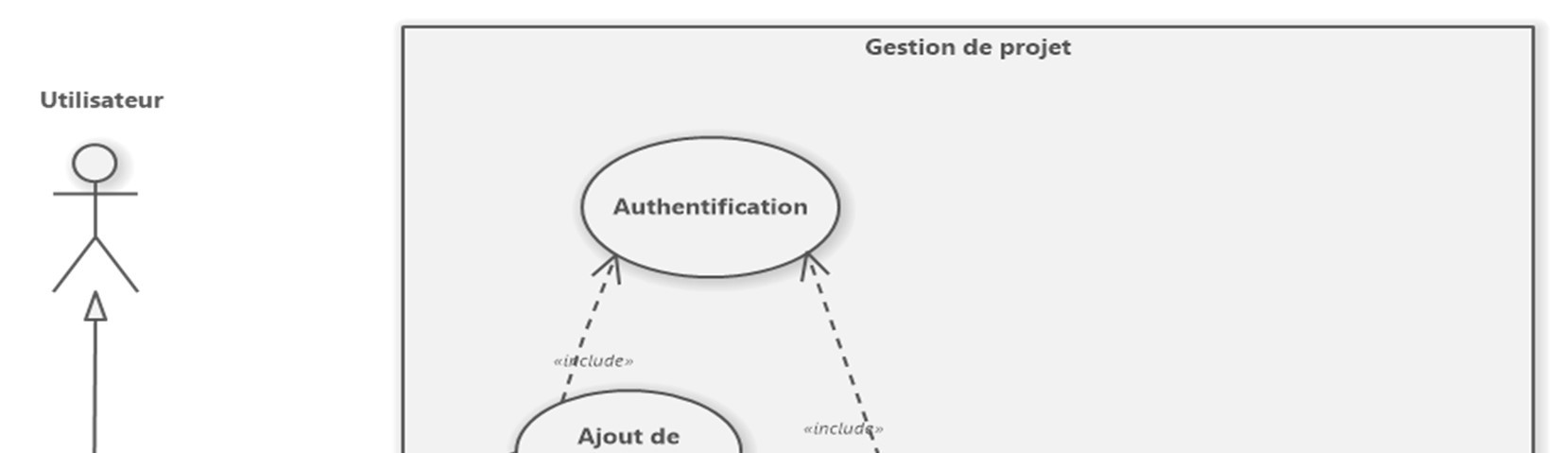
V.3.1.4. Cas d’utilisation 4 : Gérer équipe (Le chef d’équipe)

Décrit la gestion d’une équipe par un chef d’équipe désigné par le chef de projet.



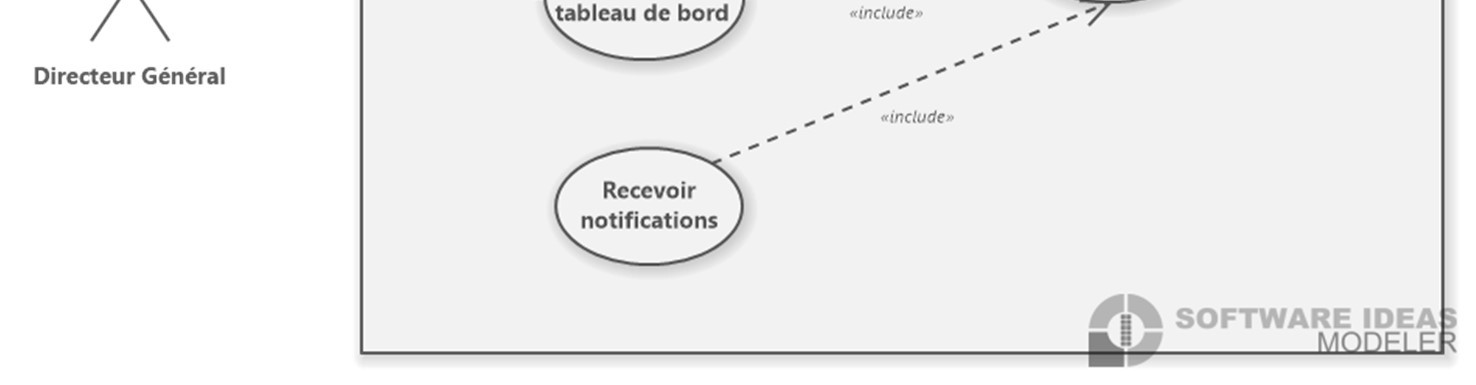
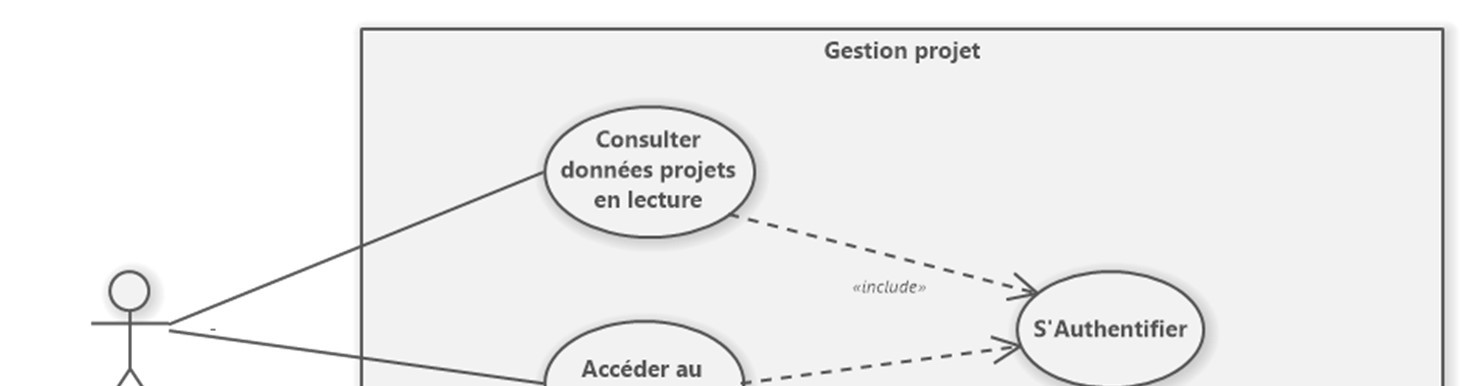
#### Figure 12- Cas d'utilisation Gérer équipe du chef de d’équipe

V.3.1.5. Cas d’utilisation 5 : Data manager



#### Figure 13- Cas d'utilisation consulter informations des membres

V.3.1.6. Cas d’utilisation 6 : Le Directeur Général



#### Figure 14- Cas d'utilisation Consulter les informations par informations par le DG

V.3.1.7. Quelques descriptions détaillées de cas d’utilisation globaux

Cas d’utilisation « Recevoir notifications »

#### Tableau 8- Description détaillée du cas d'utilisation : Recevoir notifications

|  |  |  |
| --- | --- | --- |
| Nom du cas | Recevoir notification | |
| Acteur principal | Tous les acteurs | |
| Objectif | Informer chaque acteur des évènements importants | |
| Précondition | L’utilisateur est connecté à l’application | |
| Déclencheur | Un évènement spécifique se produit dans l’application (un nouvel utilisateur ajouté, une tâche terminée...) | |
| Scénario principal | * Un évènement spécifique se produit dans l’application * L’application génère une notification correspondante et l’envoi aux utilisateurs concernés par l’alerte * L’utilisateur reçoit la notification * L’utilisateur clique sur la notification pour voir les détails | |
| Post condition | L’utilisateur est informé de l’évènement et peut prendre des mesures appropriées si nécessaire | |
| Exception |  Si l’administrateur a désactivé les notifications pour | |
|  |  | certains types d’évènements, ces notifications ne sont pas générées |
|  |  | Si une erreur se produit lors de la génération de la notification, l’application affiche un message d’erreur. |

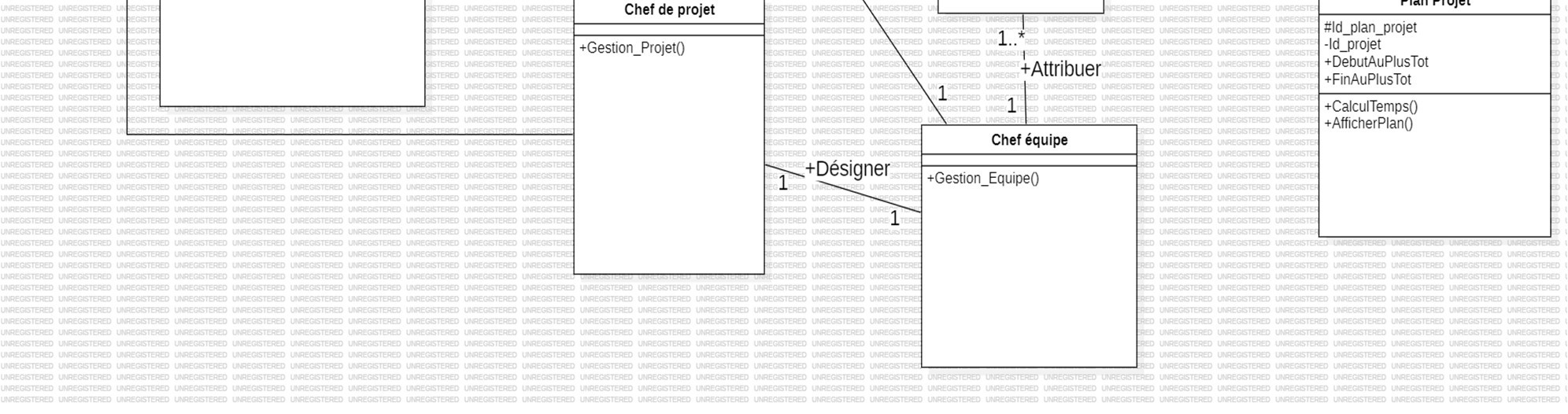
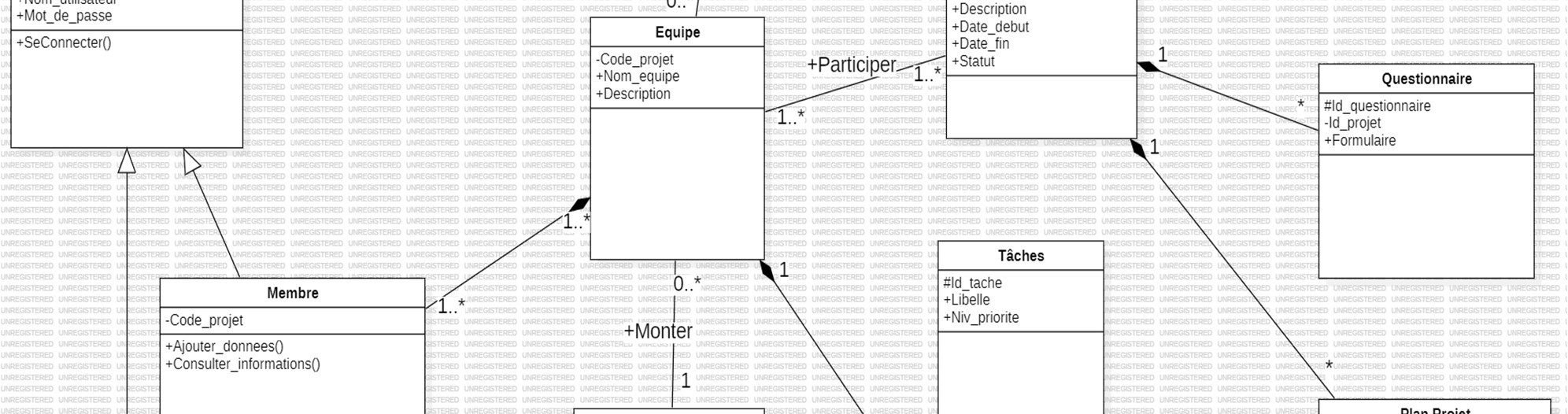
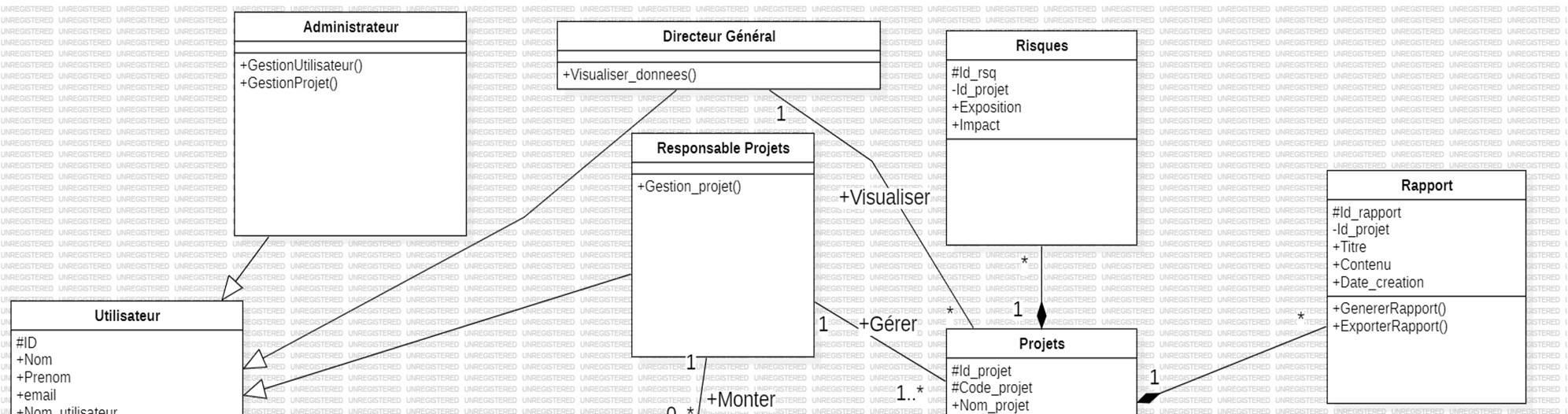
Cas d’utilisation « S’authentifier »

#### Tableau 9- Description détaillée du cas d'utilisation : S'authentifier

|  |  |
| --- | --- |
| Nom du cas | S’authentifier |
| Acteur principal | Tous les Utilisateurs |
| Objectif | Permettre à l’utilisateur d’accéder à son compte |
| Précondition | L’utilisateur a un compte valide sur l’application |
| Déclencheur | L’utilisateur entre le lien pour accéder à l’application |
| Scénario principal | * L’utilisateur entre le lien pour accéder à l’application * L’application affiche le formulaire de connexion * L’utilisateur entre ses identifiants, c’està-dire Login et Mot de passe (code de projet dans le cas des utilisateurs qui ne peuvent pas accéder à la page d’administration * L’application vérifie les identifiants * SI les identifiants sont corrects, l’utilisateur est connecté à son compte |
| Post condition | L’utilisateur est connecté à son compte et peut avoir accès aux fonctionnalités de l’application |
| Exception | * Si l’utilisateur entre des identifiants incorrects, l’application affiche un message d’erreur * Si une erreur se produit lors de la connexion, l’application affiche un |
|  | message d’erreur. |

V.3.2. Diagramme de classe

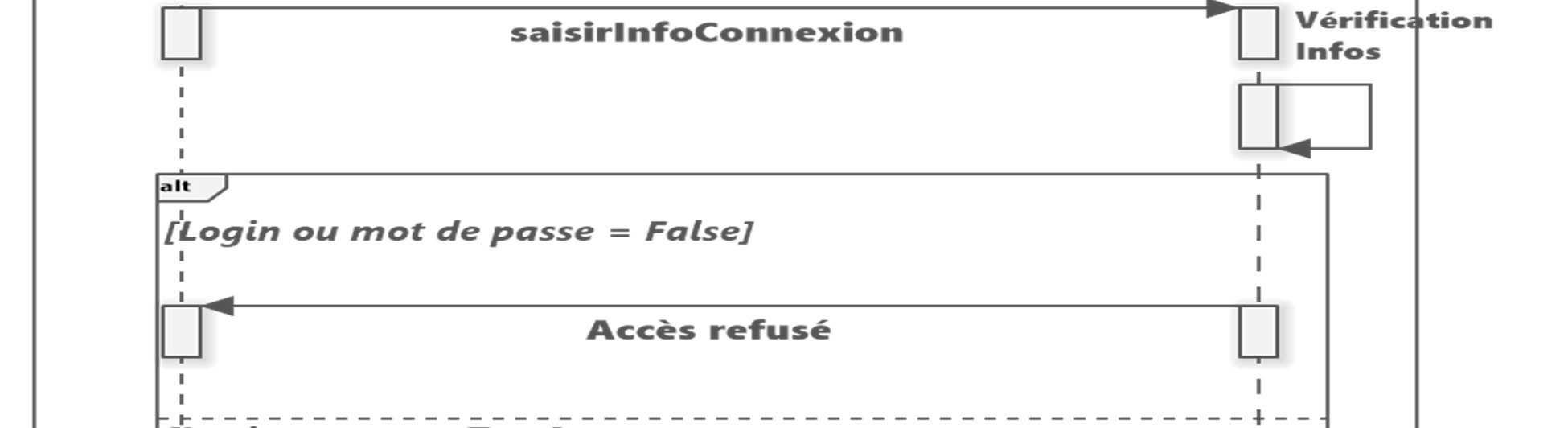
Comme défini plus haut, le diagramme de classe UML est une représentation graphique des classes d’un système logiciel.



