UNIVERSITE SIDI MOHAMED BEN ABDELLAH
FACULTE DES SCIENCES ET TECHNIQUES DE FES

Projet de Fin d'Etudes

Licence Sciences et Techniques Génie Informatique

Application Web pour la gestion de la file d'attente

الوكالة المستقلة لتوزيع الماء والكهرباء بمكناس
RADEM

Lieu de stage : la RADEM

Réalisé par :

Zineb SABRI

Assaad Masrar

Encadré par :

Pr Mohammed TALIBI ALAOUI

Soutenu le 11/06/2019 devant le jury composé de :

Pr M TALIBI ALAOUI
Pr M. C. ABOUNAIMA
Pr A. ZARGHILI

Année Universitaire 2018-2019

REMERCIEMENTS

Nous tenons d'abord à exprimer notre reconnaissance à Monsieur Jamal Benfeddoul, le directeur général de la RADEM, qui nous a donné la chance d'y effectuer notre stage.

Nous adressons ensuite notre reconnaissance à notre encadrant de stage Monsieur Ismail BOUDJAJ pour son aide précieuse durant notre durée de stage, ses encouragements, sa disponibilité, son acharnement et sa présence dans les moments les plus délicats. Ses connaissances et son savoir faire nous ont permis d'acheminer de larges connaissances en un rien de temps.

Nous exprimons également nos plus sincères remerciements à Monsieur Mohammed TALIBI ALAOUI, notre encadrant pédagogique à la Faculté des Sciences et Techniques pour ses conseils prodigieux, son orientation, son soutien précieux, sa compréhension et son sens de l'écoute. Nous lui exprimons aussi notre gratitude pour sa bienveillance sur le bon avancement de notre travail.

Sans oublier nos sincères remerciements aux membres du jury qui ont consacré du temps à notre travail et accepté de le juger. En espérant que nous seront à la hauteur de leurs attentes.

Nous tenons à remercier toute personne ayant contribué au succès de notre stage et nous ayant aidé lors de la rédaction de ce rapport.

RESUME

Dans le présent rapport, nous présentons le travail accompli durant notre période de stage de fin d'étude qui s'est étalé sur une durée de deux mois, et ce du 05/04/2019 au 05/06/2019.

Nous avons eu la chance d'effectuer notre stage au sein de la « Régie Autonome de Distribution d'Eau et d'Electricité de Meknès », et de porter notre modeste contribution à un projet réel et qui sera très prochainement implémenter dans les agences de la RADEM.

Notre projet consiste en la création d'une application desktop qui permet l'authentification des agents de la RADEM, l'appel du client suivant, et la gestion de la file d'attente à travers la création d'une application web. Celle-ci affiche le numéro de ticket du client en cours de traitement pour chaque agent sous forme d'une interface dynamique, et comporte aussi une application administrateur qui affiche des statistiques hebdomadaire sur les performances des agences et la disponibilité des agents, des agences et des services.

Nous avons réalisé ce projet en utilisant HTML, CSS, les langages de programmation Java, JavaScript et le langage UML pour une conception orientée objet.

Sommaire

INTRODUCTION 8
CHAPITRE I CONTEXTE GENERAL DU PROJET 10
1. Lieu du stage 11
2. L'organigramme fonctionnel de la RADEM 12
3. Les départements de la RADEM : 12
4. Analyse du projet : 15
4.1. Problématique : 15
4.2. Travail demandé : 15
4.3. Nos tâches : 15
4.4. Planning du projet 16
CHAPITRE II ANALYSE ET CONCEPTION 19
1. Spécification des besoins : 20
1.1. Besoins fonctionnels : 20
1.2. Besoins techniques : 21
2. Conception : 22
2.1. Définition des acteurs : 22
2.2. Diagrammes : 23
2.2.1. Diagramme de packages : 23
2.2.2. Diagrammes de cas d'utilisation: 24
2.2.3. Diagrammes de séquences: 29
2.2.4. Diagrammes de classes: 32
CHAPITRE III REALISATIONS ET INTERFACES 35
1. Langages utilisés : 36
2. Application utile : 40
3. Interfaces : 41
3.1. Ecran d'accueil : 41
3.2. Application d'authentification : 49
3.3. L'application de l'administrateur : 51
CONCLUSION 53
WEBOGRAPHIE 54

LISTE DES FIGURES

Figure 1 : Logo de la RADEM 11
Figure 2 : organigramme de la RADEM 12
Figure 3 : Diagramme de Gant 17
Figure 4 : Diagramme de package 23
Figure 5: diagramme de cas d'utilisation – Serveur 24
Figure 6 : diagramme de cas d'utilisation – Ecran 25
Figure 7 : diagramme de cas d'utilisation – Agent 26
Figure 8 : prototype authentification 27
Figure 9 : prototype appel client suivant 27
Figure 10 : diagramme de cas d'utilisation – Administrateur 28
Figure 11: diagramme de séquence – Serveur 29
Figure 12: diagramme de séquence – Ecran 30
Figure 13: diagramme de séquence – Agent 31
Figure 14: diagramme de séquence – Administrateur 32
Figure 15: diagramme de classes 33
Figure 16 : Modèle non bloquant. 36
Figure 17 : Modèle bloquant. 36
Figure 18: Modèle bloquant vs modèle non bloquant. 36
Figure 19 : exemple - Express 36
Figure 20 : exemple - Axios 37
Figure 21 : Exemple – route 1 37
Figure 22 : Exemple – route 2 37
Figure 23: exemple – HTTP 38
Figure 24: exemple – Socket.io 38
Figure 25: exemple – Socket.io connexion 38
Figure 26: exemple – Socket.io update 38
Figure 27: exemple – Cookie 38
Figure 28: exemple – Cookie 1 39
Figure 31 : interface –accueil 1 41
Figure 32: interface –accueil 2 42
Figure 33: interface –accueil 3 42
Figure 34 : URL 1 43
Figure 35 : URL 2 43
Figure 36 : Code 1 43
Figure 37 : Liste agents 43
Figure 38 : Code 2 44
Figure 39 : Réponse – code 2 44
Figure 40 : Code 3 45
Figure 41 : Réponse – code 3 45
Figure 42 : Code 4 46

Figure 44 : Réponse 2 – code 4 ..... **Erreur ! Signet non défini.**
Figure 43 : Réponse 1 – code 4 ..... **Erreur ! Signet non défini.**
Figure 45 : Code 5 47
Figure 46 : Code 6 47
Figure 47 : Météo 48
Figure 48 : Code 7 48
Figure 49 : Code 8 49
Figure 50 : Code 9 49
Figure 51 : Interface – Authentification 1 49
Figure 52 : Interface – Authentification 2 50
Figure 53 : Interface – Appel client suivant 50
Figure 54 : Interface – Application admin 1 51
Figure 55 : Interface – Application admin 2 52

LISTE DES ACRONYMES

RADEM : Régie Autonome de Distribution d'Eau et d'Electricité de Meknès.

UML : Langage de Modélisation Unifié.

JS : JavaScript.

HTTP : HyperText Transfer Protocol « protocole de transfert hypertexte ».

CORS : Cross-Origin Resource Sharing « partage des ressources entre origines multiples ».

API : Interface de Programmation d'Application.

INTRODUCTION

De nos jours, nous sommes tous témoins du développement et de l'évolution dans tous les domaines et en particulier en informatique. Cette évolution n'a laissé personne indifférent et le monde commence à introduire l'informatique dans son quotidien que ce soit un individu, une société, une entreprise, ou un simple marchand...

La RADEM a pour mission première, depuis plus de 40 ans, d'assurer une alimentation en eau potable et en électricité à la population de la ville de Meknès, et de collecter et traiter les eaux usées, domestiques et industrielles. Mais elle va au-delà de sa mission principale, en développant sans cesse une véritable relation de proximité avec ses clients, afin de leur rapporter, au quotidien, le meilleur service se reposant sur les technologies les plus performantes et sur l'informatique de pointe.

C'est dans ce cadre que s'intègre notre projet de fin d'étude qui a pour objectif de développer trois applications pour la gestion des 12 agences de la RADEM et de les y implémenter par la suite. Les trois applications sont comme suit :

- *Application desktop développée avec JAVA permettant l'authentification des agents et l'appel du client suivant*
- Application Web, développée avec Node.js et JQuery, qui a pour objectif :
  - *L'affichage des informations concernant les stations (nouveauté, nom de l'agent),*
  - *L'affichage des numéros de clients en cours de traitement,*
  - *Le défilement de l'actualité concernant les agences de la RADEM.*

  Et ce, pour que le client soit plus à l'aise et mieux informer. Et donc pour une meilleure gestion de la file d'attente des clients.
- *Application Web, développée avec JavaScript, permettant aux responsables d'agences d'avoir un aperçu des statistiques des performances des agences, la disponibilité des agences, des agents et des services*

Le projet a été réalisé au sein du service informatique de la RADEM et sera testé et déployé dans les agences prochainement. Un projet à traves lequel le chef du service informatique de la RADEM compte conquérir le marché en le mettant en vente.

Le présent rapport est un bref aperçu de notre expérience durant ce stage et a pour but l'éclaircissement de notre contribution dans ce projet. Il se structure sur trois chapitres ; le premier présente le contexte général du projet et une vue sur le travail demandé. Le deuxième contient une spécification des besoins de l'application illustré par des diagrammes

du langage de modélisation UML. Et le dernier comporte des détails de la réalisation du travail. Ce rapport est clôturé par une conclusion générale contenant l'évaluation du travail réalisé au sein de l'entreprise.

CHAPITRE I
CONTEXTE GENERAL DU PROJET

Lieu du stage

La RADEM, dont le logo est représenté sur la figure 1, a été créée par délibération du conseil municipal de la ville de Meknès en date de 1969 en vertu de dahir n°1-59-315 du 28 hijria 1379 (23 juin 1960) relatif a l'organisation communale sous la dénomination de : « REGIE AUTONOME DE DISTRIBUTION D'EAU ET D'ELECTRICITE ». C'est un établissement semi-publique à caractère industriel et commercial doté de sa personnalité civile et de l'autonomie financière. Il a succédé à la société Marocaine de Distribution (SMD) qui assurait jusqu'alors la gestion de distribution.