#### Figure 15- Diagramme de classe

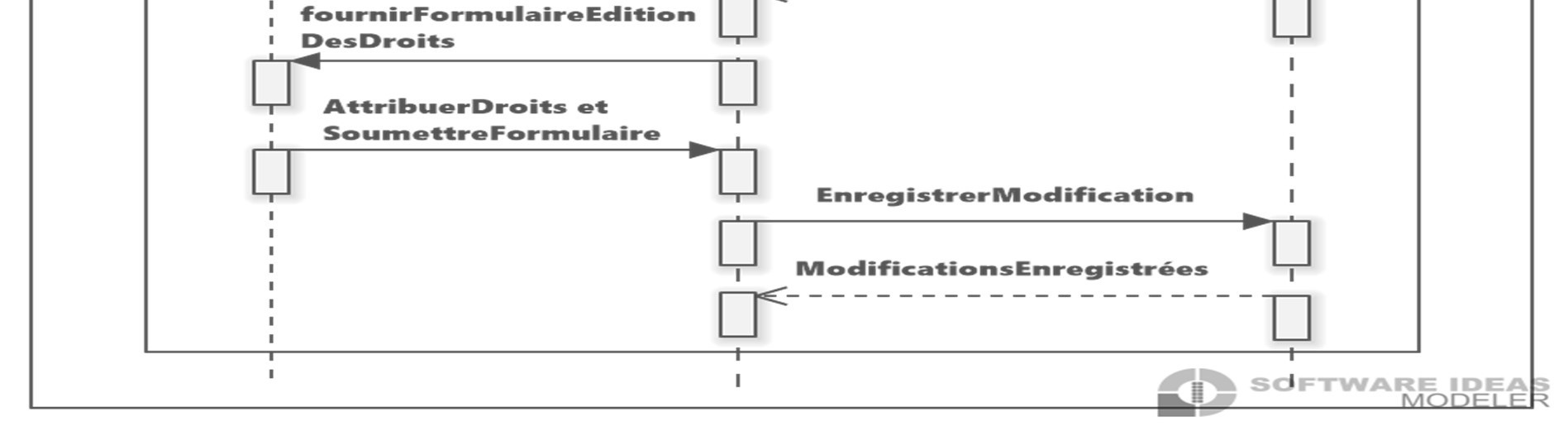
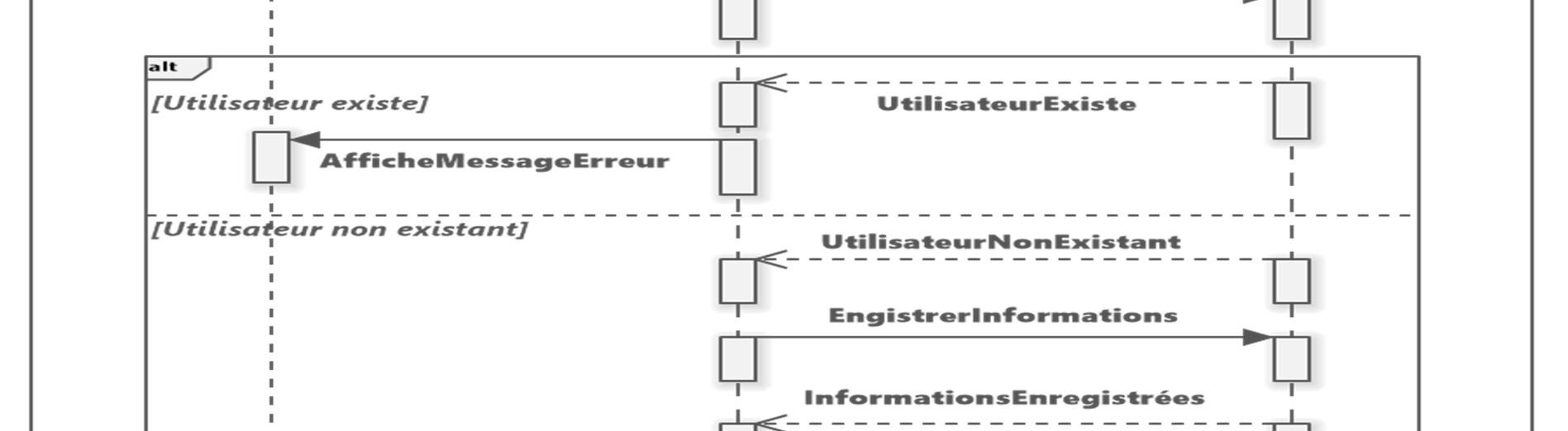
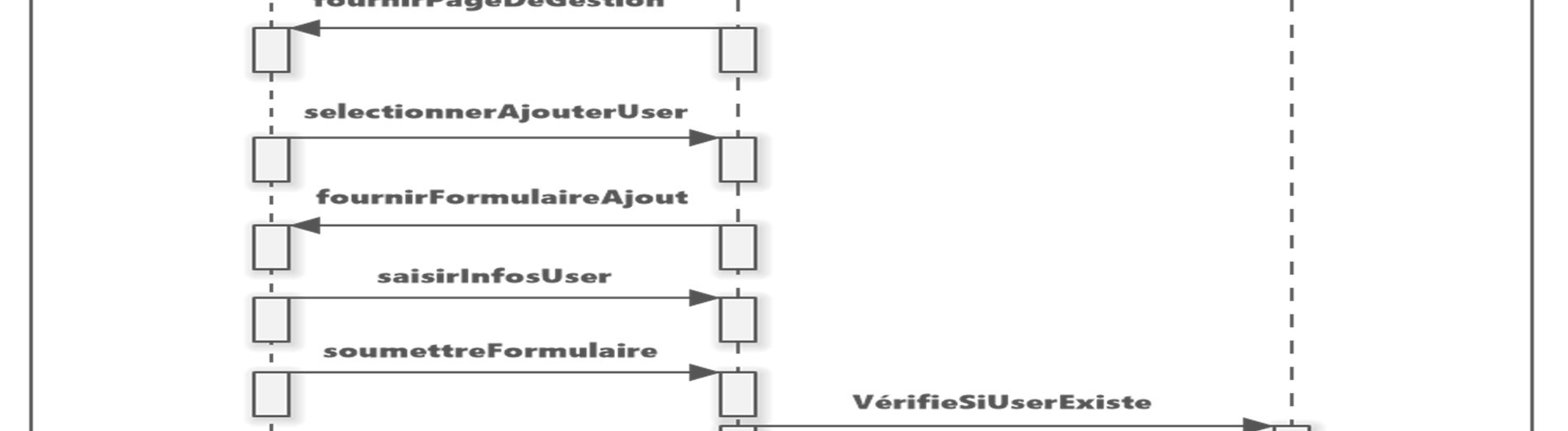
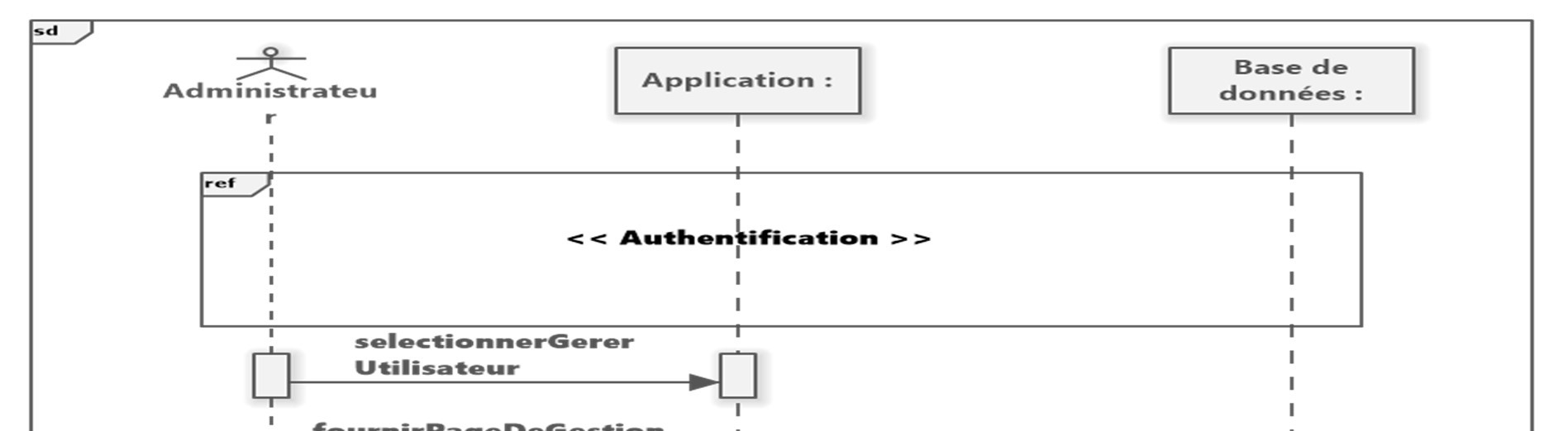
V.3.3. Diagrammes de séquences

V.3.3.1. Digramme de séquence 1 : cas d’utilisation « S’authentifier »



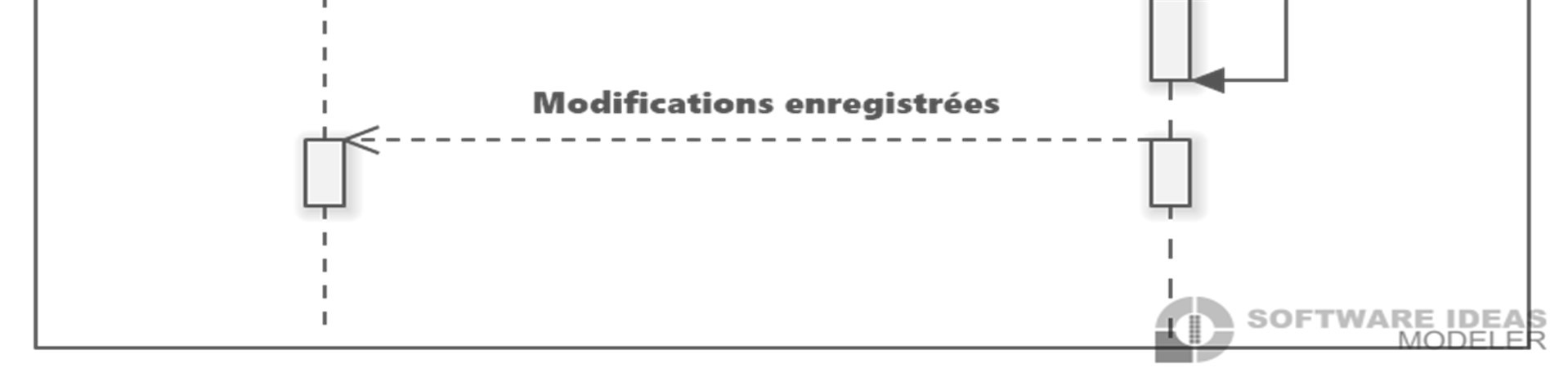
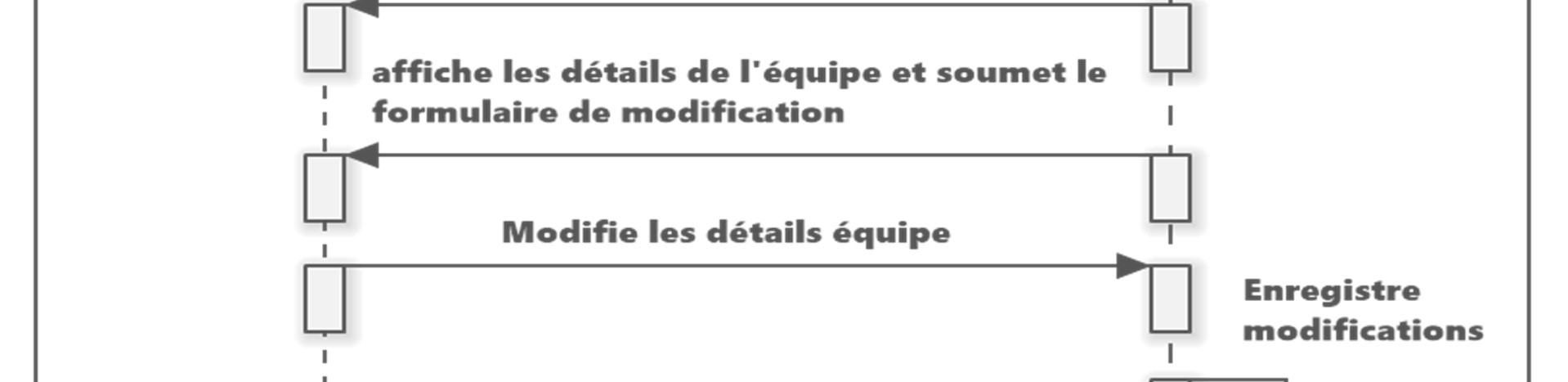
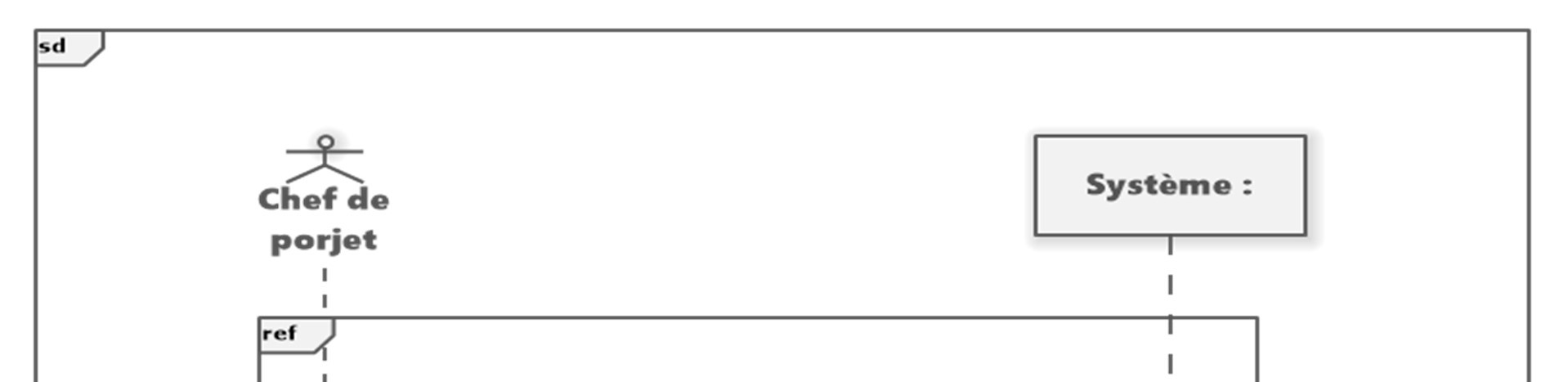
#### Figure 16- Séquence du cas d'utilisation : S'authentifier

V.3.3.2. Diagramme de séquence 2 : cas d’utilisation « Ajouter utilisateur »



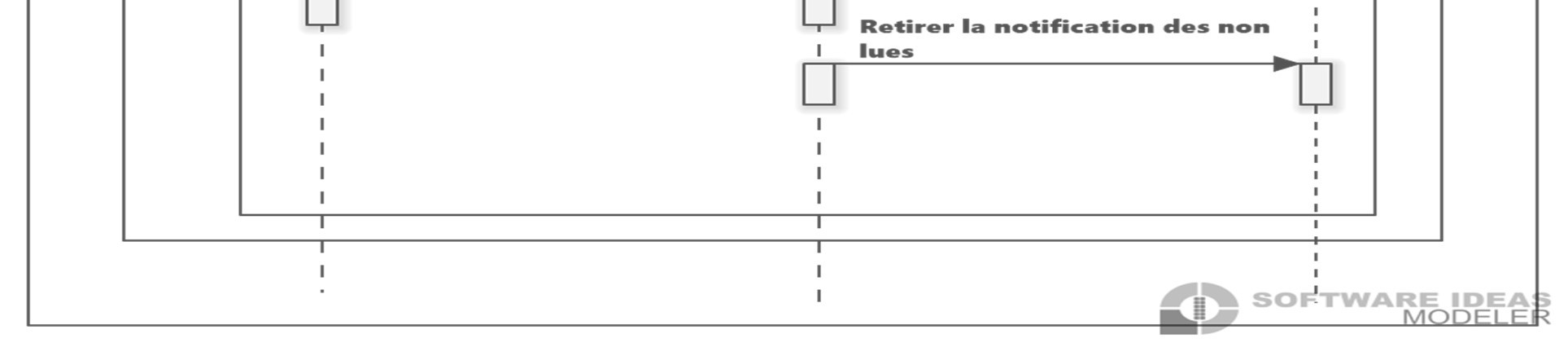
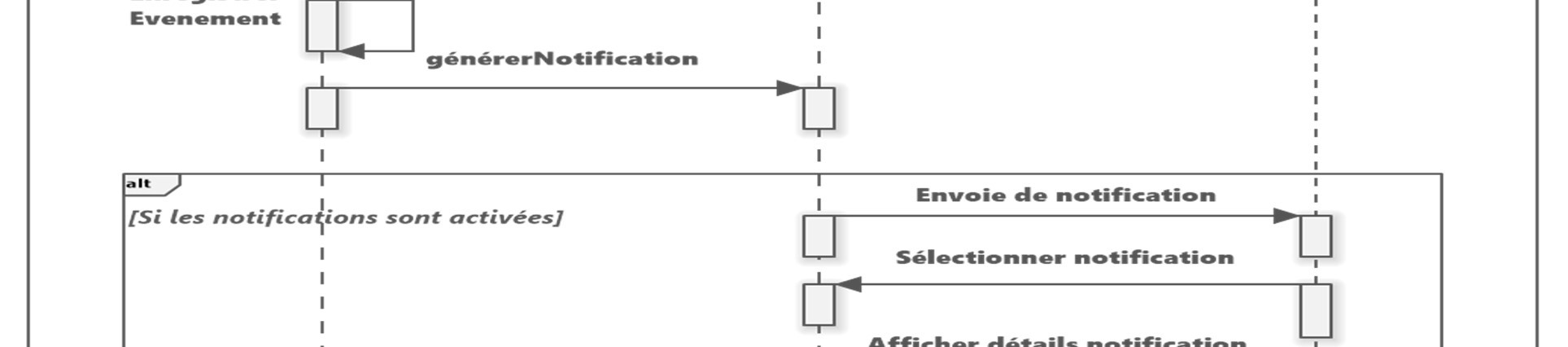
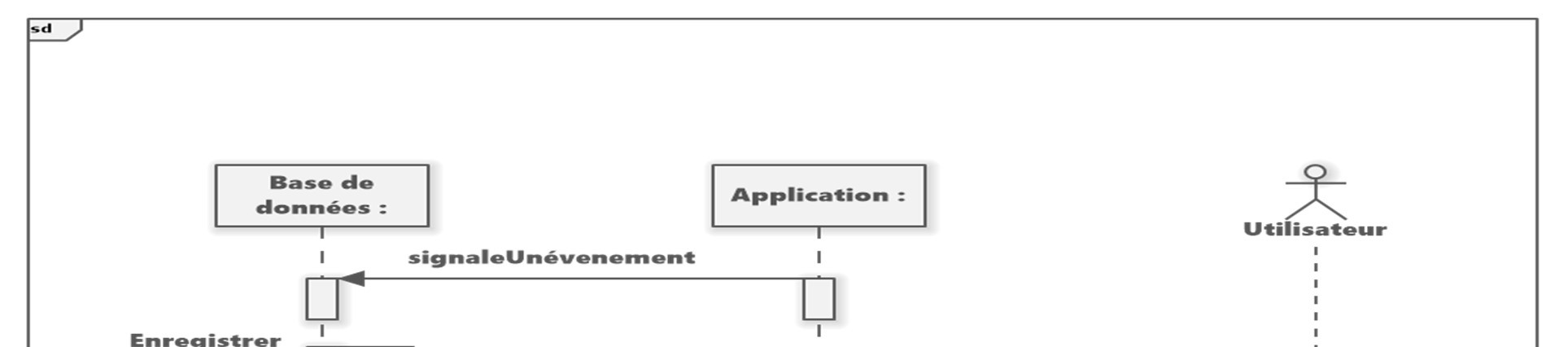
#### Figure 17- Séquence du cas d'utilisation : Ajouter Utilisateur

V.3.3.3. Diagramme de séquence 3 : cas d’utilisation « Modifier équipe »



#### Figure 18- Séquence du cas d'utilisation : Modifier Equipe

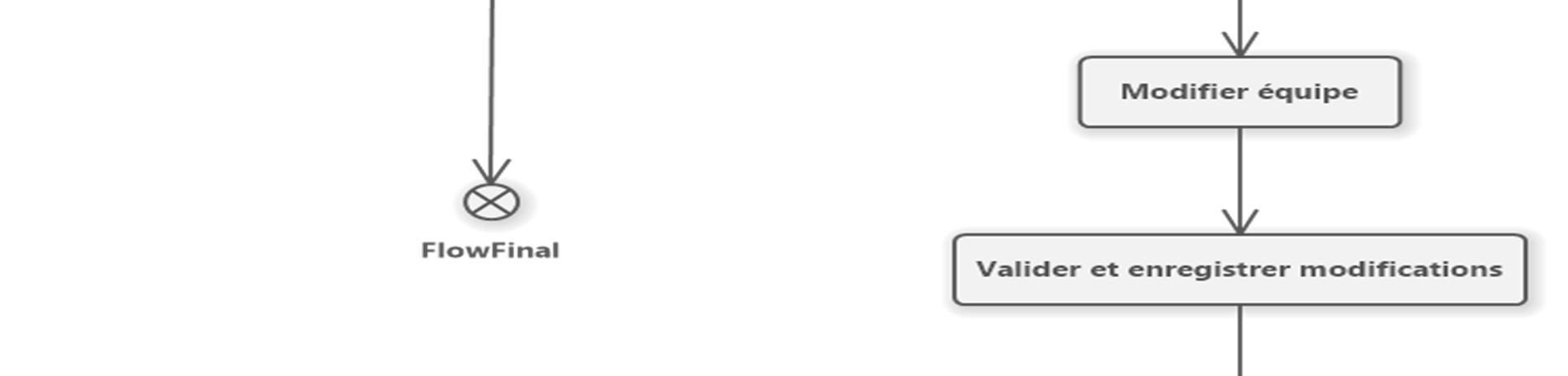
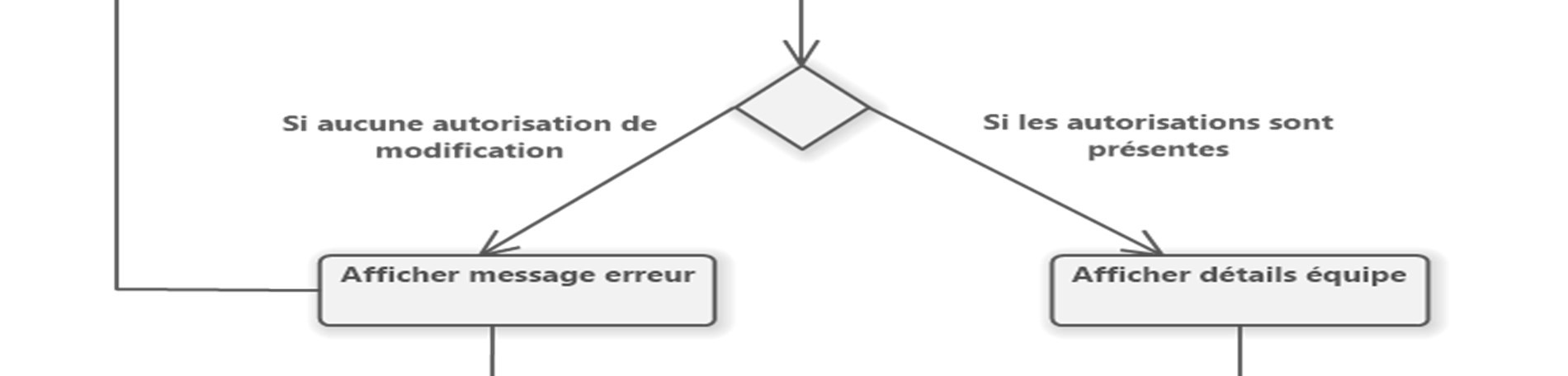
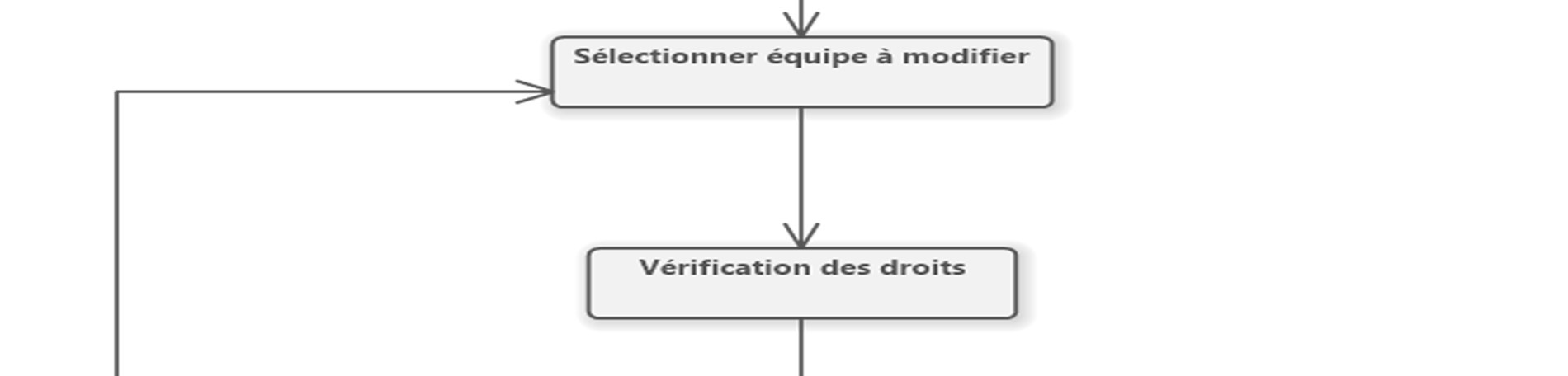
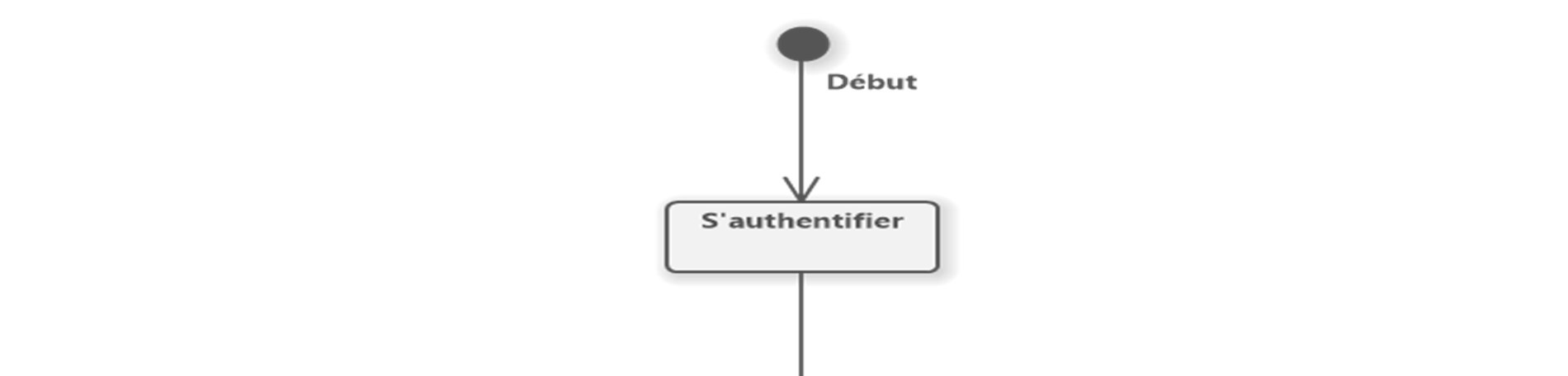
V.3.3.4. Diagramme de séquence 4 : cas d’utilisation « Recevoir notification »



#### Figure 19- Séquence du cas d'utilisation : Recevoir Notification

V.3.4. Diagrammes d’activités

Ce diagramme d’activité illustre la modification d’une équipe par un chef de projet.



#### Figure 20- Diagramme d'activité Modifier Equipe

Développement d’une application RESTful de gestion de projet : cas du service d’épidémiologie de l’Institut Pasteur de Bangui

# PARTIE 3 : IMPLÉMENTATION

Mémoire de fin de cycle Réalisé par BOGUIN Axel Tony

# Chapitre VI : Mise en œuvre

Lors du processus de développement d’une application, l’une des étapes les plus importantes à aborder est la mise en œuvre. La mise en œuvre d’une application Web est une étape cruciale qui transforme les concepts théoriques et les plans de conception en une solution fonctionnelle et tangible. Ce chapitre détaille le processus de développement de notre application web de gestion de projet, en mettant l’accent sur les choix techniques, les défis rencontrés, les solutions adaptées et vise à fournir une compréhension approfondie de la mise en œuvre pratique de notre application.

### VI.1. Architecture de l’application

L’architecture d’une application, aussi appelée architecture applicative, décrit les modèles et les techniques utilisées pour concevoir et créer une application. Il est important, lors du développement d’une application, de décrire l’architecture de celle-ci. Elles peuvent être physiques, logicielles et logiques.

VI.1.1Architecture globale de l’application

Structure globale

La structure globale de notre application définit comment les éléments du frontend et du backend communiquent entre eux. Dans notre solution, les éléments visuels et logiques sont séparés de manière à rendre le code plus lisible et plus facilement maintenable. Ainsi, le frontend est composé uniquement de codes écrits en HTML, CSS, Javascript et des tags. Les fichiers du backend sont composés de codes écrits en Python. Pour communiquer, ils utilisent le protocole HTTP et ses méthodes ; le frontend envoie des requêtes au backend qui les traite et renvoie des réponses. La communication entre les deux parties suit donc les étapes suivantes :

* Requête du frontend : le frontend envoie une requête HTTP au backend (avec ou sans données) en utilisant les différentes méthodes (GET, POST…) ;
* Traitement de la requête par le backend : le backend reçoit la requête et l’utilise pour exécuter une action appropriée (génération de contenu, accès à la BDD …) ;
* Réponse du backend : le backend envoie une réponse HTTP au frontend qui contient des données qui sont soit les résultats d’une requête de BDD, soit le fichier HTML d’une page

Web ;

* Rendu de la réponse par le frontend : le frontend reçoit la réponse et l’utilise pour mettre à jour l’interface utilisateur.

Lors des échanges de données entre le frontend et le backend, notre solution utilise principalement le format de données JSON, qui est un format léger, facile à analyser et à générer dans différents langages de programmation.

Les Modèles

Dans notre application, les modèles jouent un rôle crucial. En effet, ceux-ci servent à représenter les entités de l’application et gérer les interactions avec la base de données. Les modèles sont contenus dans le fichier « models.py ». Une classe dans un modèle représente une table dans la base de données et chaque attribut de cette classe représente un champ pour nos entités. Les codes SQL sont générés automatiquement lorsque ces tables sont créées dans le fichier des modèles. Les requêtes de base de données peuvent être effectuées en utilisant des ORM ; et les fonctionnalités de migration de base de données et les transactions sont également disponibles.

Les vues

L’une des parties les plus importantes de notre application concerne les vues. Une vue est une fonction ou une classe Python qui gère une requête HTTP et génère une réponse. Les vues de l’application sont écrites dans le fichier « views.py ». Elles ont différents rôles, dont :

* Intercepter les requêtes : elles analysent l’URL demandée et déterminent l’action à effectuer ;
* Traiter les données : elles accèdent à notre base de données afin de traiter les formulaires et d’effectuer d’autres traitements métier ;
* Générer des réponses : les vues servent à générer les réponses en HTML et JSON. Elles peuvent le faire aussi pour tout autre format de données si cela figure plus tard dans l’application.

Nos fichiers de vues contiennent tous les types de vues qui sont adaptées à toutes les situations, à savoir : la vue basée sur les fonctions, la vue basée sur les classes et la vue générique.

Les Template

Contrairement aux vues et modèles qui sont des éléments du backend, les Template sont des éléments du frontend. Ils représentent tout fichier HTML dans notre application Web et ces fichiers des variables et des balises spéciales qui permettent de générer du contenu dynamique. Ils ont pour rôle dans l’application :

* Combiner la logique métier et la présentation : les Template permettent de mélanger du code Python avec du HTML pour générer du contenu personnalisé pour chaque utilisateur ;
* Réutilisation du code : ils peuvent être inclus dans d’autres templates pour éviter la duplication du code.

VI.1.2. Architecture logicielle de l’application

Architecture monolithique

Pour l’architecture logicielle de notre solution, nous avons opté pour une architecture monolithique. L’architecture monolithique est un système unique et autonome regroupant toutes les fonctionnalités dans un seul bloc. Les applications monolithiques ont été conçues pour traiter de multiples tâches connexes. Il s’agit généralement d’applications complexes qui englobent plusieurs fonctions étroitement couplées.

Cette architecture présente plusieurs avantages pour notre solution, dont :

* Plus simples à développer et à déployer initialement ;
* Cohérence du code ;
* Coûts initiaux réduits ;

Toutefois, cette architecture peut rencontrer des défis majeurs tels que :

* La difficulté d’évolution : les applications basées sur une architecture monolithique sont plus difficiles à faire évoluer, surtout lorsqu’elles deviennent volumineuses et complexes ;
* Le manque de flexibilité : elles sont difficiles à adapter au changement de besoins et de technologies ;
* Impacte global des modifications : toute modification du code affecte l’ensemble de l’application.

En vue des défis majeurs auxquels peut faire face notre solution à l’avenir, il serait plus intéressant que l’application évolue vers une architecture microservices qui serait plus flexible.

Architecture client-serveur

L’architecture client-serveur est un modèle de conception d’application qui joue un rôle fondamental dans le développement de solutions web. Elle repose sur une interaction entre un client, qui demande des services, et un serveur qui les fournit.

Pour le cas de notre application, nous utilisons une architecture client-serveur. Dans ce modèle, le client envoie une requête au serveur en utilisant le HTTP. Le serveur va traiter la requête et renvoyer une réponse. Cette séparation des responsabilités permet une gestion efficace des ressources et une meilleure répartition de la charge de travail.

#### VI.2. Présentation des outils et technologies utilisées

Le développement d’une application passe par plusieurs processus. A chaque niveau de développement, des outils et des technologies sont utilisés. Pour le développement de notre application, nous avons dû utiliser plusieurs outils et technologies qui sont plus utilises les uns que les autres.

VI.2.1. Outils et services utilisés

##### Système d’exploitation : Ubuntu

La distribution de Linux, Ubuntu, est celle que nous avons utilisée afin de mener à terme le développement de notre application. Il est un OS qui est beaucoup utilisé pour le développement d’applications.

##### Environnement de modélisation : Software Idea Modeler

C’est un logiciel utilisé pour la modélisation. Cet outil a servi à concevoir nos diagrammes UML et certaines figures telles que l’architecture d’un site Web statique et dynamique.