Figure 1 : Logo de la RADEM

La régie est chargée d'assurer à l'intérieur du périmètre de distribution le service public de distribution d'Eau et d'Electricité. La régie est également chargée de l'exploitation des captages et adduction d'Eau.

La RADEM assure une distribution efficace et un service de qualité en matière d'eau et d'électricité, c'est pourquoi des efforts considérables sont déployés pour moderniser ses réseaux de distribution d'une manière scientifique et progressive afin de mieux préparer l'étape de l'intercommunalité et répondre aux besoins futurs.

L'organigramme fonctionnel de la RADEM

La structure de la R.A.D.E.M est présentée par l'organigramme suivant :

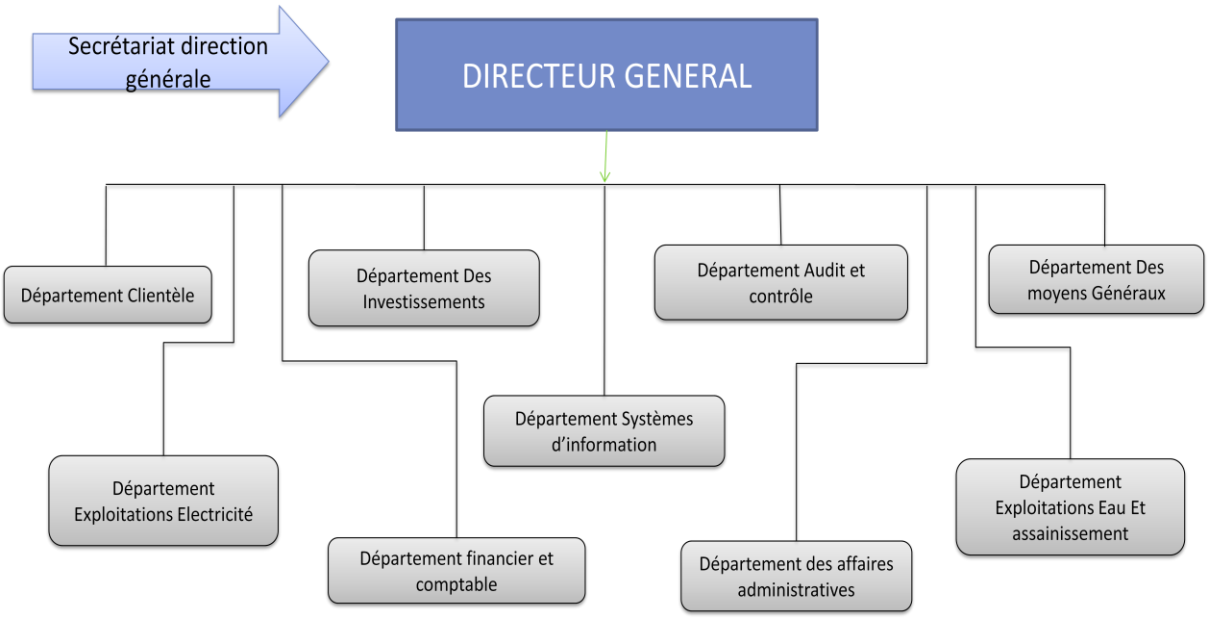

Figure 2 : organigramme de la RADEM

Les départements de la RADEM :

La RADEM comporte 9 départements et chaque département contient des divisions et services comme nous le montrons en ce qui suit :

- Département exploitation eau et assainissement :

Le département exploitation eau et assainissement est composé de trois divisions :

- Division contrôle de la qualité des eaux
- Division conduite des réseaux eau et assainissement
- Division maintenance et entretien des équipements

Ce département est principalement chargé de :

- La surveillance permanente de la qualité d'eau potable distribuée.
- L'exploitation et la gestion des réservoirs et des stocks d'eau potable.
- La réalisation des projets de renouvellement, de réhabilitation, de normalisation et de mise à niveau des réseaux d'eau et d'assainissement.

■ Département des investissements :

Le département des investissements est composé de deux divisions :

- Division études infrastructure et lotissement
- Division travaux infrastructure et lotissement

Les divisions ont plusieurs missions parmi lesquelles les études concernant le renforcement, la réhabilitation, le renouvellement ou l'extension du réseau ainsi que les études et les travaux de branchement et leur suivi.

■ Département des moyens généraux :

Le département des moyens généraux et composé de deux divisions :

- Division logistique
- Division approvisionnement

Ces divisions s'occupent essentiellement de l'approvisionnement de la régie suivant les besoins des différents départements et le suivi des travaux d'équipement des postes.

■ Département financier et comptable :

Le département financier et comptable est composé de 3 divisions :

- Division comptable chargée de la comptabilité auxiliaire, la comptabilité générale et la comptabilité analytique.
- Division financière chargée de la gestion financière du budget et du patrimoine.
- Division pairie chargée de la validation des dépenses.

■ Département des affaires administratives :

Ce département est composé de deux services :

- Service gestion des ressources humaines.
- Service affaires juridiques.

■ Département audit et contrôle interne :

Ce département est un service de contrôle et d'inspection de tous les services de la Régie. Il a pour rôle de faire des recommandations à la direction.

Il est composé de deux divisions :

- Division contrôle de gestion.

- Division audit interne.

■ Département système d'information :

Ce département a pour mission la gestion du matériel et des systèmes d'information dont dispose la régie et prend en charge leur maintenance et leur suivi.

Ses principaux services sont :

- Service réseau et production.
- Service étude et développement.
- Service systèmes d'information et sécurité.

■ Département clientèle et marketing :

Ce département se compose de 5 divisions :

- Division étude et travaux client chargée des études de branchement et vérification ainsi que les travaux de branchement.
- Division technico-domiciliaire chargée du mouvement compteurs, codification et la relance et coercition.
- Division clients particuliers chargée de la relation clientèle recouvrement et agence commerciale et règlement des comptes.
- Division grands clients chargée du suivi et recouvrement des administrations et grands clients et de la relation promoteurs/grands clients.
- Division support et développement chargée de l'administration de la base de données clientèle, statistiques et reportent ainsi que la relève et le traitement des éditions produits.

■ Département exploitation électricité :

Ce département est constitué de deux divisions :

- Division conduite des réseaux MT et BT.
- Division entretien et maintenance des équipements.

Analyse du projet :

1.1. Problématique :

Les délais d'attente représentent la principale source d'insatisfaction. Le manque de ressources professionnelles peut être la cause ou la source d'insatisfaction ressentis chez les clients, ce sentiment leur donne l'impression d'être mal servis et crée un sentiment d'inconfort voire d'insécurité chez eux, ce qui peut nuire la qualité du service.
D'où l'importance d'aborder le problème de la file d'attente, et surtout le temps d'attente.

1.2. Travail demandé :

Inspiré par la gestion des agences en Europe, le responsable du département informatique de la RADEM a voulu développer la gestion de leurs 12 agences.

L'idée est de créer:

- Une application desktop qui, après authentification, permettra à l'agent de faire appel au client suivant
- Une application web qui sur un écran permet le défilement des informations concernant les agents, la file d'attente et l'actualité des agences sur une page web dynamique et identique dans toutes les agences.

1.3. Nos tâches :

Vu que le travail a déjà été abordé à notre arrivée, notre encadrant de stage nous a chargé de tâches au fur et à mesure de l'avancement du projet. Ces tâches sont les suivantes :

➔ La création d'une application desktop pour :
- L'authentification des agents,
- L'appel du client suivant.

➔ La création d'une application web dynamique pour :
- Le défilement des informations concernant les agents, la file d'attente et l'agence.

➔ La création d'une application web (Template) qui permet aux administrateurs d'avoir accès aux performances hebdomadaires des agences, les agences disponibles, le taux de traitements, les services rendus ....

Les fonctions et les variables sont prédéfinis dans le serveur de la RADEM sous forme de web services auxquels nous avons fait appel et redéfini.

1.4. Planning du projet

Nous avons donc établi notre plan en dépend de la succession des tâches. Le tableau suivant représente notre plan de travail :

Tâches	*Etapes*	*Début*	*Durée*	*Fin*
Gestion file d'attente	*Documentation*	*5 avril 2019*	*16*	*21 avril 2019*
	Conception	*22 avril 2019*	*6*	*28 avril 2019*
	Programmation	*29 avril 2019*	*14*	*13 mai 2019*
Application desktop	*Documentation*	*14 mai 2019*	*1*	*15 mai 2019*
	Conception	*14 mai 2019*	*1*	*15 mai 2019*
	Programmation	*16 mai 2019*	*4*	*20 mai 2019*
Application admin	*Documentation*	*21 mai 2019*	*1*	*22 mai 2019*
	conception	*22 mai 2019*	*1*	*23 mai 2019*
	Programmation	*24 mai 2019*	*4*	*28 mai 2019*
Rapport		*29 mai 2019*	*8*	*6 juin 2019*
Vidéo	*prise d'écran*	*1 juin 2019*	*1*	*2 juin 2019*
	montage	*2 juin 2019*	*2*	*4 juin 2019*

Nous avons également effectué une représentation graphique de notre plan sous forme du diagramme de Gantt qui permet de visualiser dans le temps les diverses tâches et étapes composant notre projet :