###### Editeur de texte : VS Code

Largement utilisé par les développeurs, VS Code est l’un des meilleurs éditeurs de code. Il dispose de nombreuses extensions qui facilitent le développement et augmentent la productivité.

###### Outils de contrôle de version : Git

Git est un outil utilisé pour faire le contrôle de version ou encore « versionning ». Il permet de conserver un historique des modifications apportées sur un projet afin d’identifier rapidement les problèmes et les modifications.

###### Service d’hébergement : GitHub

GitHub est un service d’hébergement en ligne qui permet d’héberger des dépôts ou « repository ». Il nous a servi pour héberger les différentes versions de notre application sur le Cloud.

###### Django REST Framework

Django REST Framework est une puissante boîte à outils flexible utilisée pour la création d’API Web. Cette boîte présente d’énormes avantages, parmi lesquels :

* Une API consultable via le navigateur : l’API consultable facilite la vie aux développeurs en leur conférant un énorme avantage. Grâce à cela, on peut faire l’exploration et les tests de l’API directement sur le navigateur ;
* La politique d’authentification : il propose directement des politiques d’authentification, y compris des packages pour OAuth1 et OAuth2 ;
* La sérialisation : Rest Framework prend en charge la sérialisation des données à partir de sources ORM ou non ORM ;
* Elle est personnalisable : on peut personnaliser cet outil en l’adaptant à nos besoins spécifiques ;
* La documentation complète et la communauté : il dispose d’une documentation très étendue en plus d’un excellent support de communauté.
* Utilisé et approuvé par plusieurs entreprises : les entreprises telles que Mozilla, Red Hat ou encore Heroku, utilisent Rest Framework.

VI.2.2. Technologies utilisées

###### Framework : Django

Django est un Framework Web puissant et polyvalent, conçu pour simplifier le processus de développement d’applications web. Il a été conçu en 2003 et est devenu open-source en 2005. Django est un Framework solide qui repose sur le concept du « Batteries included », ce qui implique qu'il propose de nombreuses fonctionnalités prêtes à être utilisées, comme la gestion des bases de données, la gestion des utilisateurs, la sécurité, l'administration web et bien d'autres encore. Cela offre aux développeurs la possibilité de se focaliser sur les aspects particuliers de leurs applications sans avoir à repenser la roue à chaque plan. En outre, Django soutient vivement les bonnes pratiques de développement web, telles que la distinction nette entre le code métier et la présentation, la protection automatique contre les failles de sécurité web courantes et la facilité d'extension grâce à un système modulaire dédié aux applications réutilisables. Il fonctionne en utilisant :

* Le MVT : le Modèle-Vue-Template décrit plus haut, est celui utilisé par Django pour

organiser le fonctionnement de ses applications ;

* Un ORM : l’ORM de Django est l’un des plus puissants en raison de sa syntaxe expressive, sa portabilité et sa gestion automatique des schémas.

SGBDR : MySQL

MySQL est un Système de Gestion de Base de Données Relationnelle très puissant qui utilise le langage SQL.

VI.2.3. Langages utilisés

* 1. Python : c’est un langage de programmation orienté objet qui est interprété, facile à apprendre et à comprendre. Il est largement utilisé, car il est open-source, puissant et flexible.

* 1. Javascript : Javascript est un langage de programmation qui peut être utilisé côté client ou serveur. Il est essentiel pour le développement d’applications Web. Il permet d’ajouter des interactions, des animations et du dynamisme aux pages Web.

* 1. HTML : est un langage de balisage hypertexte utilisé pour structurer le contenu d’une page web. Il est utilisé par pratiquement tous les sites web.

* 1. CSS : est un langage informatique utilisé pour styliser et mettre en forme des pages Web écrites en HTML. Il est un langage essentiel utilisé par tous les programmeurs frontend.

VI.2.4. Installations

L’utilisation de Django avec son langage de programmation Python, nous a permis d’installer plusieurs bibliothèques et librairies qui nous ont permis d’aboutir au résultat final.

* Django :

Avant de commencer toutes les manipulations pour le développement de notre application, nous avons d’abord installé Django. La commande suivante permet donc de le faire :

* Django-Datta-able :

C’est un Dashboard Django qui donne une certaine structure à la page grâce au Template qu’il offre pour faciliter la mise en place d’interface utilisateur.

* Celery

Celery est une bibliothèque Python qui permet de traiter les tâches de manière asynchrone. Ici, nous l’utilisons pour envoyer les e-mails de manière asynchrone.

* RabbitMQ

Il est un serveur de messagerie codé en Erlang. Il est disponible pour les OS tels que Linux, Mac Os ou Windows. Il est utilisé pour échanger des messages entre différents services de manière synchrone ou asynchrone.

* Daphne

C’est un serveur ASGI codé en Python pour UNIX. Il a été développé pour alimenter Django Channels. Il supporte la négociation automatique des protocoles, il n’y a donc pas besoin de préfixer les URL pour déterminer les points de terminaison WebSocket par rapport aux points de terminaison HTTP.

* Django Channels

Channels est une bibliothèque qui ajoute des fonctionnalités asynchrones à Django. Il permet d’ajouter le support pour web socket, long-polling HTTP, l’offloading de tâches et d’autres fonctionnalités asynchrones au code.

##### VI.2.5. Implémentation de la sécurité

L’Institut Pasteur de Bangui est un centre de recherche qui traite des données très sensibles. Il est donc impératif, pour notre solution, de garantir la sécurité de ces données. Ceci est l’une des raisons principales qui nous a poussé à choisir un Framework tel que Django. Django intègre plusieurs mécanismes de sécurité qui protègent les données des applications Web contre les attaques les plus courantes. Dans le tableau suivant, nous allons décrire les vulnérabilités courantes des applications Web, et comment Django prend les mesures de sécurité contre ces dernières :

Tableau 10- Mesures de sécurité intégrées à Django

|  |  |  |
| --- | --- | --- |
| Vulnérabilités | Descriptions | Mesures prises par Django |
| Cross Site Scripting (XSS) | Les attaques XSS permettent à un intrus d’injecter des scripts clients dans les navigateurs des utilisateurs | L’utilisation des gabarits Django protège contre la majorité des attaques XSS. Les gabarits de Django échappent des caractères spécifiques qui sont particulièrement dangereux en HTML |
| Cross Site Request Forgery  (CSRF) | Elles permettent à une personne malveillante d’exécuter des actions malveillantes en utilisant les données d’authentification d’un autre utilisateur sans que ce dernier ne s’en rende compte. | Django offre une protection intégrée contre la plupart des attaques de type CSRF, du moment que le module soit activé. |
| Stockage non sécurisé de mot  de passe | Il peut permettre à un attaquant d’accéder aux mots de passes en clair des utilisateurs | Django fourni un mécanisme de hachage des mots de passe sécurisé par défaut en utilisant l’algorithme PBKDF2 avec un hash SHA-256 |
| Injection SQL | Les attaques par injection SQL permettent à un attaquant d’exploiter les vulnérabilités inhérentes à la manière dont une base de données exécute les requêtes de recherche. | Django utilise des requêtes SQL paramétrées, ce qui protège les applications contre les injections SQL. |
| Clickjacking | Elle est une attaque, où un attaquant utilise plusieurs cadres transparents pour tromper un utilisateur en cliquant sur un bouton ou un | Django fournit une protection intégrée contre le clickjacking en utilisant le middleware X-  Frame-Option |
|  | lien sur une autre page alors qu’il pense qu’il clique sur le dessus de la page. |  |

Mise à part les systèmes de sécurité intégrés à Django, nous opterons aussi pour une communication utilisant le protocole HTTPS et un serveur DNSSEC.

Le HTTPS est une version sécurisée du HTTP. Ce protocole joue un rôle clé dans l’aspect sécurité et de confidentialité :

* Sécurité des données : il utilise le chiffrement pour sécuriser les données échangées entre le navigateur et le serveur ;
* Confidentialité : il assure la confidentialité des données en garantissant que seules les parties autorisées peuvent accéder aux informations échangées.

Le DNSSEC est une extension du DSN qui ajoute une couche de sécurité supplémentaire. Il garantit :

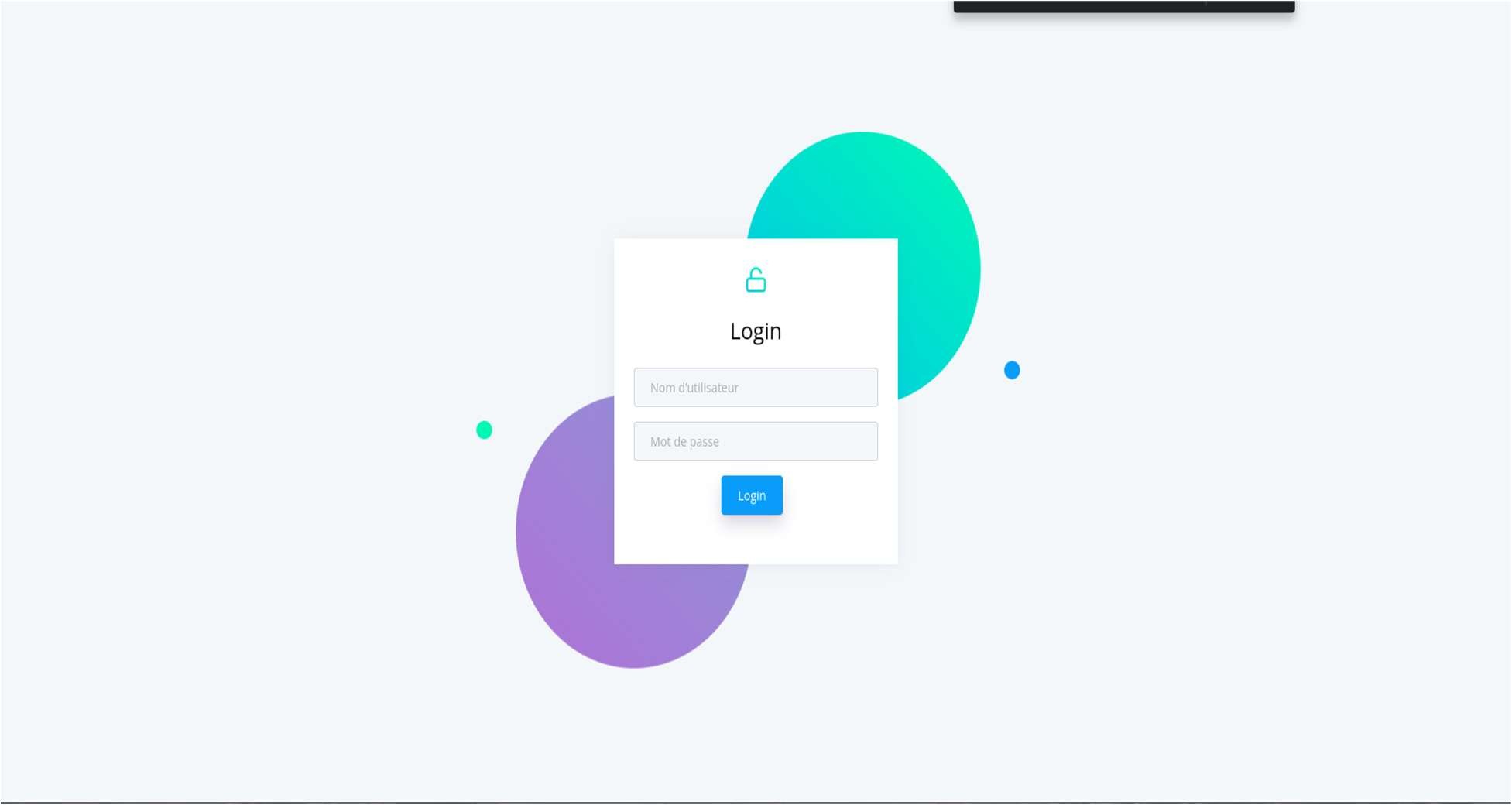
* L’authenticité : il fournit l’autorité d’origine, l’intégrité des données et le déni d’existence authentifié. Cela signifie qu’il confirme que l’on communique bel et bien avec l’application Web ;
* Sécurité des données : il utilise un chiffrement à clé pour vérifier que le DNS correspond à un domaine légitime. Cela rend les données illisibles pour toute personne qui intercepterait la communication ;
* Protection contre les attaques : il a pour vocation de protéger la résolution DNS contre un certain nombre d’attaques, notamment les tentatives d’usurpation DSN.

Ainsi, toutes ces mesures de sécurité garantiront au maximum la sécurité et l’intégrité des données.

VI.2.6. Présentation des écrans de test de la solution

 Accès à l’application pour l’administrateur

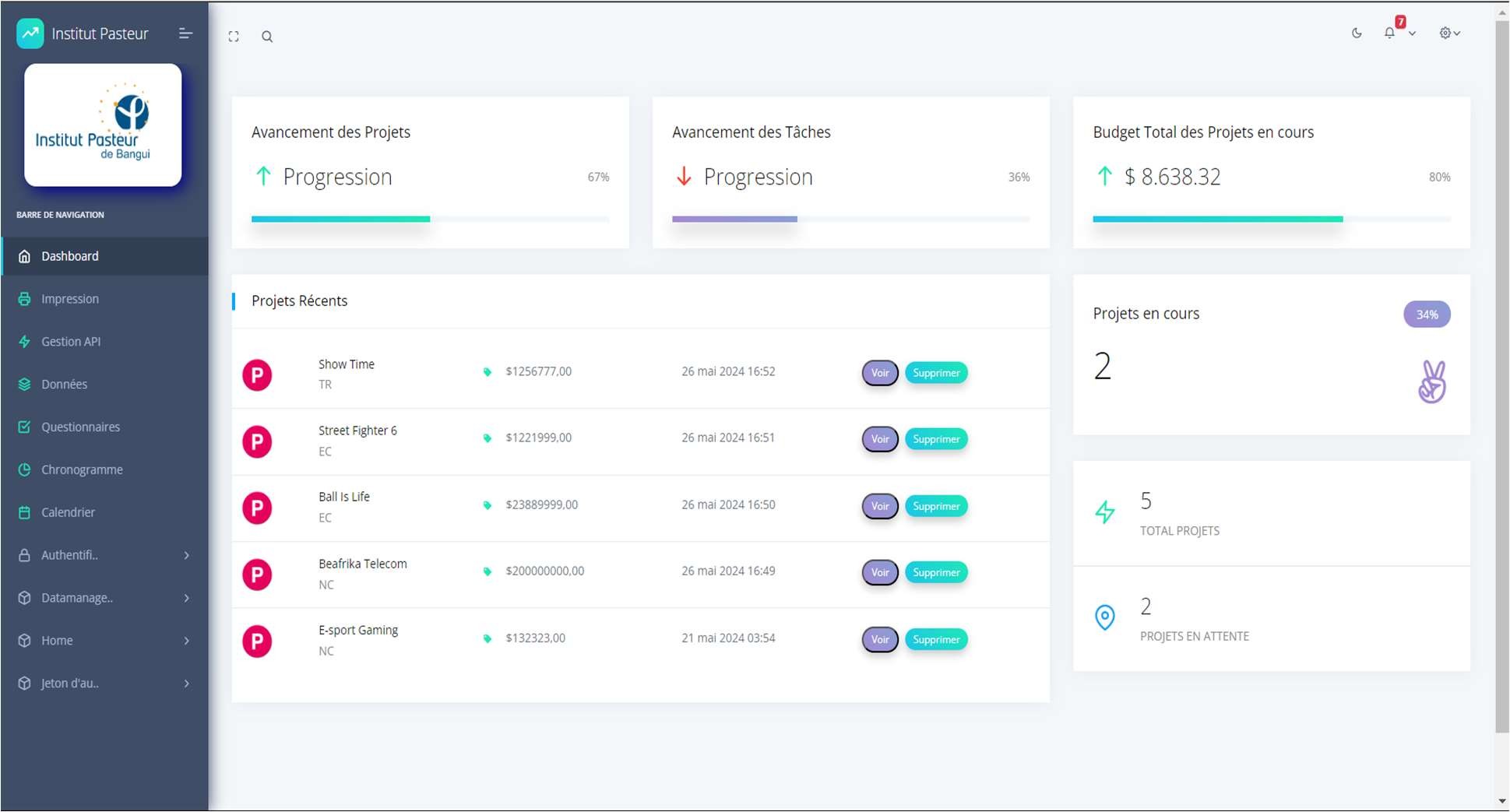
Il s’agit ici de la page d’authentification. Lorsque l’administrateur ou un autre utilisateur autorisé à visiter le site administrateur saisi l’adresse de l’application, il sera amené à s’authentifier ici avant d’accéder à l’application. Il suffira de renseigner deux informations : Login et mot de passe.



###### Figure 21- Page de Login de l'espace d'Administration

Dashboard

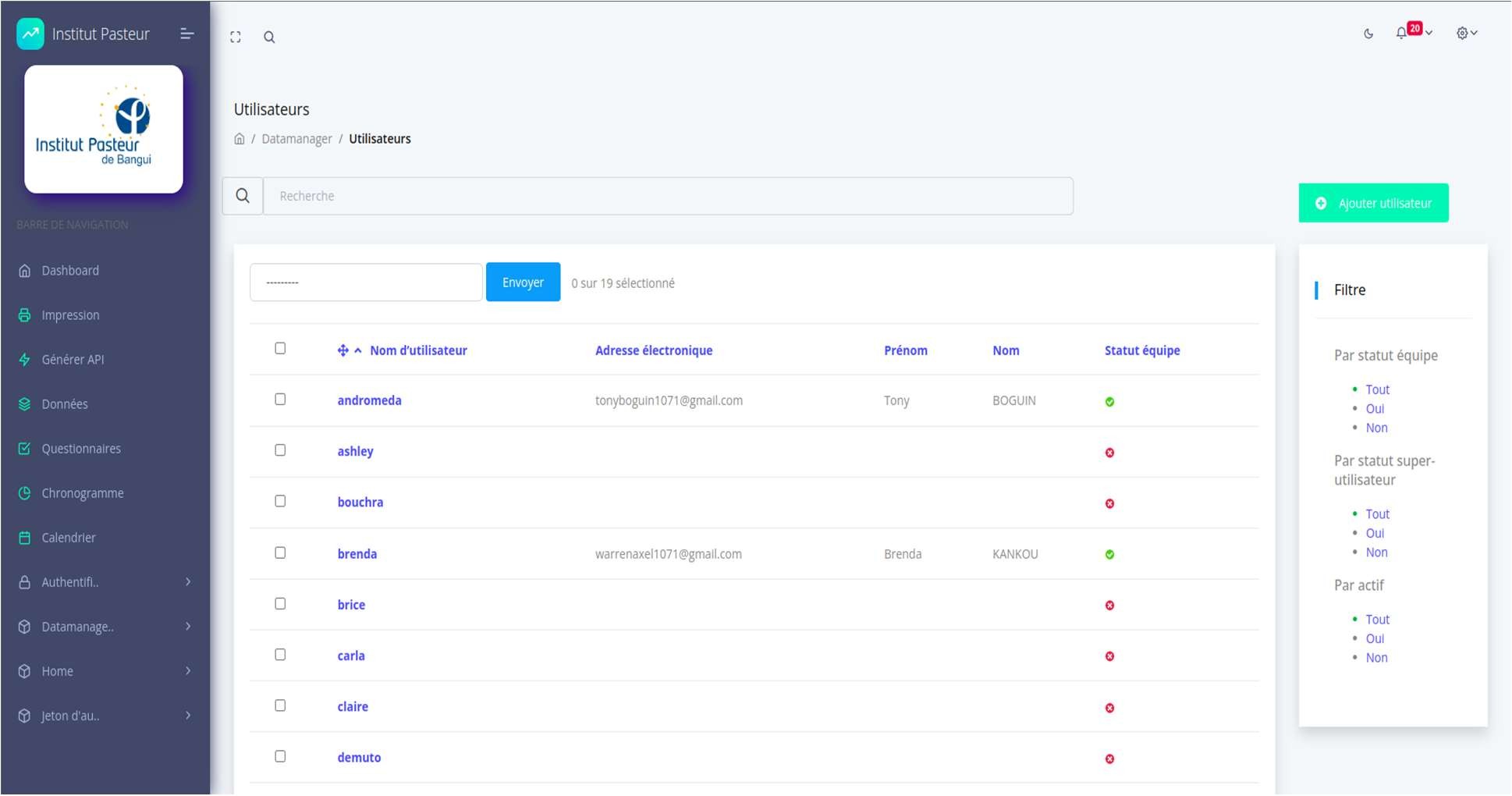
Le Dashboard, ou encore « Tableau de bord », représente la page qui permet d’avoir une vue d’ensemble sur la base de données de la gestion de projet.



###### Figure 22- Tableau de bord partie Administration

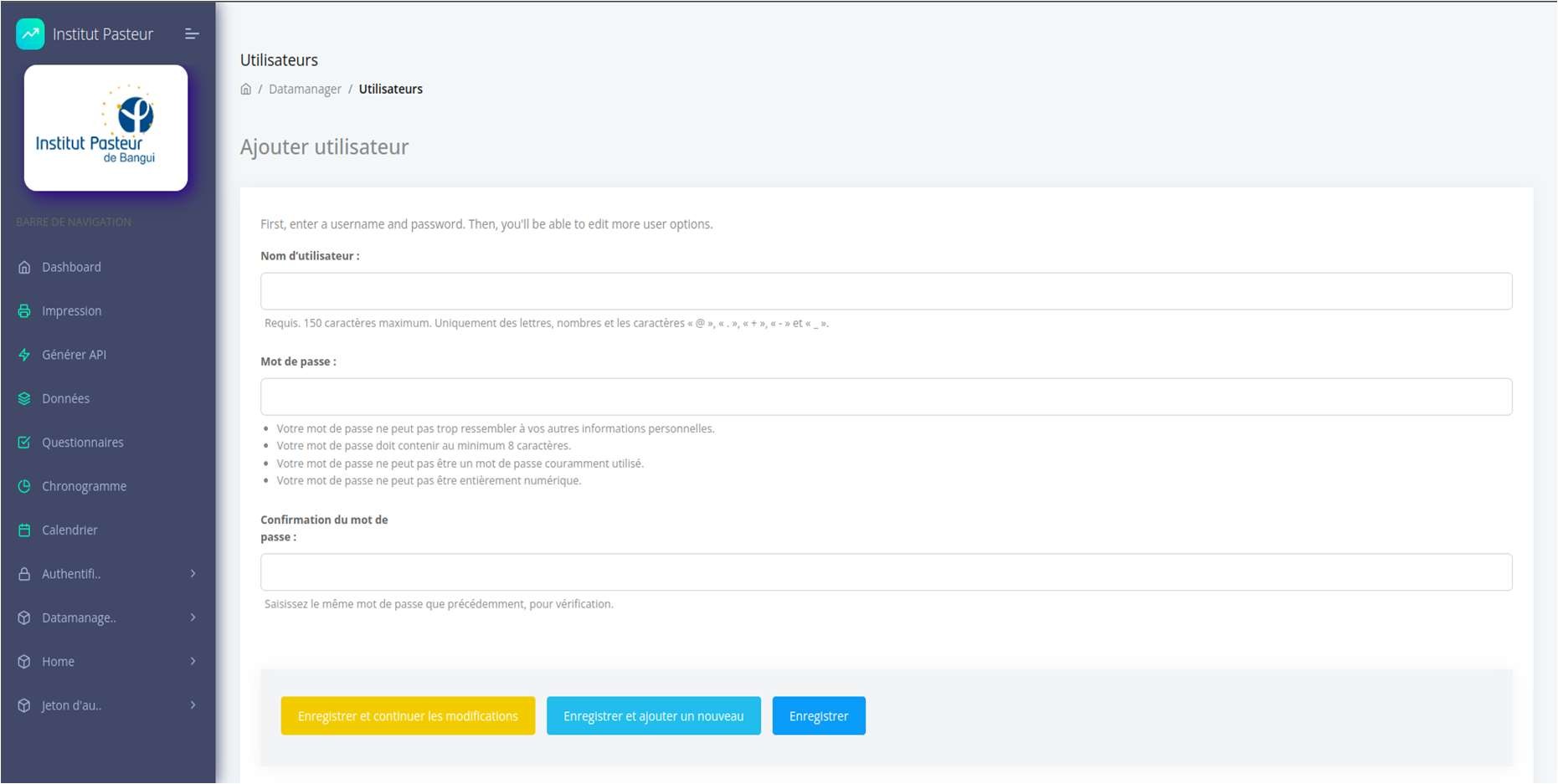
Gestion des utilisateurs et des droits

Cette section montre l’accès à la page pour la gestion des utilisateurs. Elle présente d’abord une liste de tous les utilisateurs. A partir de cette page, on peut ajouter, modifier ou supprimer un utilisateur.

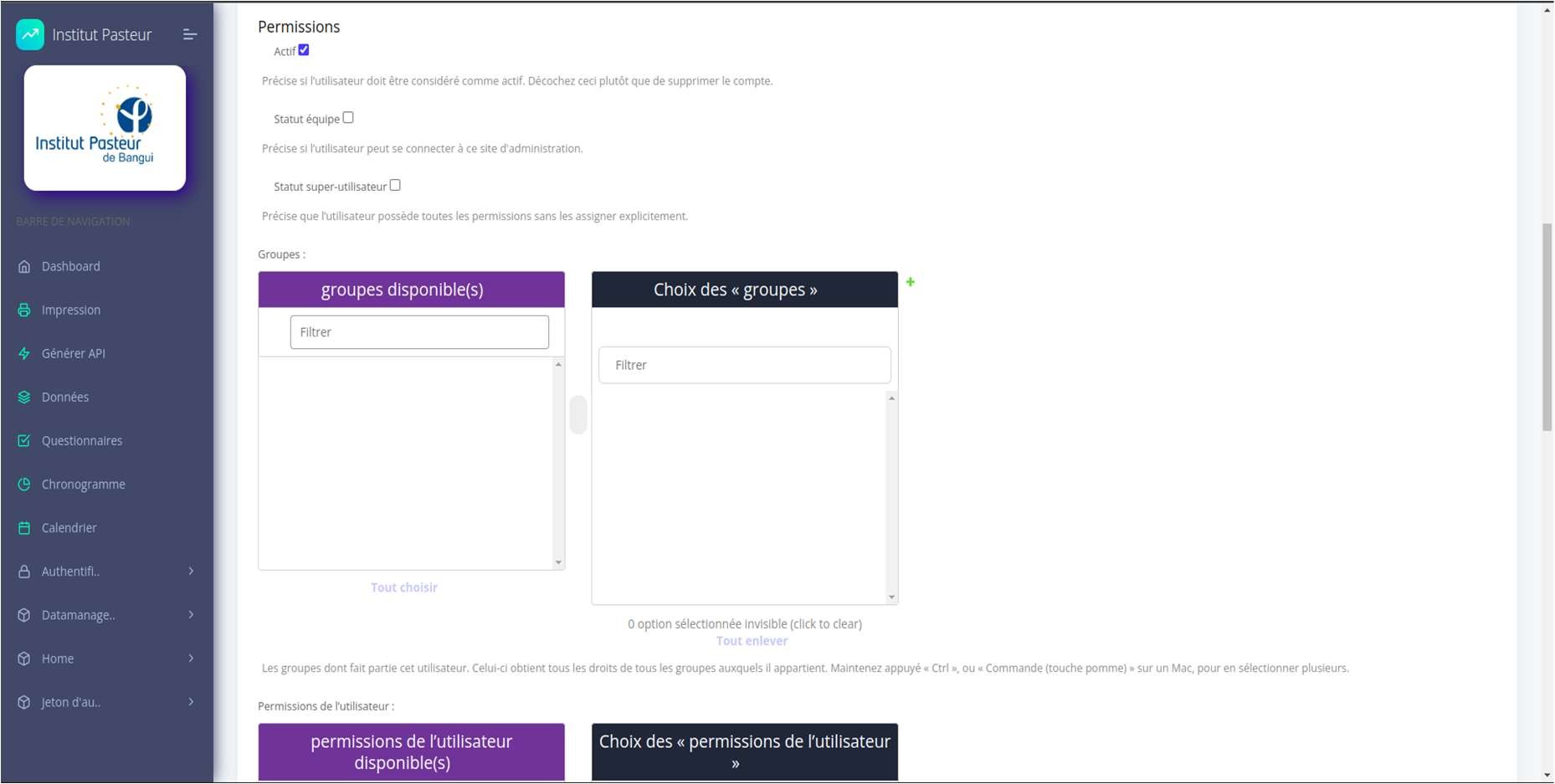


###### Figure 23- Gestion des utilisateurs

Le formulaire ci bas présente l’ajout d’un utilisateur, puis le suivant, l’attribution des droits.



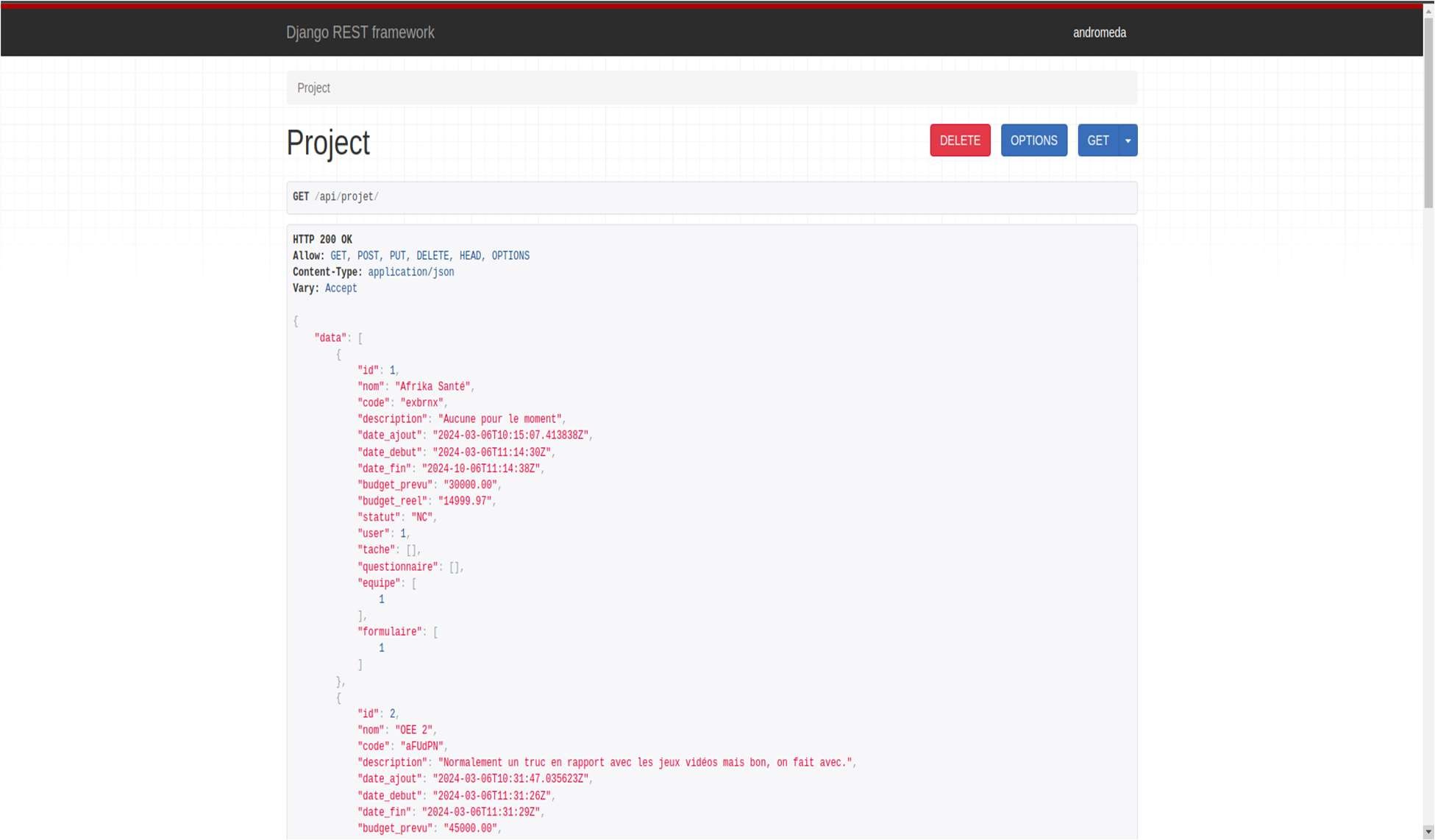
###### Figure 24- Ajout d'un compte utilisateur



###### Figure 25- Gestion des permissions

Interface API Django-Rest-Framework

Cette section présente l’interface API qu’offre Django Rest Framework pour gérer l’API et faire les tests avec les méthodes de la requête HTTP (GET, POST, PUT, etc.).