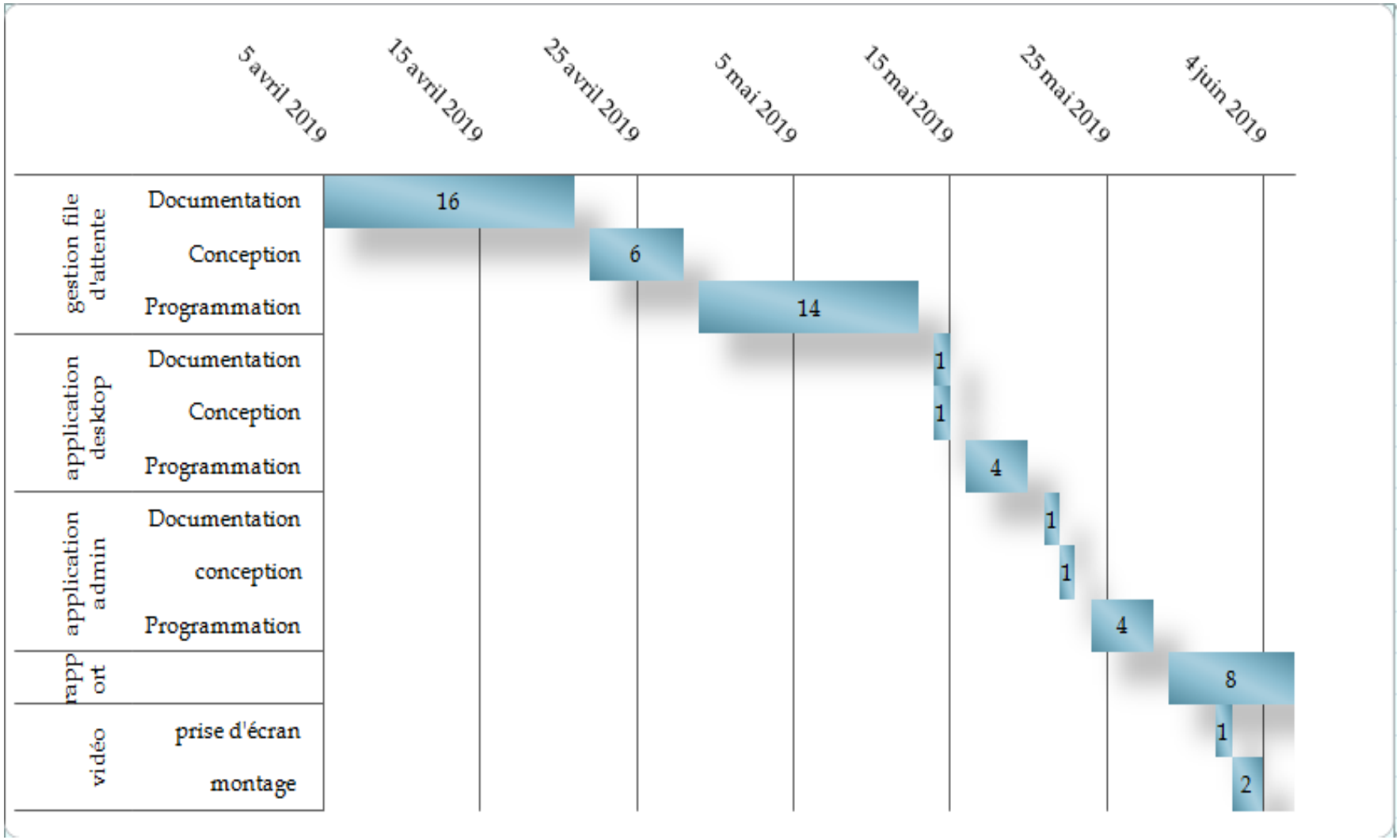

Figure 3 : Diagramme de Gant

Dans ce chapitre, nous avons présenté notre lieu de stage, son diagramme et nous avons défini ses départements. Pour passer par la suite à l'élaboration de la problématique et la présentation du travail demandé ainsi que les tâches qui nous ont été accordé.

CHAPITRE II
ANALYSE ET CONCEPTION

Spécification des besoins :

Cette phase consiste à comprendre le contexte du projet. L'application dont la RADEM veut se doter doit être évolutif et offrir les informations nécessaires à temps réel.

Pour ceci, le travail à réaliser doit satisfaire les exigences de la totalité des utilisateurs. Nous présentons dans ce qui suit tous les besoins fonctionnels classés par acteur ainsi que les besoins non fonctionnels communs à tous ces acteurs.

1.1. Besoins fonctionnels :

Un acteur est une personne, un matériel ou un logiciel qui interagit avec le système dans le but de réaliser une plus value. Les acteurs en interaction avec notre système sont :

- Le serveur
- L'écran
- L'agent
- L'administrateur

- Le serveur contient les fonctions que nous avons redéfinies. Son rôle est de récupérer les informations nécessaires de la base de données:

  - ➔ Les informations des agences,
  - ➔ Les informations des agents,
  - ➔ La file d'attente : l'appel du client suivant par rapport au service qu'il souhaite avoir,
  - ➔ L'actualité des agences.

L'écran doit faire le défilement des informations récupérées de la base de données en temps réel à savoir :

- ➔ Les informations des agences et des agents,
- ➔ Le numéro du client suivant pour chaque agent,
- ➔ L'actualité concernant les agences de la RADEM,

- Ainsi que d'autres informations à savoir :

  - ➔ La météo,

- La date.

- L'agent doit pouvoir :
  - S'authentifier,
  - Faire appel au client suivant.

- L'administrateur doit avoir accès aux informations des agences à savoir :
  - La performance des agences,
  - Le nombre d'agents, d'agences et les services disponibles,
  - Le temps moyen de travail des agents.

Ces informations doivent se modifient hebdomadairement.

1.2. Besoins techniques :

Ce sont des exigences qui ne concernent pas spécifiquement le comportement d'un système, mais qui identifient ses contraintes internes et externes. Les principaux besoin non fonctionnels de notre application se résument dans les points suivants :

- La sécurité : l'application doit respecter la confidentialité des données, et exiger une limitation d'accès des différents utilisateurs selon leurs rôles
- La performance : Temps de réponse : le chargement de l'application, l'ouverture d'écran et les délais de rafraîchissement ne doivent pas être longs.
- La compatibilité : l'interface doit être compatible dans les écrans de toutes les agences de la RADEM.
- L'ergonomie : l'application doit offrir des interfaces conviviales, simples et faciles à utiliser.
- Le code doit être clair afin de permettre de futures améliorations.

Conception :

1.3. Définition des acteurs :

Un acteur est l'idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec le système.

Dans le cadre de notre analyse, le système nécessite l'intervention acteurs suivant :

- **Le serveur :** acteur secondaire, c'est le système qui contient les fonctions et qui récupère les informations à afficher sur l'écran des agences de la base de données de la RADEM.
- **L'écran :** acteur secondaire, il représente l'outil sur lequel serons affichées les informations des agences, des agents et de la file d'attente.
- **L'agent :** l'employé de l'agence, son rôle par rapport à notre application est de s'authentifier et de faire appel au client suivant.
- **L'administrateur :** tout employé ayant le pouvoir d'accès aux informations confidentielles du serveur.

1.4. Diagrammes :

1.4.1. Diagramme de packages :

Le diagramme de paquetage est un diagramme UML qui fournit une représentation graphique de haut niveau de l'organisation de votre application, et vous aide à identifier les liens de généralisation et de dépendance entre les packages.

Dans notre cas, les packages intéressants sont : serveur, client (l'écran), agent et administrateur.

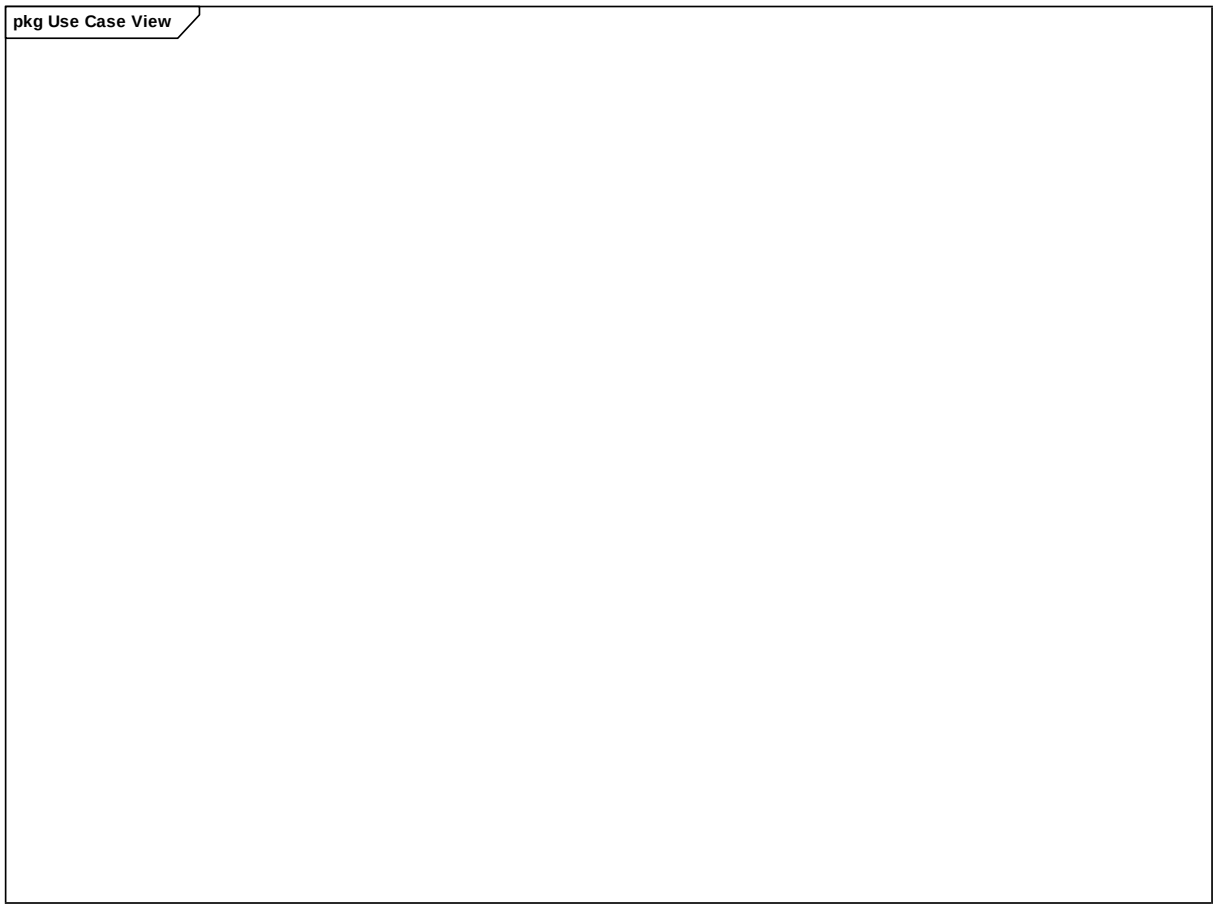

Figure 4 : Diagramme de package

1.4.2. Diagrammes de cas d'utilisation:

Les diagrammes de cas d'utilisation sont des diagrammes UML utilisés pour donner une vision globale du comportement fonctionnel d'un système logiciel.

Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Serveur :

Le serveur accède à la base de données d'où il récupère toutes les informations nécessaire.

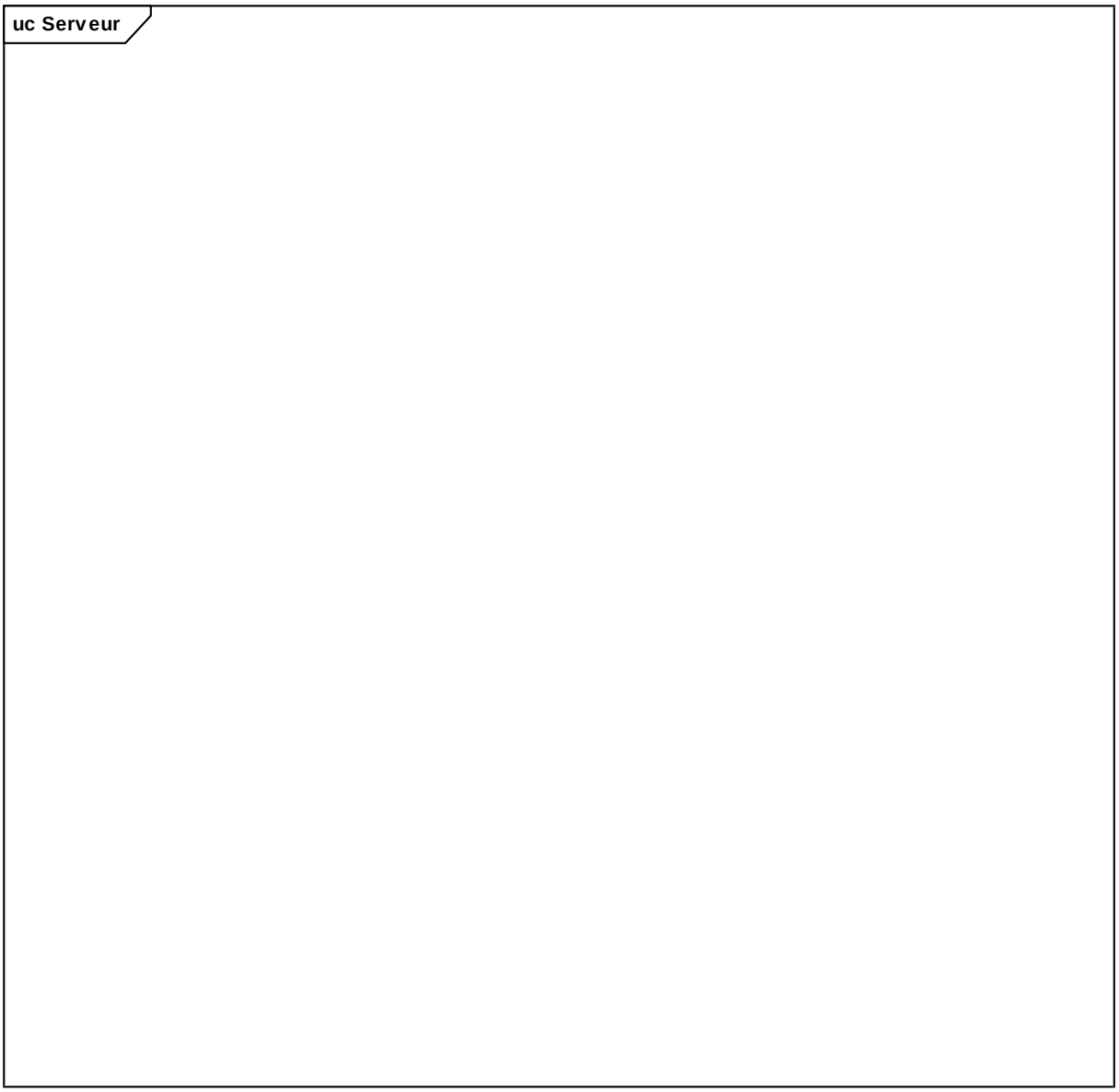

Figure 5: diagramme de cas d'utilisation – Serveur

La gestion de la file d'attente de fait grâce à deux web services :

Le premier permet l'appel au client suivant par rapport à un service donné.

Supposons maintenant qu'un client souhaite faire une déclaration et payer son abonnement, il n'aura besoin de tirer qu'un seul ticket qui lui permettra d'accomplir ces deux services même si cela doit être fait dans deux stations (guichets) différentes. Quand le client aurait fini sa déclaration, il serait appelé automatiquement dans le deuxième guichet même si son numéro de ticket est dépassé.

Ecran :

Sur l'écran d'accueil de chaque agence seront affichées les informations qui lui correspondent, comme nous le montre le diagramme suivant :

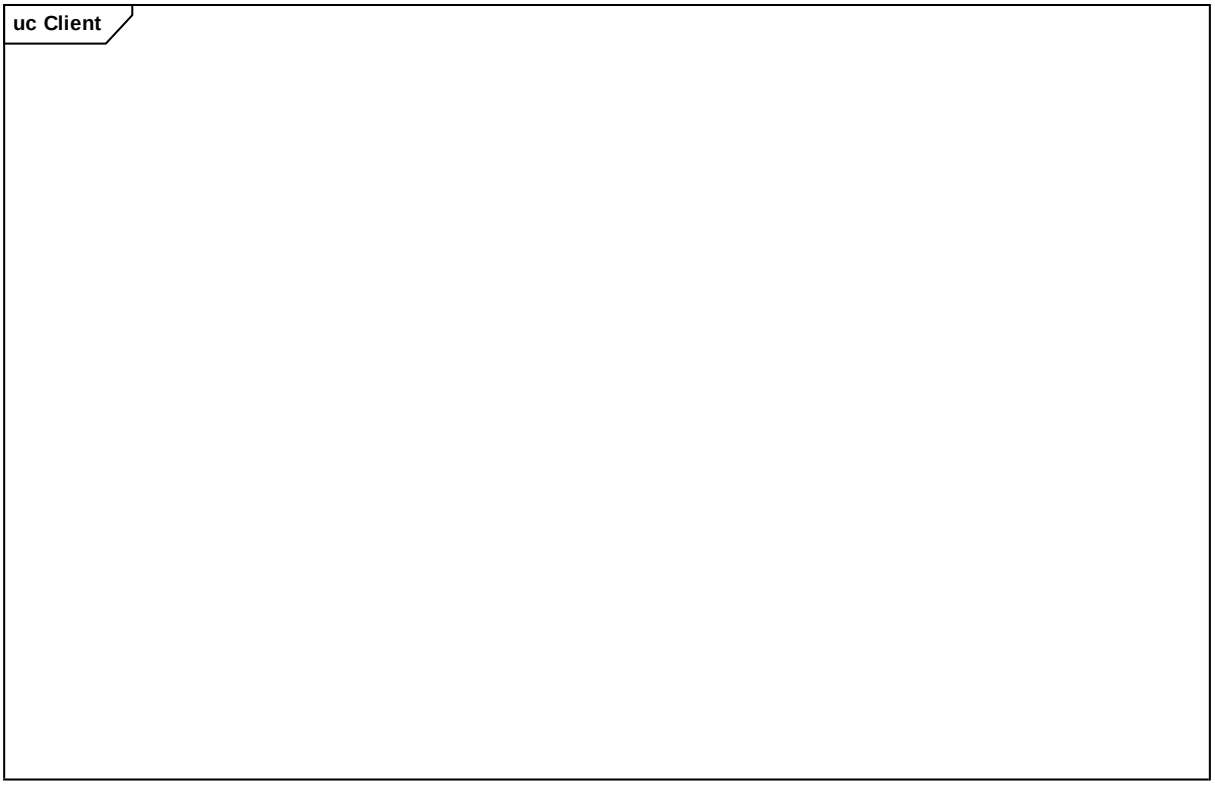

Figure 6 : diagramme de cas d'utilisation – Ecran

- **Agent :**

L'application desktop qu'utilisera l'agent lui permettra d'appeler le client suivant après qu'il s'authentifie.