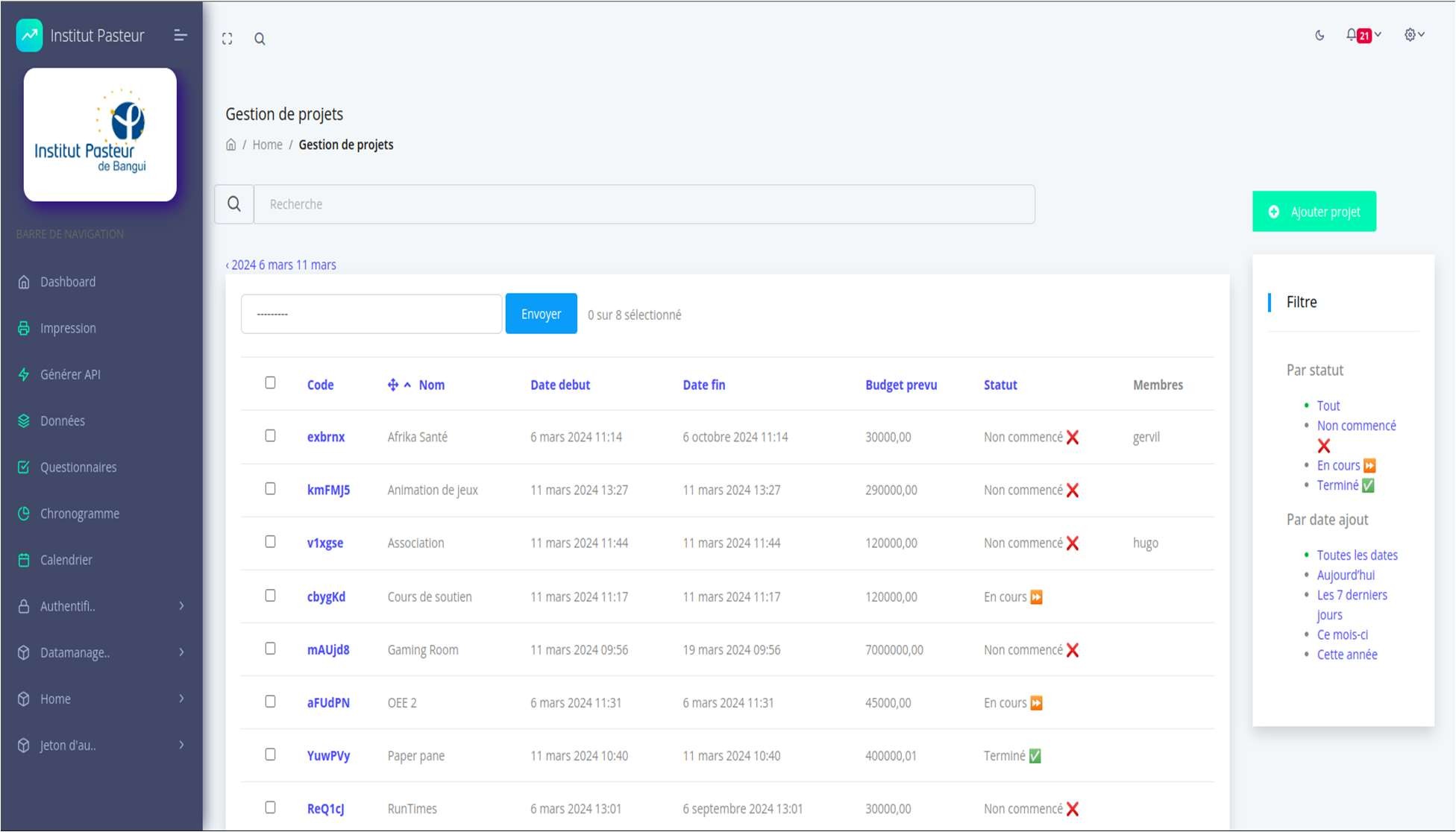


###### Figure 26- Interface de gestion de l'API

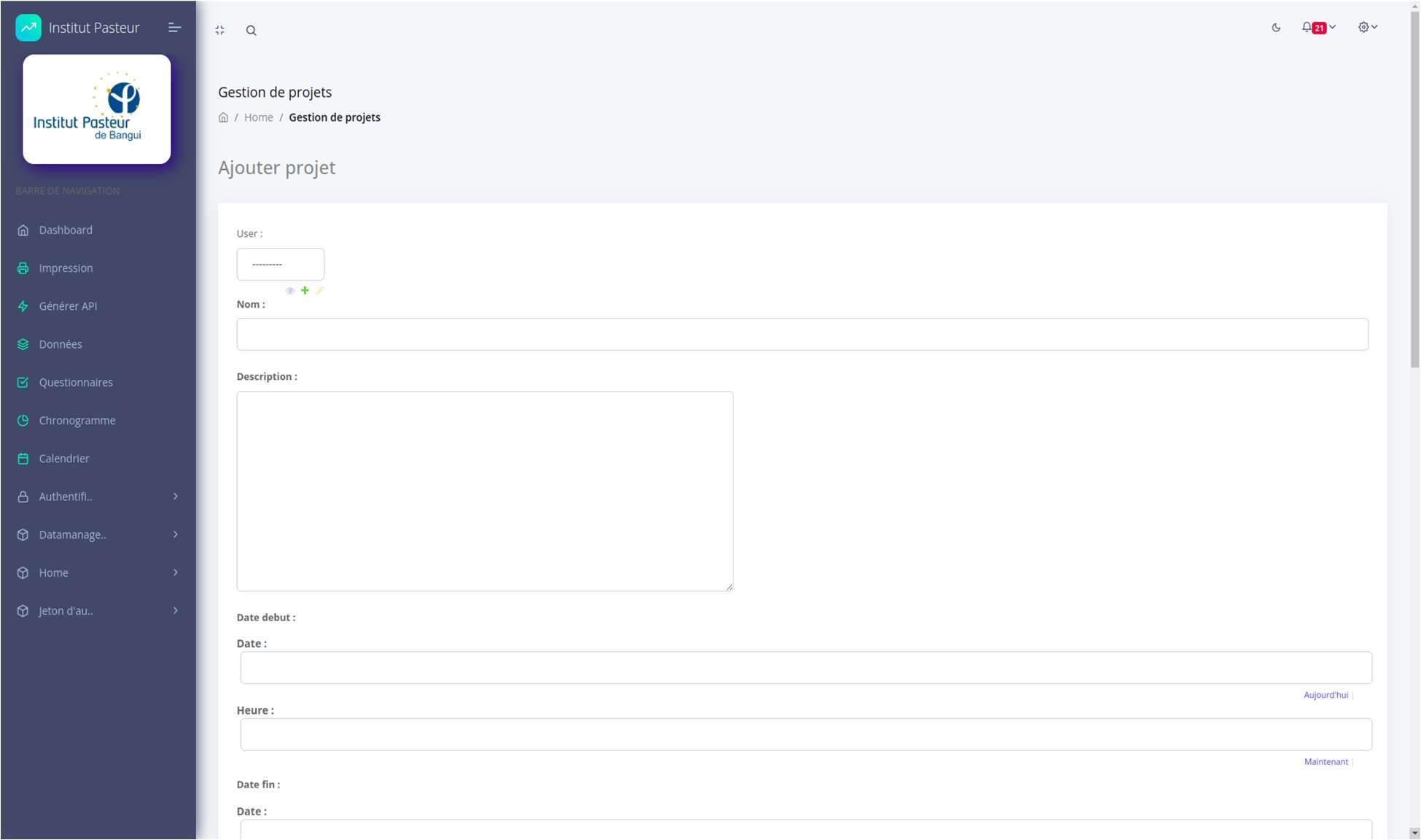
 Accès à l’application pour les responsables

Gestion de projet

Sur cette page, le responsable de projets peut gérer les projets et avoir la main mise sur tous les projets. Les captures suivantes montrent l’accès à la page de gestion de projet et d’ajout d’un projet.



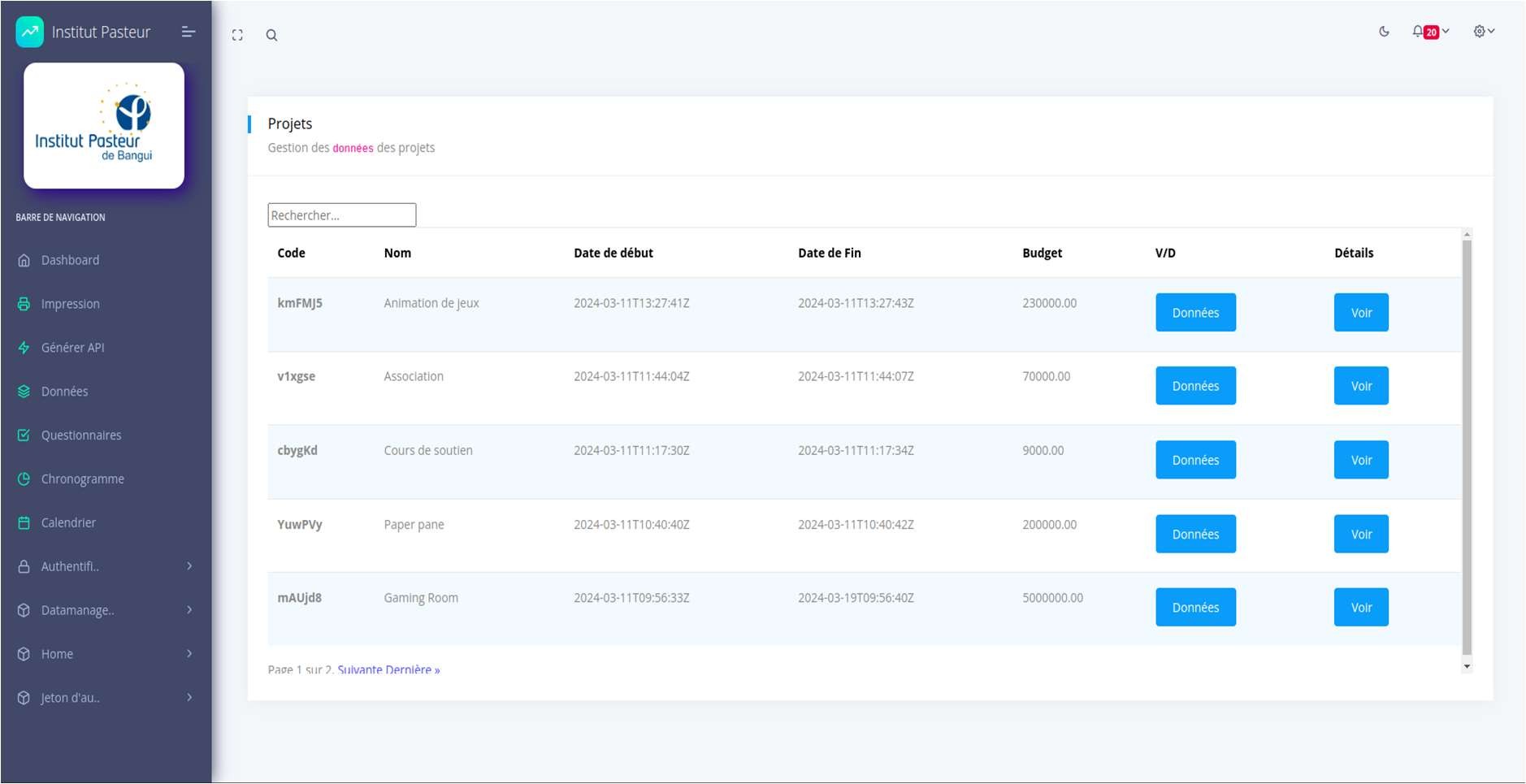
###### Figure 27- Espace de gestion des projets



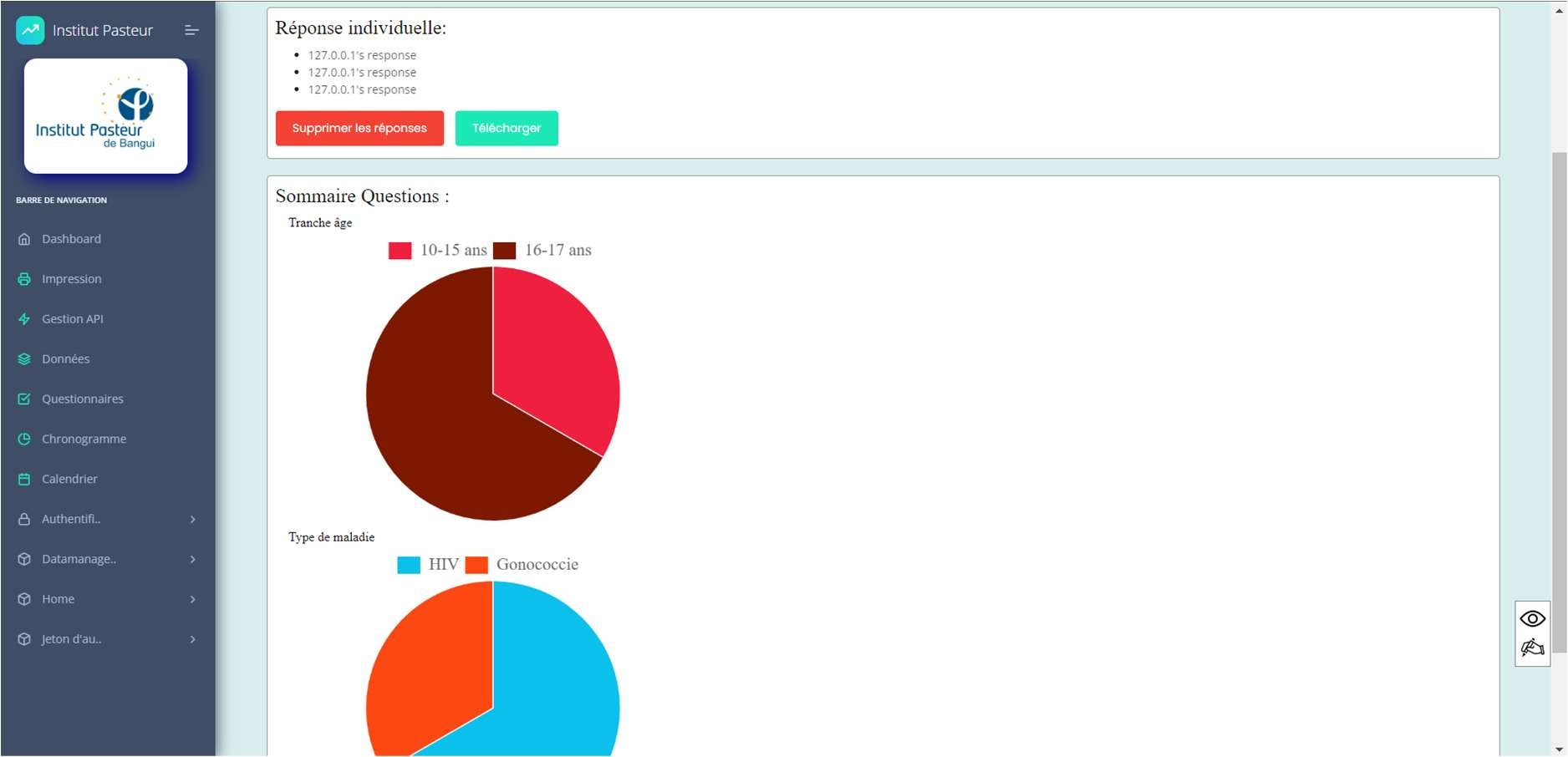
###### Figure 28- Formulaire d'ajout de projet

Données projets

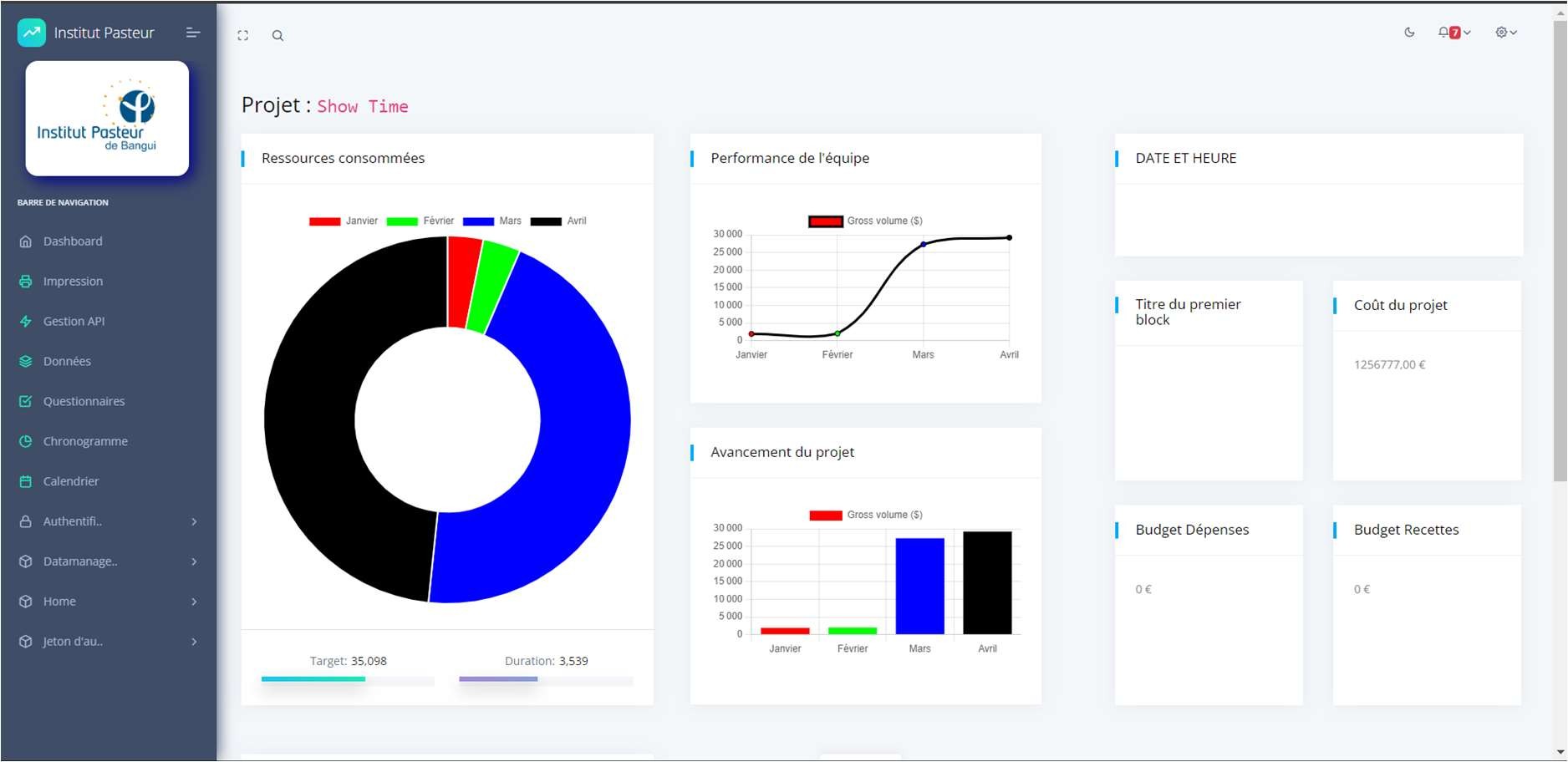
Ces interfaces présentent un tableau dans lequel on peut repérer facilement un projet et modifier les données récoltées sur celui-ci et la statistique de données récoltées sur un projet.