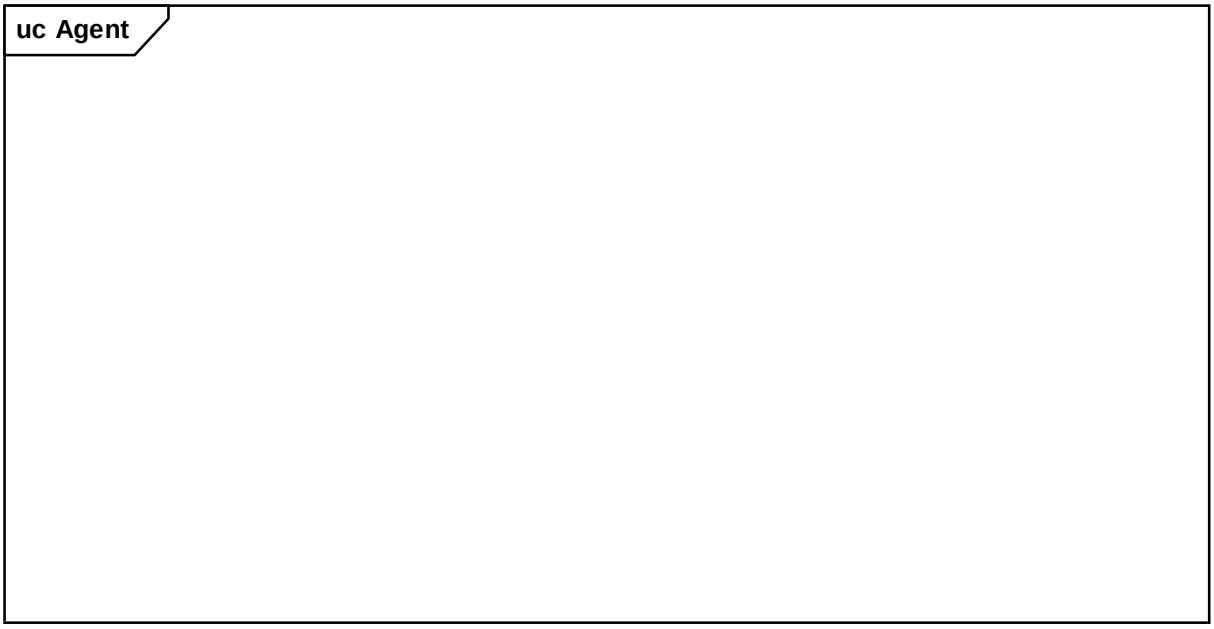

Figure 7 : diagramme de cas d'utilisation – Agent

Prototypes de cas d'utilisations

- **Authentification :**

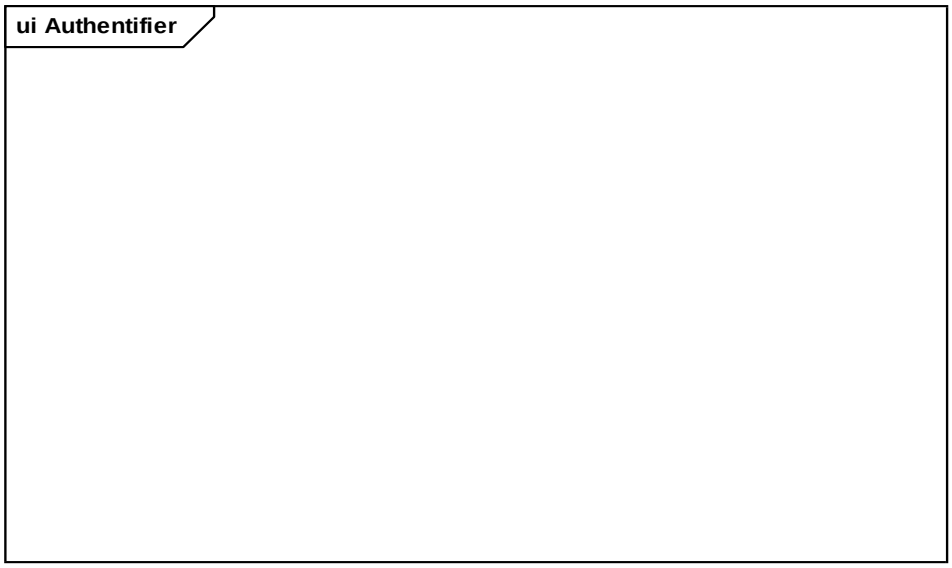

Figure 8 : prototype authentification

- **Appel client suivant :**

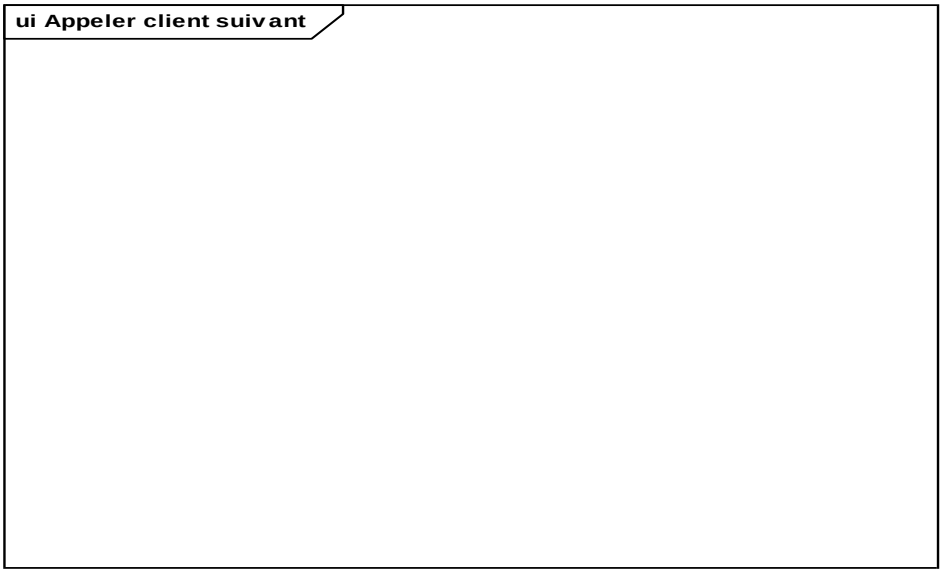

Figure 9 : prototype appel client suivant

- **Administrateur :**

A travers l'application de l'administrateur, ce dernier aura accès directe aux informations hebdomadaires de toutes les agences de la RADEM. Il aura également accès à un calendrier qu'il peut gérer.

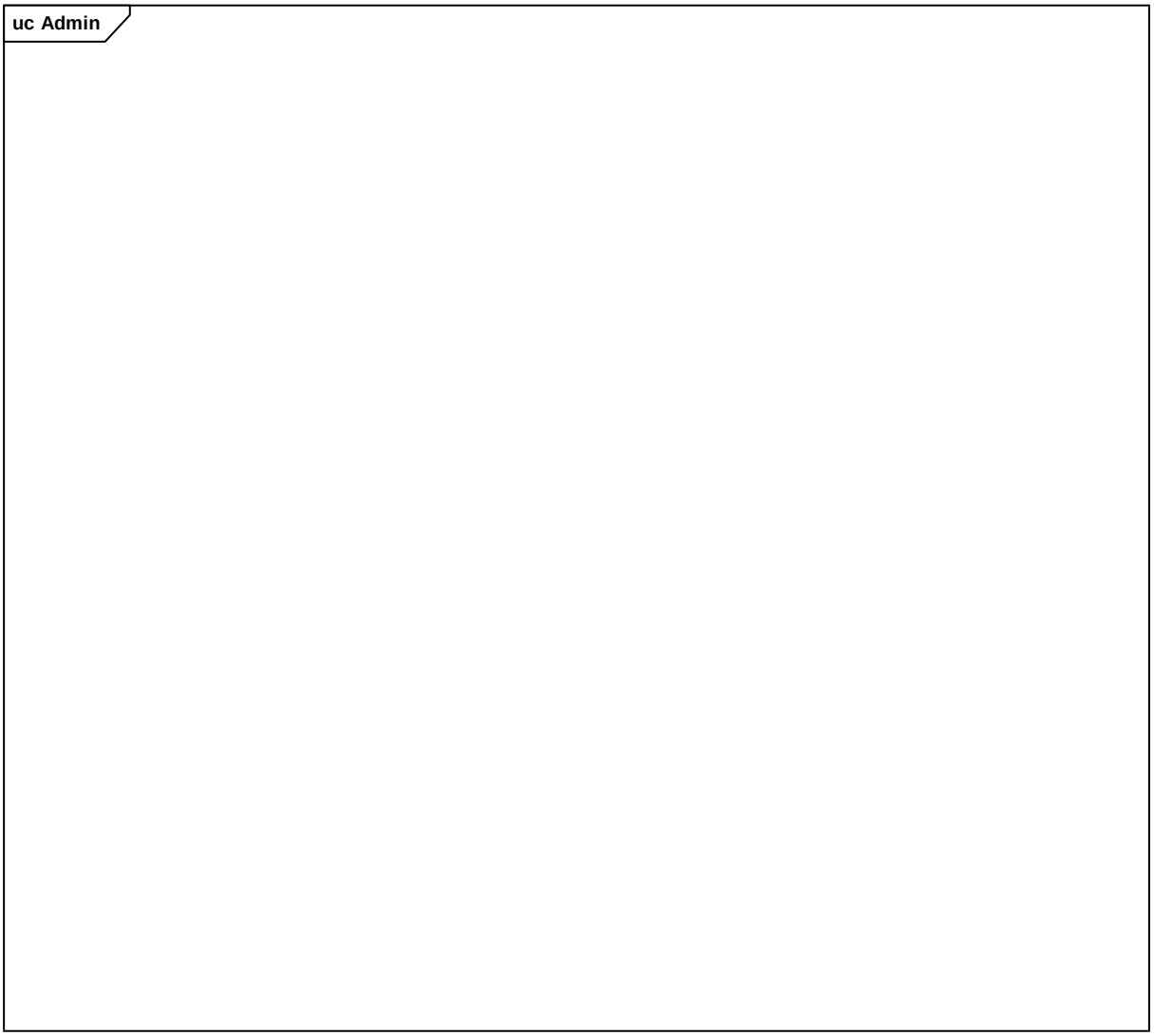

Figure 10 : diagramme de cas d'utilisation – Administrateur

1.4.3. Diagrammes de séquences:

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation Unified Modeling Language.

- **Serveur :**

Ce diagramme nous présente l'interaction entre le serveur et la base de données de la RADEM, et montre la chronologie entre la récupération des informations et leur affichage sur l'écran.

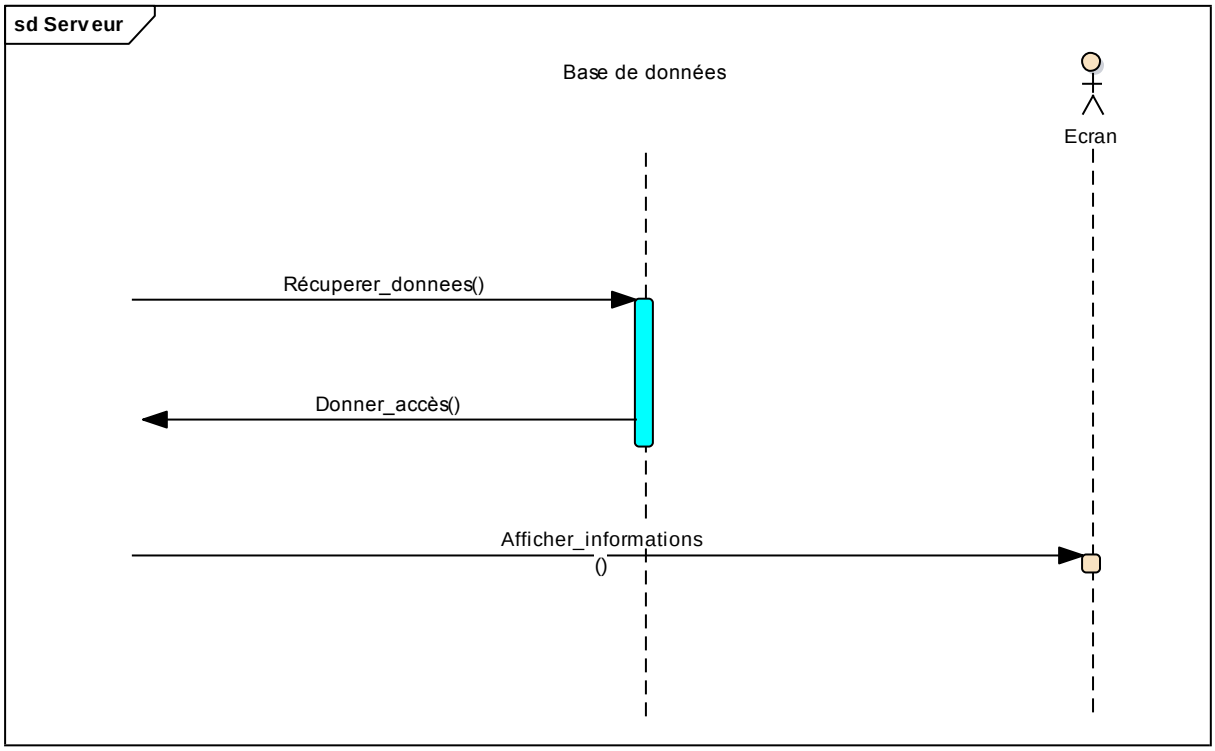

Figure 11: diagramme de séquence – Serveur

- **Client :**

Ce diagramme montre comment les informations sont affichées sur l'écran.

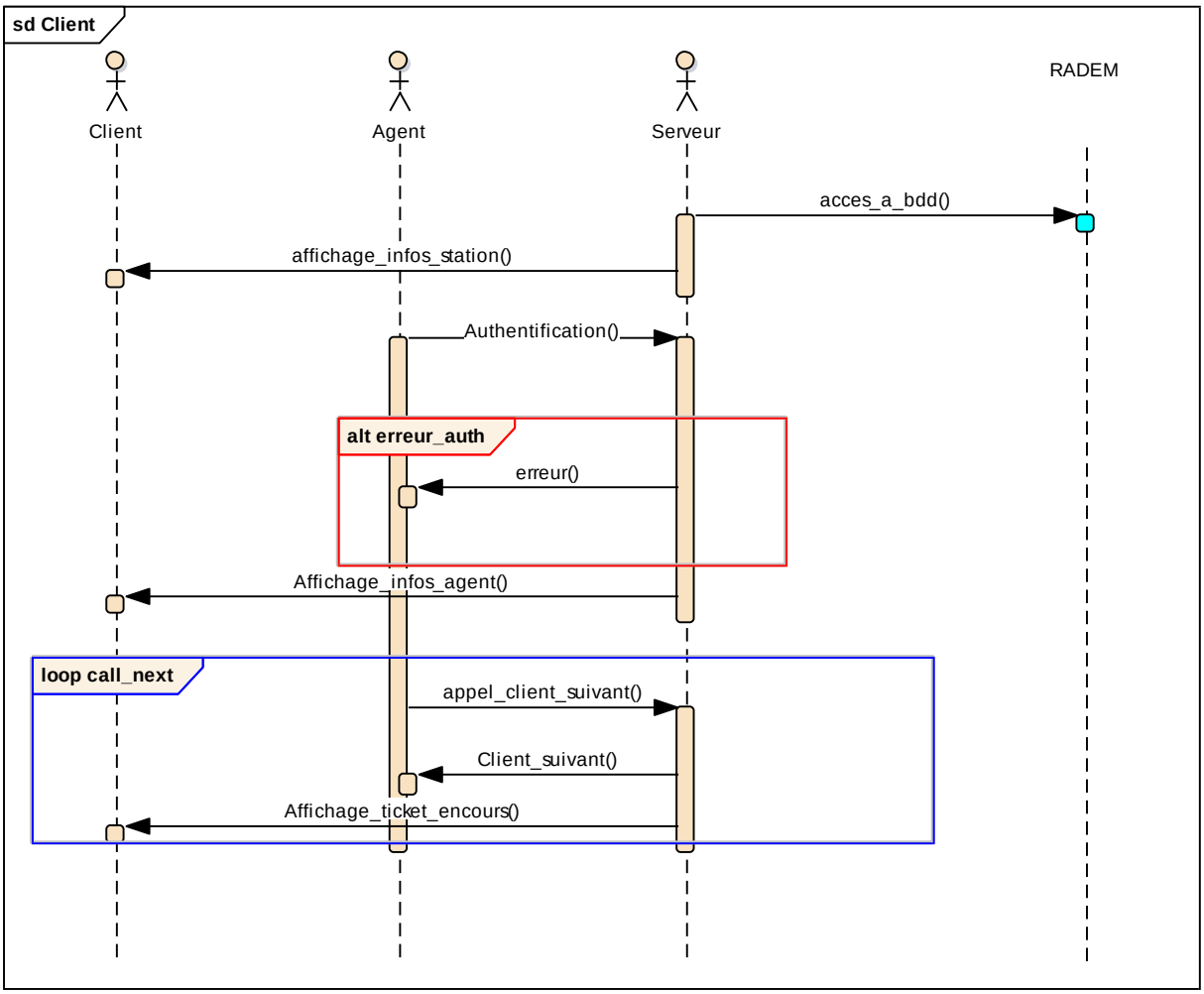

Figure 12: diagramme de séquence – Ecran

- **Agent :**

Dans ce diagramme, nous présentons la gestion de l'authentification des agents.

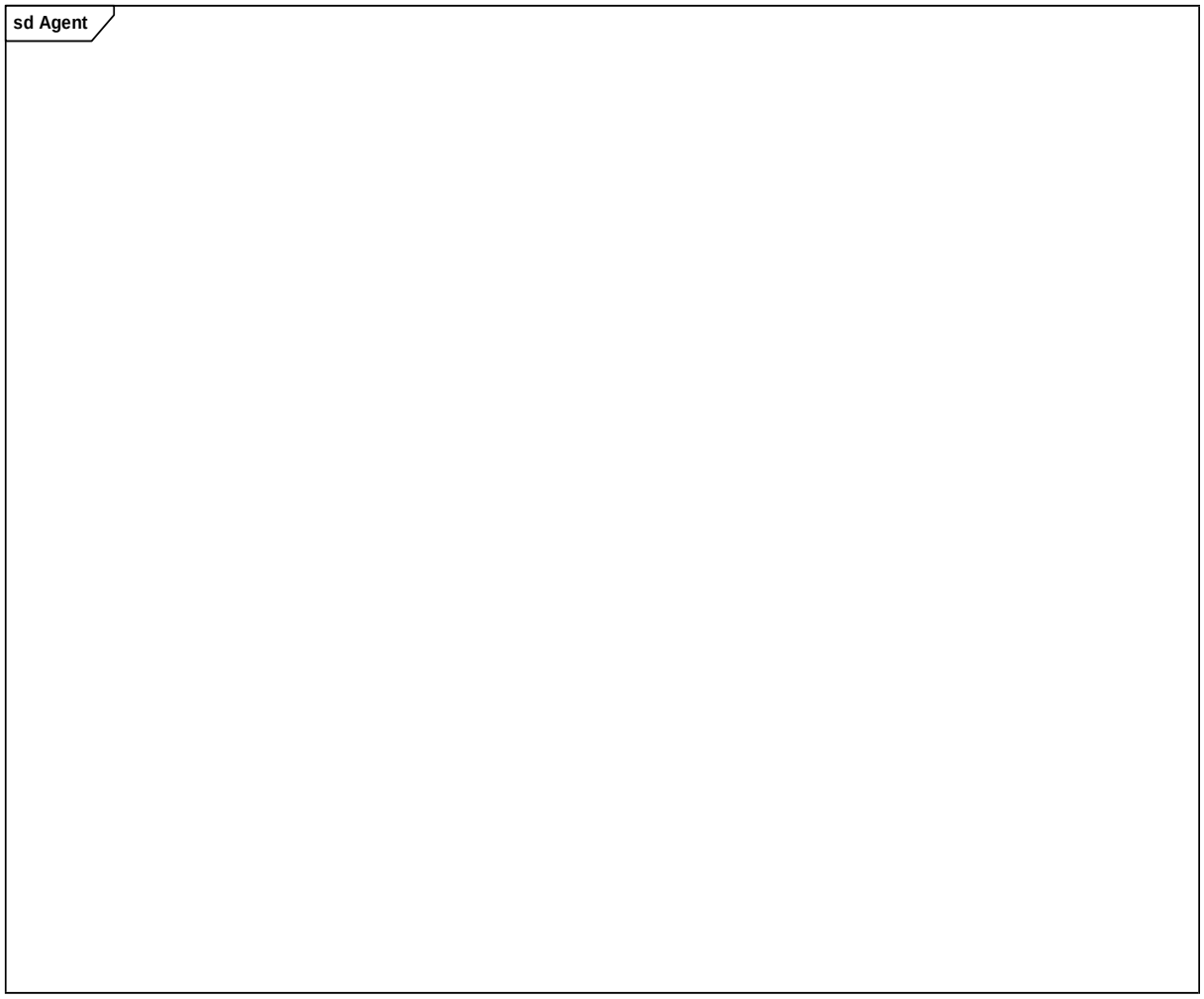

Figure 13: diagramme de séquence – Agent

- **Administrateur :**

Nous montrons par ce diagramme les fonctionnalités qu'aura un administrateur dans l'application administrateur.