###### Figure 29- Espace de gestion des données des projets



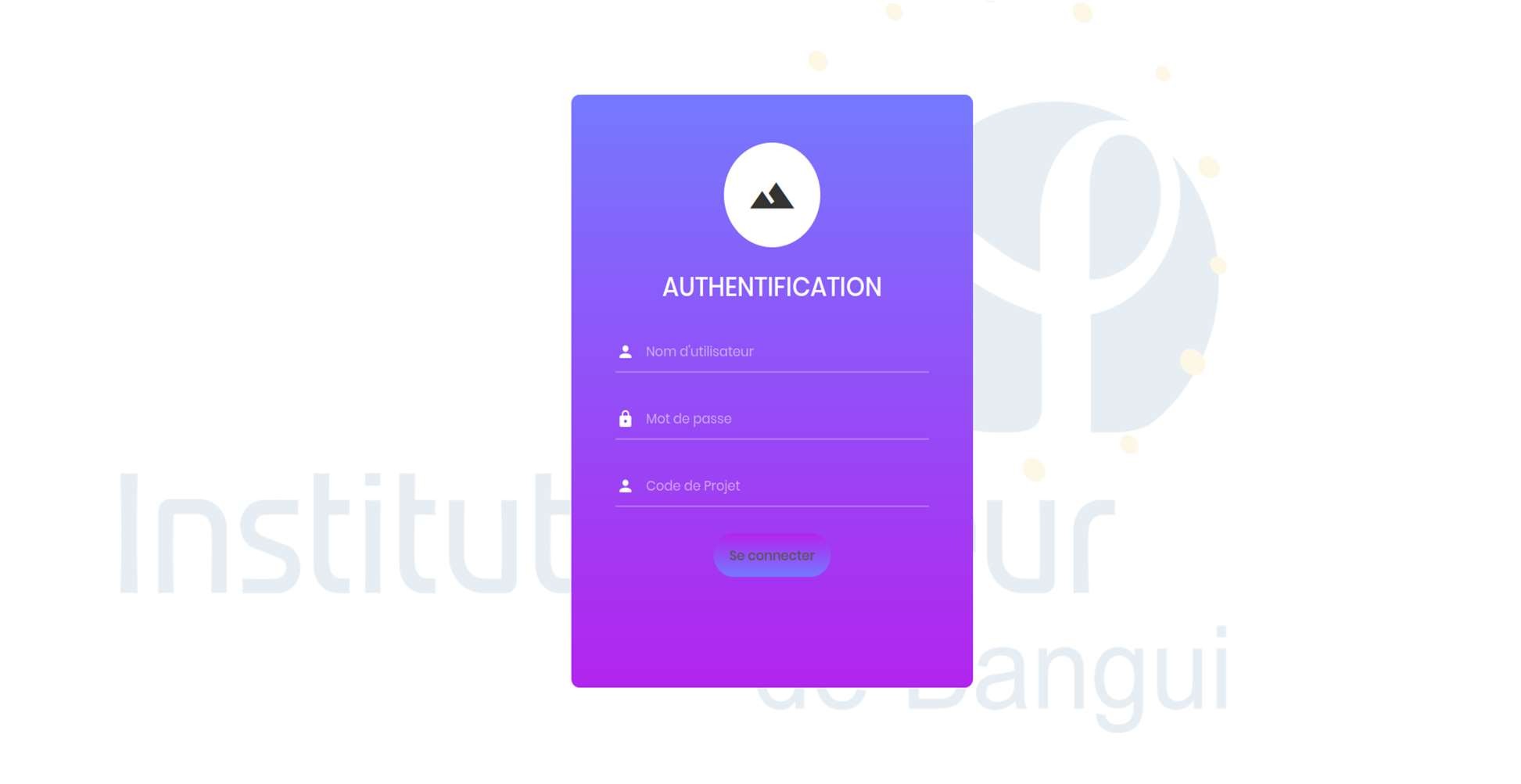
###### Figure 30- Statistique des données récoltées



###### Figure 31- Espace de visualisation des KPI d'un projet

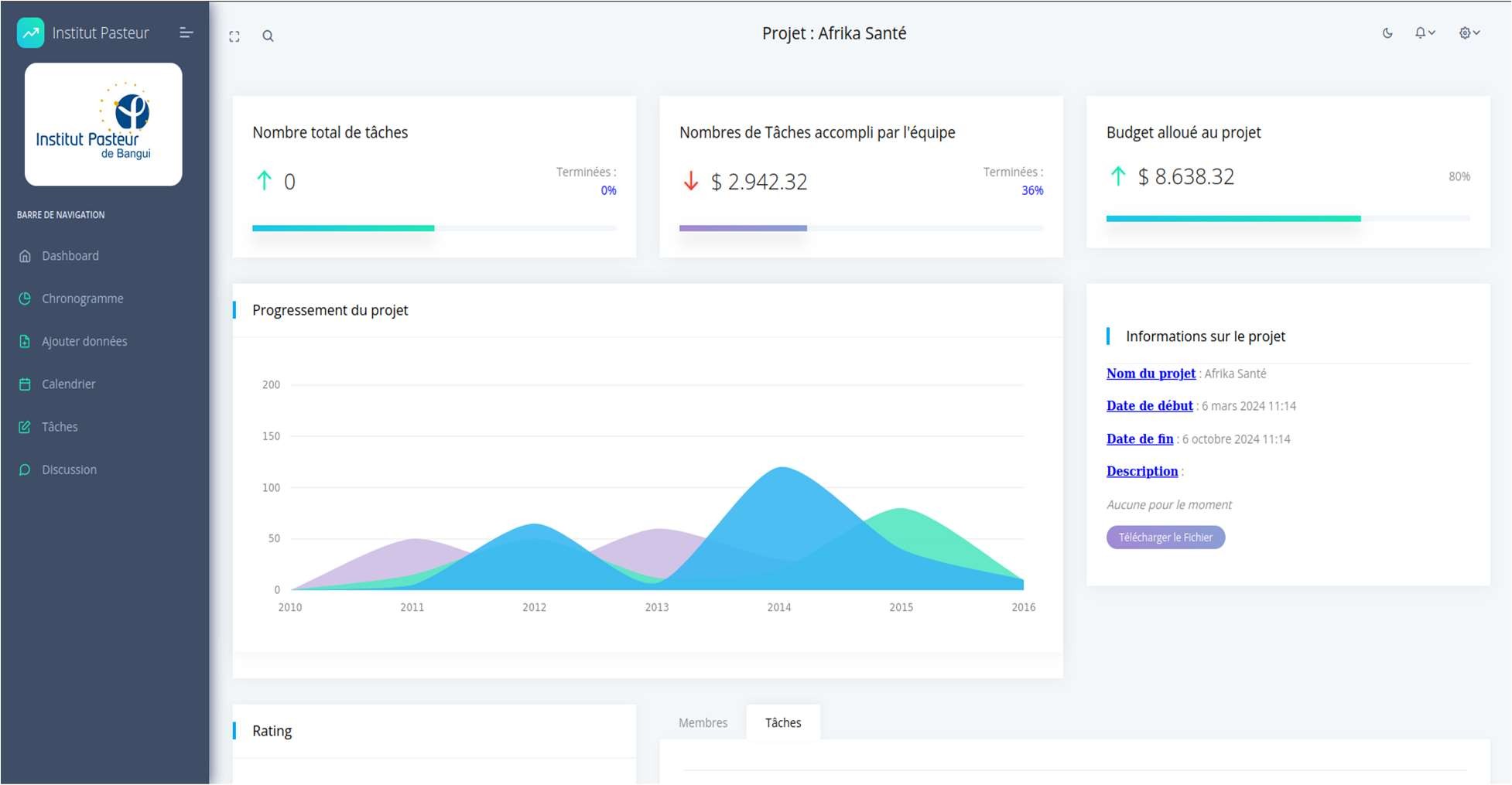
 Accès à l’application pour les Data managers (et membres)

Lorsqu’un utilisateur Data manager ou membre de projet essaye d’accéder à l’application, il devra remplir le formulaire de cette page d’authentification qui comprend trois (03) informations : Login, Mot de passe et Code de projet.



###### Figure 32- Page d'authentification des membres du projet

Une fois authentifié, cet utilisateur accède à un tableau de bord présenté comme ci-dessous. Il présente à l’utilisateur les informations qu’il a le droit de voir sur le projet.



###### Figure 33- Tableau de bord des membres du projet

La capture suivante montre le formulaire des Data managers qui leur permet d’ajouter les données qu’ils récoltent sur le terrain.

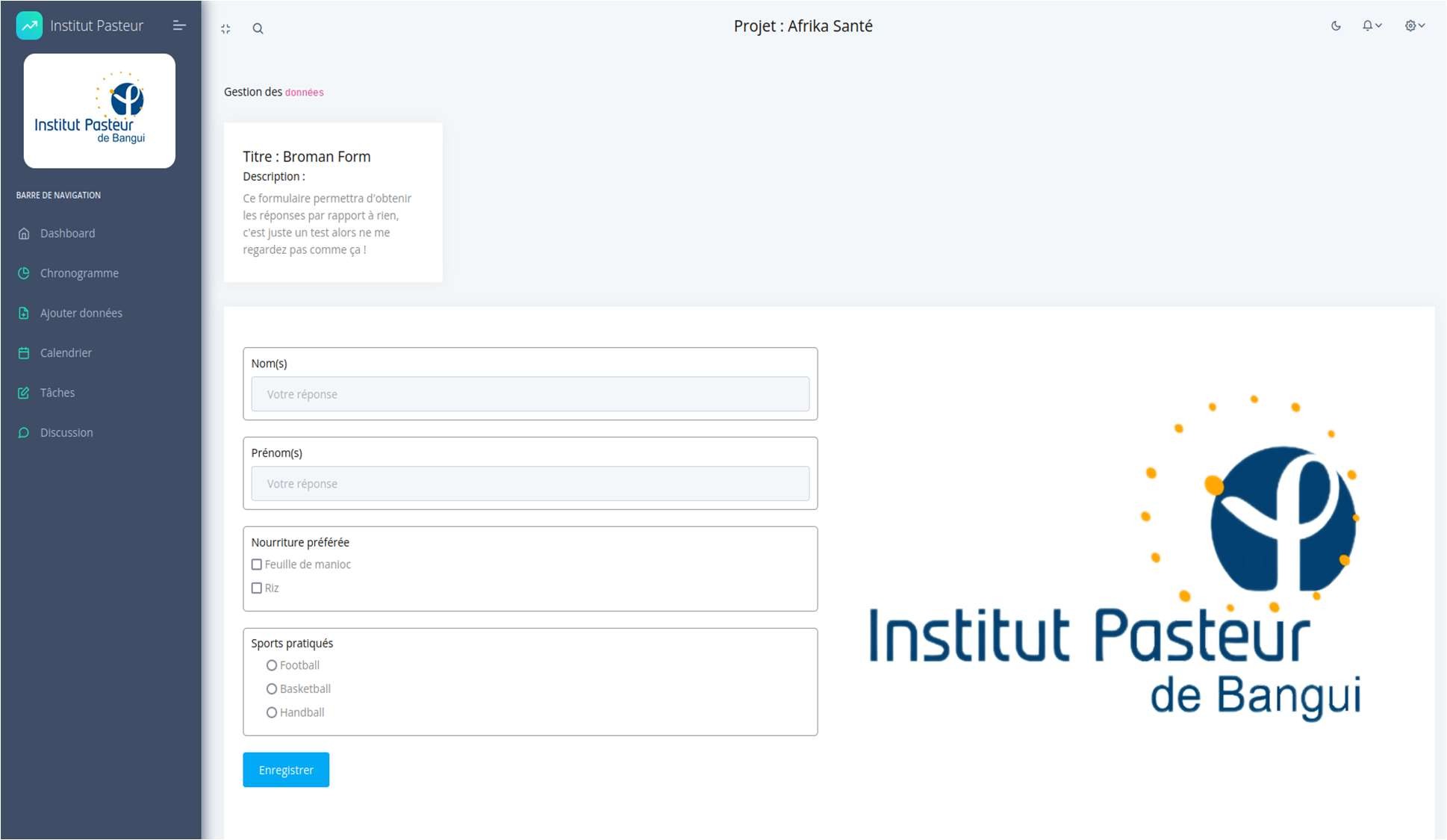


Figure 34- Formulaire du projet

## CONCLUSION

Au sein de l’Institut Pasteur, où la gestion de projet efficace est essentielle à la réussite des initiatives, nous avons abordé ce problème crucial dans un contexte de recherche et d’innovation. L’institut faisait face à l’utilisation d’outils rudimentaires, ce qui rendait la gestion des projets plus complexe et moins flexible.

Pour répondre au problème de l’utilisation des outils traditionnels, devenus aujourd’hui obsolètes, nous avons développé une application RESTFul de gestion de projet qui permet une gestion agile de tous les projets, en passant par la gestion des données récoltées à la gestion des ressources du projet. Grâce à cette solution, l’IPB pourra facilement gérer les données récoltées sur le terrain grâce à une interface alimentée par une API REST qui est uniforme à tous les Data managers ; à gérer efficacement les ressources de chaque projet et ainsi, optimiser la prise de décision.

En somme, ce travail n’a pas seulement été un aboutissement académique, mais aussi un pas en avant pour les applications de gestion de projet RESTFul et la gestion de projet efficiente et adaptée aux défis contemporains. Plus tard, cette solution devra continuer à être mise à jour et intégrera des fonctionnalités de communication en temps réel pour une meilleure collaboration, des modèles de calculs plus performants pour la statistique des données et aussi des modèles d’intelligence artificielle pour optimiser ses fonctionnalités.

# BIBLIOGRAPHIE

[1], Mémoire : Application Web pour la gestion des analyses de laboratoire, Ayoub GHANI

[2], Apprenez les bases du langage python, Open Classroom ;

[3], Débutez avec le Framework Django, Open Classroom ;

[4], Allez plus loin avec le Framework Django, Open Classroom ;

[5], Créez des pages web dynamiques avec Javascript, Open Classroom ;

[6], Mettez en place une API REST avec Django REST Framework, Open Classroom ;

[7], UML 2, Analyse et conception.

P a g e

# WEBOGRAPHIE

[1], Documentation Django ;

[2], Ecrivez votre première application Django : tutoriel ;

[3], Application Django réutilisables et plugin ;

[4], Ask codez : communauté de développeurs en ligne ;

[5], Comprendre le fonctionnement des projets et la mise en place des acteurs ; [6], Qu’est-ce qu’une API et comment l’utiliser ? Le guide pour tout comprendre ; [7], Une API, qu’est-ce que c’est ?

[8], Tout savoir sur les Framework ;

[9], Foundation Framework CSS ;

[10], Comment faire un chronogramme grâce à GitMind ;

[11], Comprendre le fonctionnement d’un site web statique et d’un site web dynamique ; [12], Présentation des 14 types et exemples des Diagrammes UML ;

P a g e

# ANNEXES

ANNEXES 1 : Organigramme de l’Institut Pasteur de Bangui

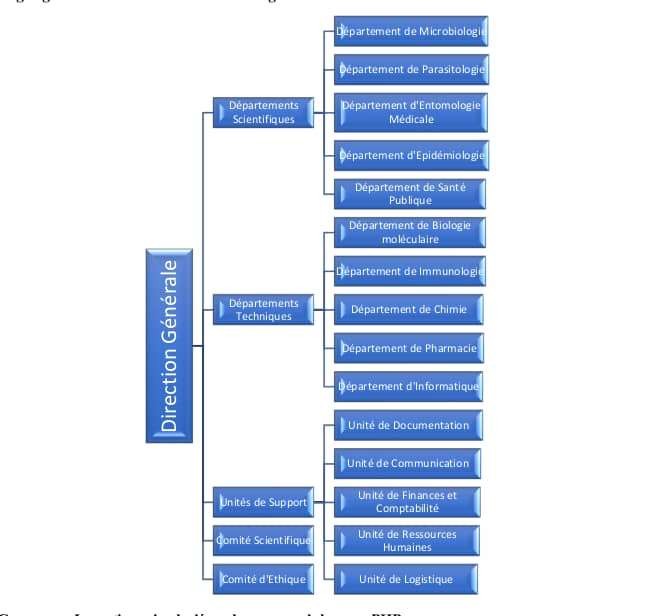


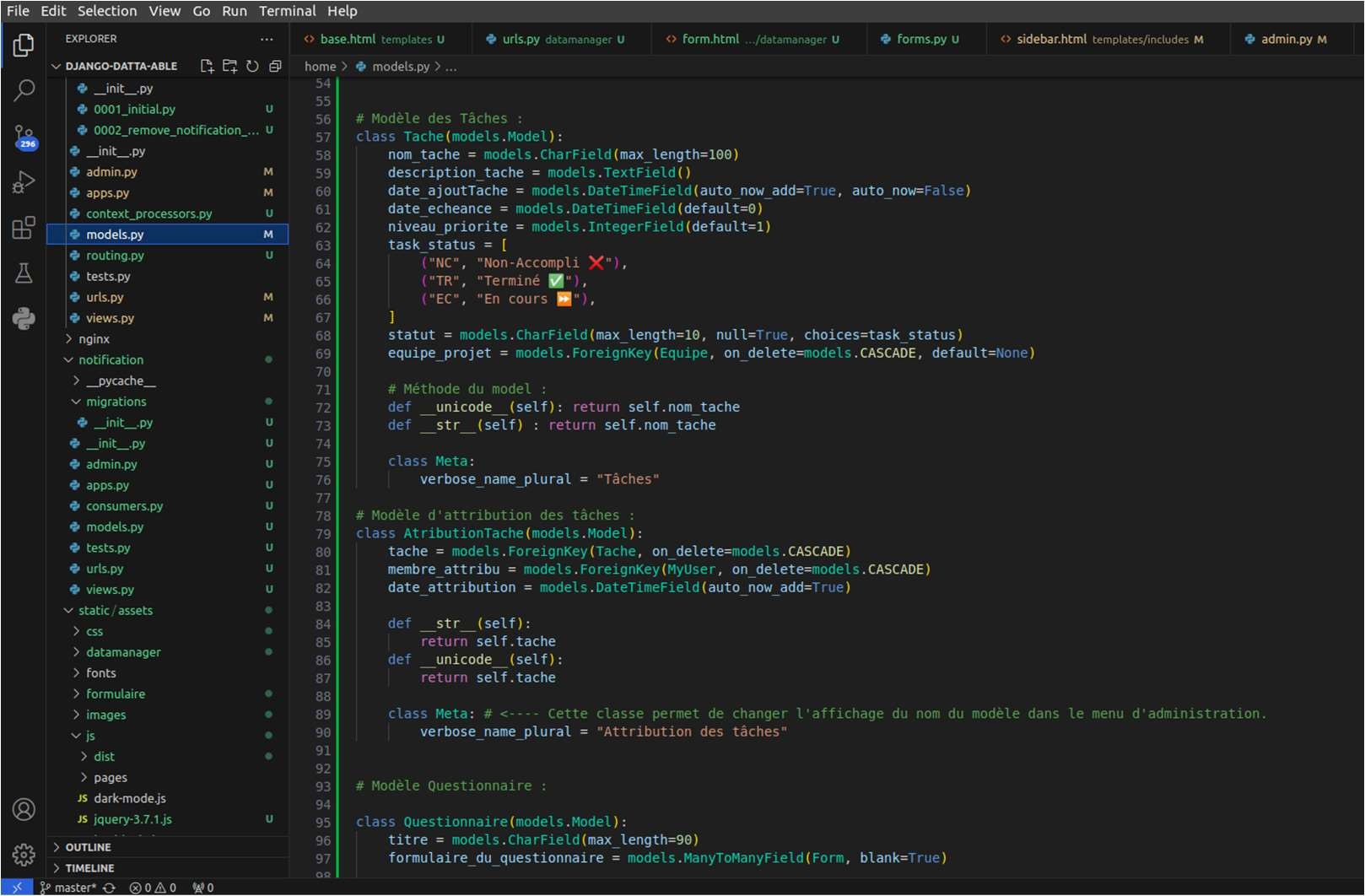
Figure 35- Organigramme de l'IPB

ANNEXE 2 : Installation de Django, des bibliothèques et de serveurs de messagerie