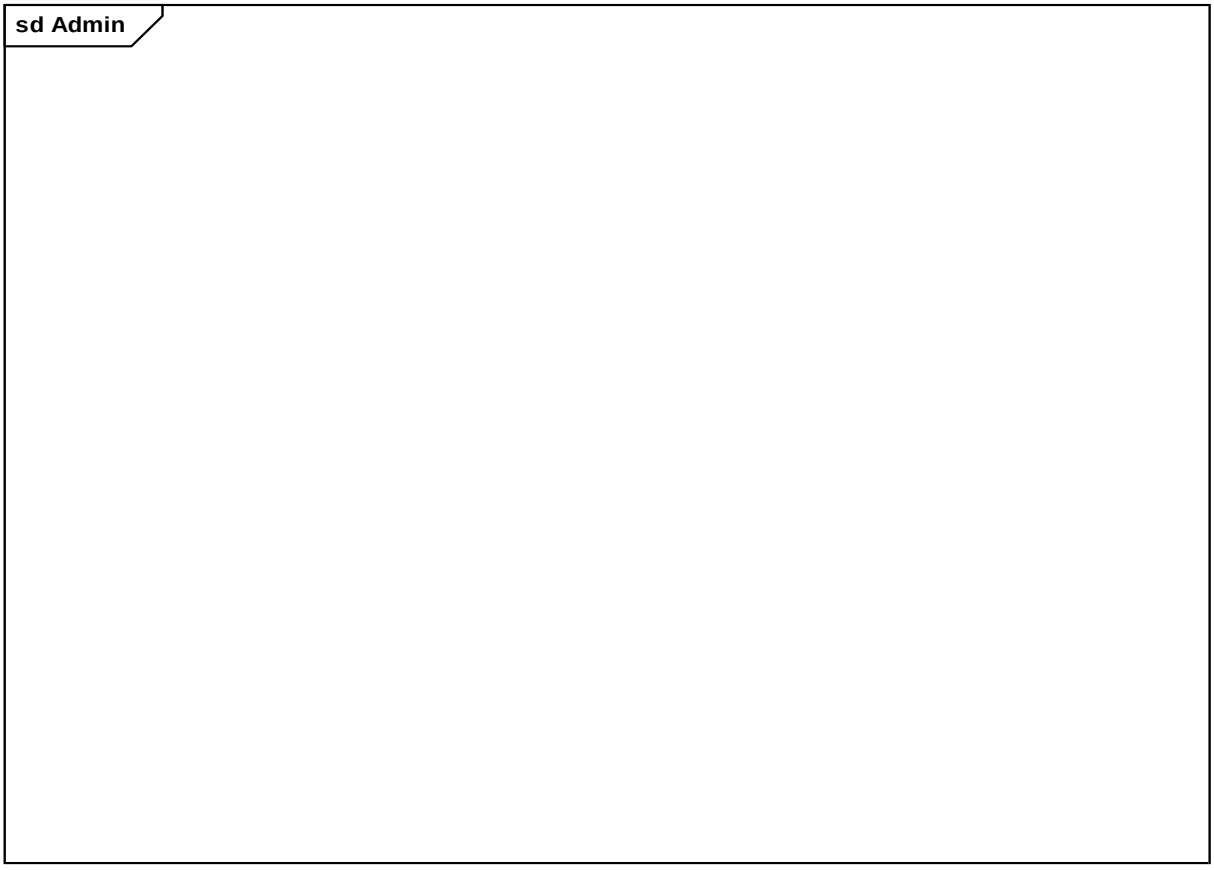

Figure 14: diagramme de séquence – Administrateur

1.4.4. Diagrammes de classes:

Le diagramme de classes est un schéma utilisé pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Parmi toutes les tables existantes dans la base de données de la RADEM, nous présentons sur ce diagramme celle dont nous avons eu besoin.

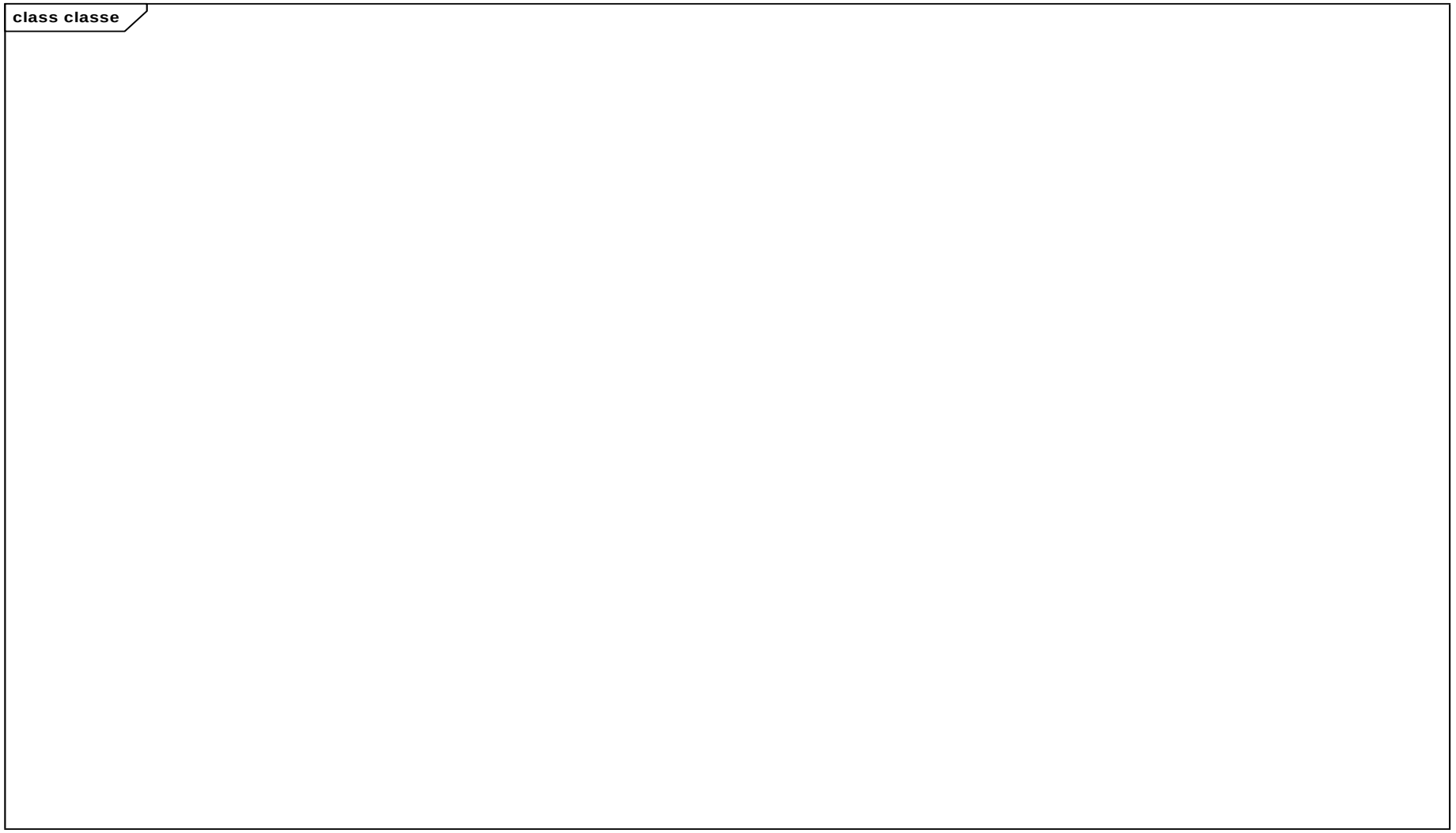

Figure 15: diagramme de classe

Dans ce chapitre, nous avons précisé les besoins fonctionnels en fonction des acteurs, et les besoins non fonctionnel, de notre projet. Nous avons défini les acteurs et nous avons effectué une conception de notre système, et ce à travers des diagrammes de cas d'utilisation, des diagrammes de séquences, un diagramme de classe et un diagramme de packages.

CHAPITRE III
REALISATIONS ET INTERFACES

Langages utilisés :

Nous avons utilisé quelques nouvelles notions du JavaScript notamment JQuery et NodeJS qui ont été primordiales pour la réalisation du projet:

➔ **NodeJS** : une plateforme logicielle libre et événementielle en JavaScript NodeJS nous permet d'**utiliser le langage JavaScript sur le serveur.**

Node.js a pu mettre en place une architecture de code entièrement non bloquante ce qui a pour objectif de minimiser le temps de traitement. Les figures 16, 17 et 18 montrent la différence entre deux architectures bloquante et une non-bloquante.

```
1 Télécharger un fichier
2 Afficher le fichier
3 Faire autre chose
```

Figure 16 : Modèle non bloquant.

```
1 Télécharger un fichier
2     Dès que c'est terminé, afficher le fichie
3 Faire autre chose
```

Figure 17 : Modèle bloquant.