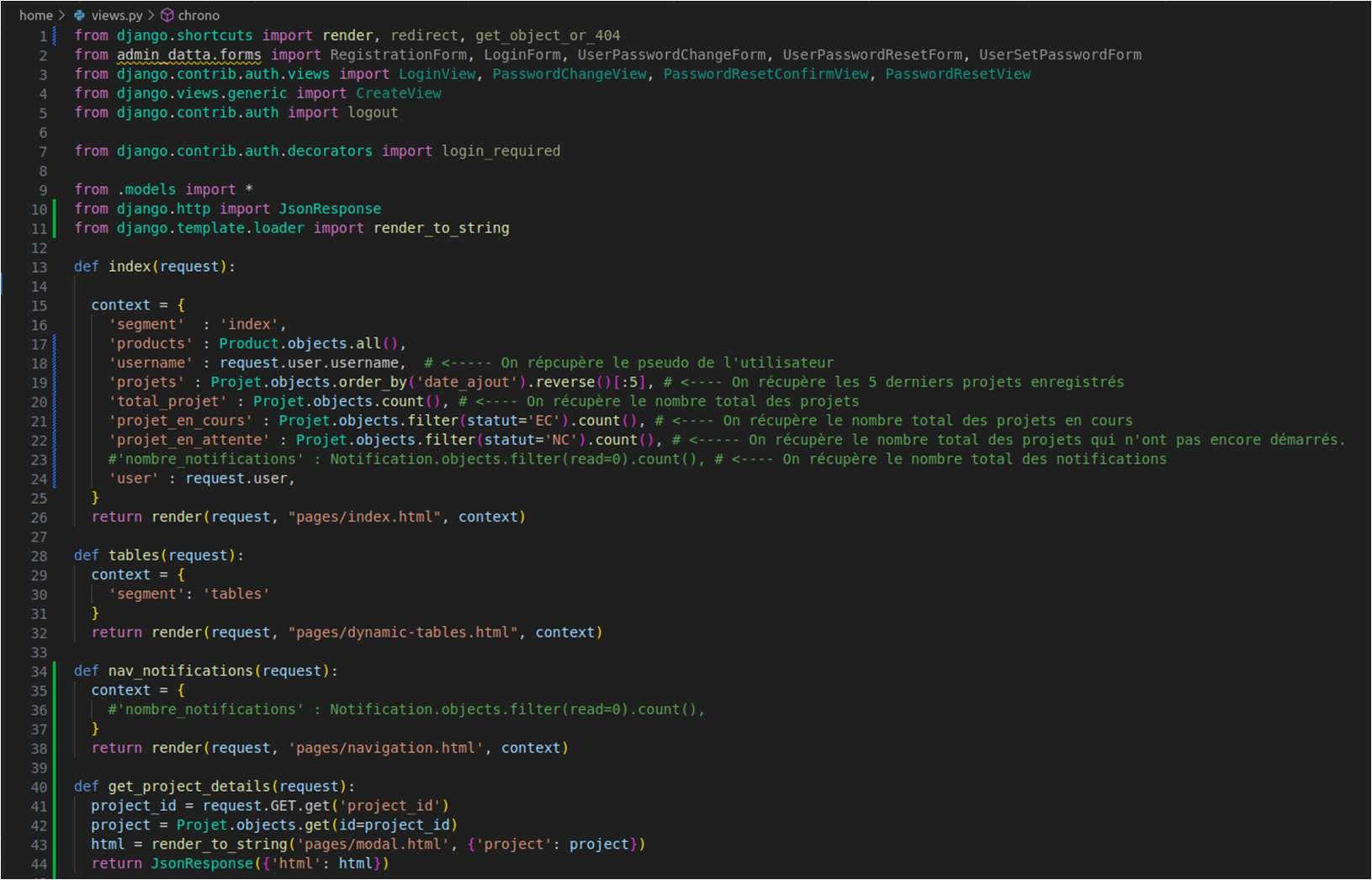
### Tableau 11- Installation de quelques bibliothèques

|  |  |
| --- | --- |
| Noms | Commandes d’installation |
| Django | # pip install django==4.2 |
| Celery | # pip install celery |
| Django channels | # pip install django-channels |
| Serveur Daphne | # pip install daphne |
| Serveur RabbitMQ | # sudo apt install rabbitmq-server -y |
| Django-admin-datta | # pip install django-admin-datta==1.0 |
| Whitenoise | # pip install whitenoise==6.5 |
| Gunicorn | # pip install gunicorn==21.2 |
| MySQL | # pip install mysqlclient==2.2.2 |

ANNEXE 3 : Code source des Modèles, Vues et Template



### Figure 36- Code source des modèles



### Figure 37- Code source des vues

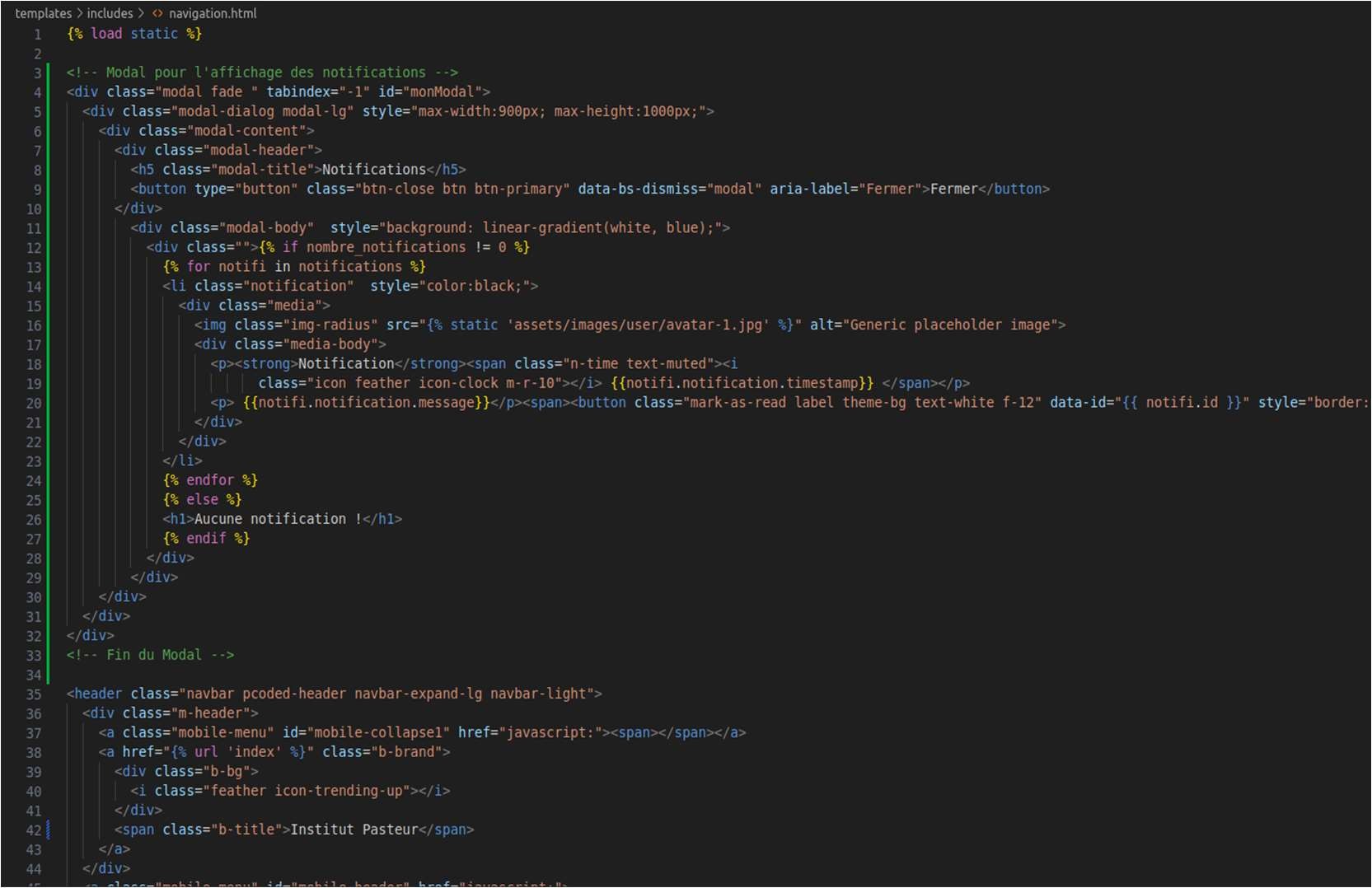


Figure 38- Code source des Template

# TABLES DES MATIERES

DÉDICACE ......................................................................................................................................... i

REMERCIEMENTS ......................................................................................................................... ii

AVANT-PROPOS .............................................................................................................................. iii

SOMMAIRE ...................................................................................................................................... iv

Liste des sigles et abréviations .......................................................................................................... v

Liste des tableaux .............................................................................................................................. vi

Liste des figures ................................................................................................................................ vii

INTRODUCTION GÉNÉRALE ...................................................................................................... 1

PARTIE 1 : PRÉSENTATION GÉNÉRALE .................................................................................. 1

CHAPITRE I : Présentation de la structure d’accueil ................................................................... 2

I-1. Historique................................................................................................................................. 2

I-2. Domaines d’activités ............................................................................................................... 2

I-3. Mission .................................................................................................................................... 3

I-4. Organigramme de l’Institut Pasteur de Bangui (Voir annexe) ........................................... 3

CHAPITRE II : Analyse de l’existant .............................................................................................. 4

II-1. Étude de l’existant ................................................................................................................. 4

II-2. Critique de l’existant ............................................................................................................. 4

II-3. Objectifs et propositions........................................................................................................ 5

II-4. Coût du projet ........................................................................................................................ 6

PARTIE 2 : ÉTUDES THÉORIQUES ............................................................................................. 7

Chapitre III : Étude des concepts et méthodes ................................................................................ 8

III-1. Présentation du Web ............................................................................................................ 8

III.1.1. Introduction au web ...................................................................................................... 8 III.1.2. Les composants et fonctionnement du web ................................................................. 8 a. Les sites web statiques ................................................................................................... 11

b. Les sites web dynamiques ............................................................................................. 12

III.1.3. Les avantages du Web ................................................................................................. 13

III-2. Les Framework .................................................................................................................. 14

III.2.1. Présentation ................................................................................................................. 14

III.2.2 Les avantages d’un Framework .................................................................................. 15

III.2.3. Les différents types de Framework............................................................................ 16

III.2.3.1. Les Framework Frontend ....................................................................................... 16

III.2.3.2. Les Framework Backends ...................................................................................... 16

III-3. Les modèles ......................................................................................................................... 17

P a g e

III.3.1. Le modèle MVC ........................................................................................................... 17

III.3.2. Le modèle MVT ........................................................................................................... 18

III-4. L’ORM ................................................................................................................................ 20

III.4.1. Les avantages des ORM .............................................................................................. 21

III-5. Les API ................................................................................................................................ 21

III.5.1. Présentation ................................................................................................................. 21

III.5.2. Les API REST .............................................................................................................. 23

III.5.2.1. Les principes de conception REST ........................................................................ 24

III.5.2.2. Fonctionnement des API REST.............................................................................. 25

III.5.2.3. Avantages des API REST ........................................................................................ 27

III-6. UML .................................................................................................................................... 28

III-7. Merise .................................................................................................................................. 29

Chapitre IV : Étude comparative et choix de solutions ................................................................ 31

IV.1. Études comparatives ........................................................................................................... 31

IV.1.1. Étude comparative UML/MERISE ............................................................................ 31

IV.1.2. Étude comparative Django/Flask ............................................................................... 32

IV.1.3. Étude comparative API REST/API SOAP ................................................................ 33

IV.2. Choix de solutions ............................................................................................................... 34

Chapitre V : Conception et Modélisation de l’application ........................................................... 36

V.1. Processus de développement ................................................................................................ 36

V.1.1. Processus itératif ............................................................................................................ 36

V.1.2. Processus centré sur l’architecture .............................................................................. 36

V.1.3. Processus piloté par le cas d’utilisation ....................................................................... 36

V.1.4. Processus orienté par la réduction des risques ........................................................... 37

V.2. Identification des acteurs ..................................................................................................... 37

V.2.1. L’Administrateur ........................................................................................................... 37

V.2.2. Le responsable des projets ............................................................................................ 38

V.2.3. Le chef de projet ............................................................................................................ 38

V.2.4. Le chef d’équipe ............................................................................................................. 38

V.2.5. Le membre (ou Data manager) .................................................................................... 39

V.2.6. Le Directeur Général ..................................................................................................... 39

V-3. Conception de l’application ................................................................................................. 39

V.3.1. Diagrammes de cas d’utilisation .................................................................................. 39

V.3.1.1. Diagramme de cas d’utilisation 1 : Administrateur ................................................ 40

V.3.1.2. Cas d’utilisation 2 : Responsable des projets ........................................................... 41

V.3.1.3. Cas d’utilisation 3 : Chef de projet ........................................................................... 42

V.3.1.4. Cas d’utilisation 4 : Gérer équipe (Le chef d’équipe) ............................................. 43

P a g e

V.3.1.5. Cas d’utilisation 5 : Data manager ........................................................................... 44

V.3.1.6. Cas d’utilisation 6 : Le Directeur Général ............................................................... 44

V.3.1.7. Quelques descriptions détaillées de cas d’utilisation globaux ................................ 45

V.3.2. Diagramme de classe ..................................................................................................... 47

V.3.3. Diagrammes de séquences ............................................................................................. 48

V.3.3.1. Digramme de séquence 1 : cas d’utilisation « S’authentifier » .................................... 48

V.3.3.2. Diagramme de séquence 2 : cas d’utilisation « Ajouter utilisateur » ..................... 49

V.3.3.3. Diagramme de séquence 3 : cas d’utilisation « Modifier équipe » ........................ 50

V.3.3.4. Diagramme de séquence 4 : cas d’utilisation « Recevoir notification » ................. 51

V.3.4. Diagrammes d’activités ................................................................................................. 52

PARTIE 3 : IMPLÉMENTATION ................................................................................................. 52

Chapitre VI : Mise en œuvre ........................................................................................................... 53

VI.1. Architecture de l’application ............................................................................................. 53

1. 1.1Architecture globale de l’application ........................................................................... 53

VI.1.2. Architecture logicielle de l’application ...................................................................... 55

* 1. Présentation des outils et technologies utilisées ............................................................... 56
     1. Outils et services utilisés .............................................................................................. 56
     2. Technologies utilisées ................................................................................................... 57
     3. Langages utilisés .......................................................................................................... 58
     4. Installations .................................................................................................................. 58
     5. Implémentation de la sécurité .................................................................................... 59
     6. Présentation des écrans de test de la solution ........................................................... 61
* Accès à l’application pour l’administrateur ................................................................... 61
* Accès à l’application pour les responsables .................................................................... 65
* Accès à l’application pour les Data managers (et membres) ......................................... 67

CONCLUSION................................................................................................................................. 70

BIBLIOGRAPHIE ............................................................................................................................. I

WEBOGRAPHIE.............................................................................................................................. II

ANNEXES........................................................................................................................................ III

TABLES DES MATIERES ........................................................................................................... VII

P a g e

1. Taux d’adoption obtenu sur le site mailchimp.com [↑](#footnote-ref-1)