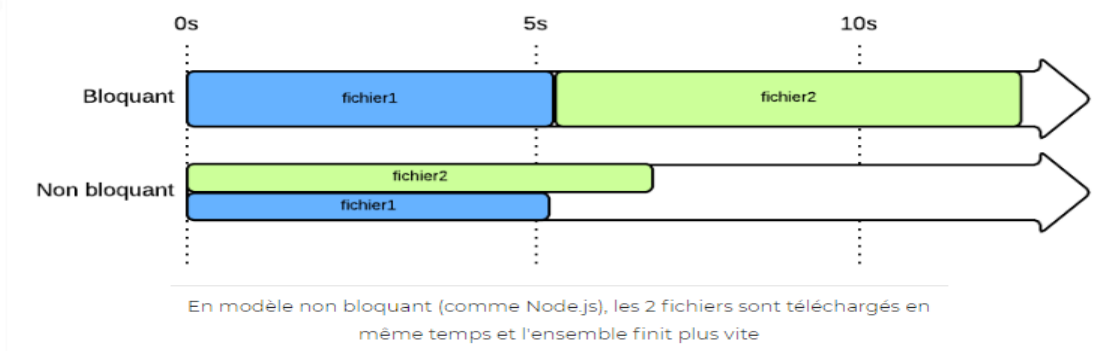

Figure 18: Modèle bloquant vs modèle non bloquant.

Parmi les bibliothèques du JavaScript, les Framework et les modules de Node.js que nous avons utilisés, il y a:

- **Express.js** est un framework pour construire des applications web basées sur Node.js. C'est de fait le framework standard pour le développement de serveur en Node.js.

```
var express = require('express');
var app = express();
```

- Demander d'inclusion d'Express.
- Création d'un objet « **app** » en appelant la fonction express().

Figure 19 : exemple - Express

- **Axios** est une bibliothèque JavaScript très populaire que nous utilisons pour exécuter des requêtes HTTP, qui fonctionne à la fois sur les plates-formes Browser et Node.js.

```
const axios = require('axios');
```

- Inclusion d'Axios.

Figure 20 : exemple - Axios

Création des routes avec fonction express et exécution des requêtes http:

```
app.get('/informationstation', function (req, res) {
    let site = req.query.site;
    axios.get(process.env.urlrest + '/getStationList', {
            params: {
                codeSite: site,
                status: '1',
                _type: 'json'
            }
        })
        .then(function (response) {
            res.send(response.data);
        });
});
```

Figure 21 : Exemple – route 1

```
app.get('/lastticket', function (req, res) {
    let site = req.query.site;
    let station = req.query.station;
    axios.get(process.env.urlrest + '/getIndicator', {
            params: {
                codeSite: site,
                stationId: station,
                _type: 'json'
            }
        })
        .then(function (response) {
            res.send(response.data);
        });
});
```

Figure 22 : Exemple – route 2

- **HTTP :** Node.js possède un module intégré appelé HTTP, qui lui permet de transférer des données via le protocole HTTP (Hyper Text Transfer Protocol).

```
var http = require('http').Server(app);
```

- Inclusion du module HTTP et appelle du serveur

Figure 23: exemple – HTTP

- **HttpClient** : c'est un service **angular** qui permet l'interaction avec un serveur.
- **Angular :** est un framework JavaScript Open Source développé par Google. Il utilise l'architecture MVM (Modèle Vue Modèle), proche du modèle MVC.
  Il est considéré comme un langage « côté client », ceux-ci permettent de gérer l'interface utilisateur de chaque page (affichage, interactions…) de façon dynamique et viennent en complément aux langages côté serveur.
- **Socket.io** est un module de **Node.js** qui permet de créer des Web **Sockets,** c'est-à-dire des connections bidirectionnelles entre clients et serveur qui permettent une communication en temps réel.

```
var io = require('socket.io')(http);
```

Inclusion du module Socket.io.

Figure 24: exemple – Socket.io

```
io.on('connection', function (socket)
```

La connexion.

Figure 25: exemple – Socket.io connexion

```
io.sockets[element.idsocket].emit('update')
```

Mise à jour de l'écran d'une station

Figure 26: exemple – Socket.io update

Cookie: un fichier texte enregistré dans le disque dur d'un internaute à la demande du serveur gérant le site Web visité.

Dans notre cas, le cookie nous permet de récupérer le code de l'agence qui se connecte au serveur pour pouvoir lui envoyer les informations qui la concernent.

```
var cookieParser = require('cookie-parser');
```

Inclusion du cookie.

Figure 27: exemple – Cookie

```
app.get('/acceuil/:code', function (req, res) {
    let site = req.params.code;
  getAgence(site).then(function(agence){
    res.cookie("code", site);
    res.cookie("agence",agence)
    res.sendFile(__dirname + '/public/index.html');

  });
});
```

Figure 28: exemple – Cookie 1

Récupération du code du site et son renvoie en cookie avec les informations de l'agence en question.

- **CORS :** Le « *Cross-origin resource sharing* »ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.

```
var cors = require('cors');
```

Figure 29: exemple – CORS

Inclusion du CORS.

```
app.use(cors());
```

Figure 30: exemple – CORS usage

Usage simple du CORS.

- L'objet **Promise** est utilisé pour réaliser des traitements de façon asynchrone. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur voire jamais.

➔ **JQuery** est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. Elle permet de manipuler les éléments mis en place en HTML (textes, images, liens, vidéos, etc.) et mis en forme en CSS (position, taille, couleur, transparence, etc.) en utilisant des instructions simples qui donnent accès aux immenses possibilités de JavaScript et d'AJAX.

Application utile :

Beaucoup de données ont été prédéfinis sur la base de données, à savoir les fonctions mais surtout les variables dont nous avons eu besoin pour le développement de notre application. Pour les récupérer, nous avons utilisés **Postman** qui est une application permettant d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer, et de les organiser en Collections.

Postman assure également la gestion des Environnements, qui permet de contextualiser des variables et d'exécuter des requêtes ou des séries de requêtes dans différents configurations.

Pour avoir un retour du serveur via Postman, on exécute l'url :
http://192.168.60.134/eSirius/webservices/rs/client/v1.5 suivie de celui de la fonction à laquelle nous souhaitons avoir accès :

Exemple :

- http://192.168.60.134/eSirius/webservices/rs/client/v1.5/getListStaion pour accéder à la liste des stations dans une agence.
- http://192.168.60.134/eSirius/webservices/rs/client/v1.5/getServiceList pour avoir la liste des services.

Interfaces :

1.1. Ecran d'accueil :

Nous commençons par l'interface d'accueil qui est une application web. C'est une interface dynamique qui sera visible dans les écrans de toutes les agences de la RADEM. Nous avons choisi de vous montrer trois interfaces de trois agences différentes.

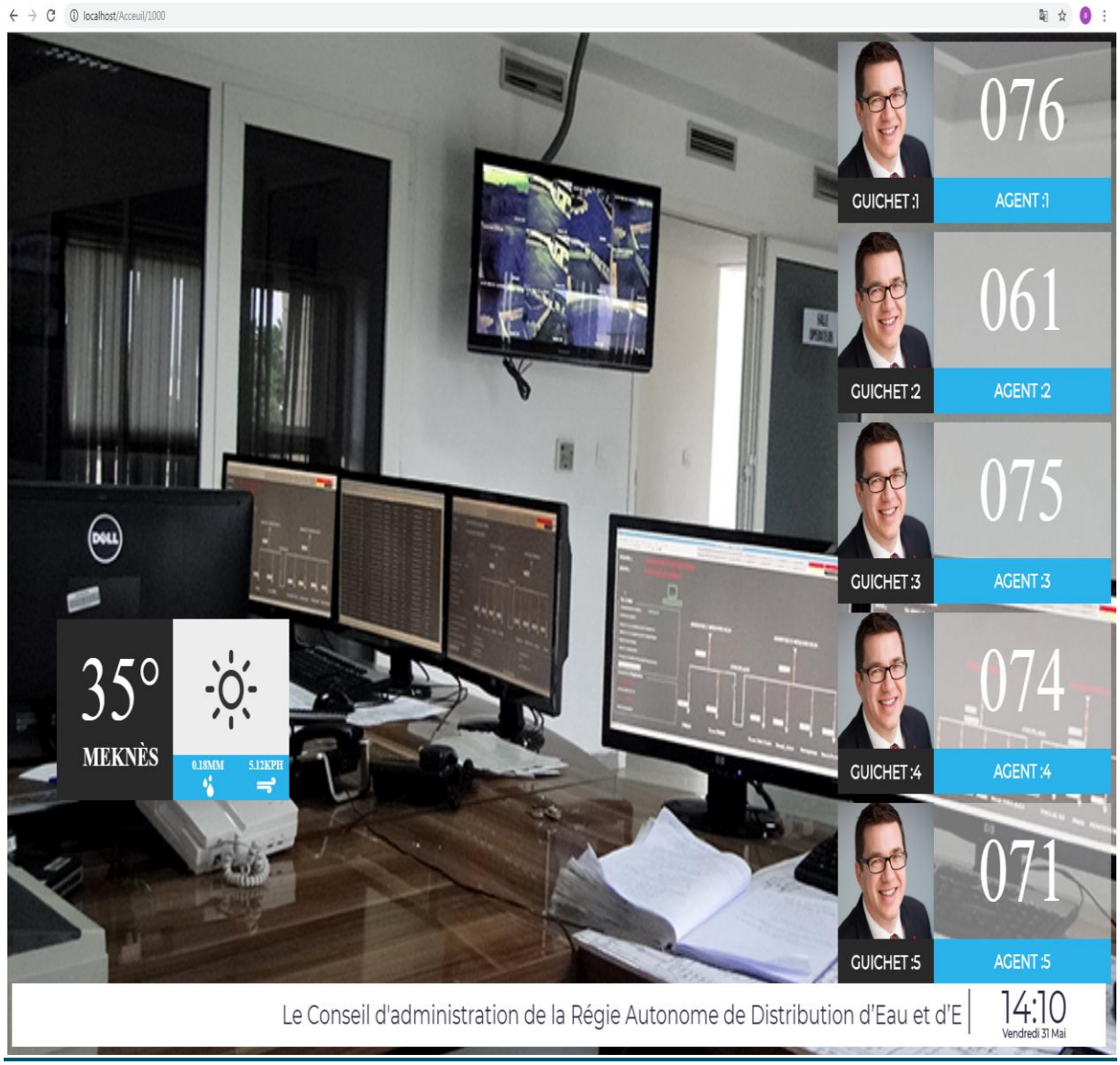

Figure 31 : interface –accueil 1

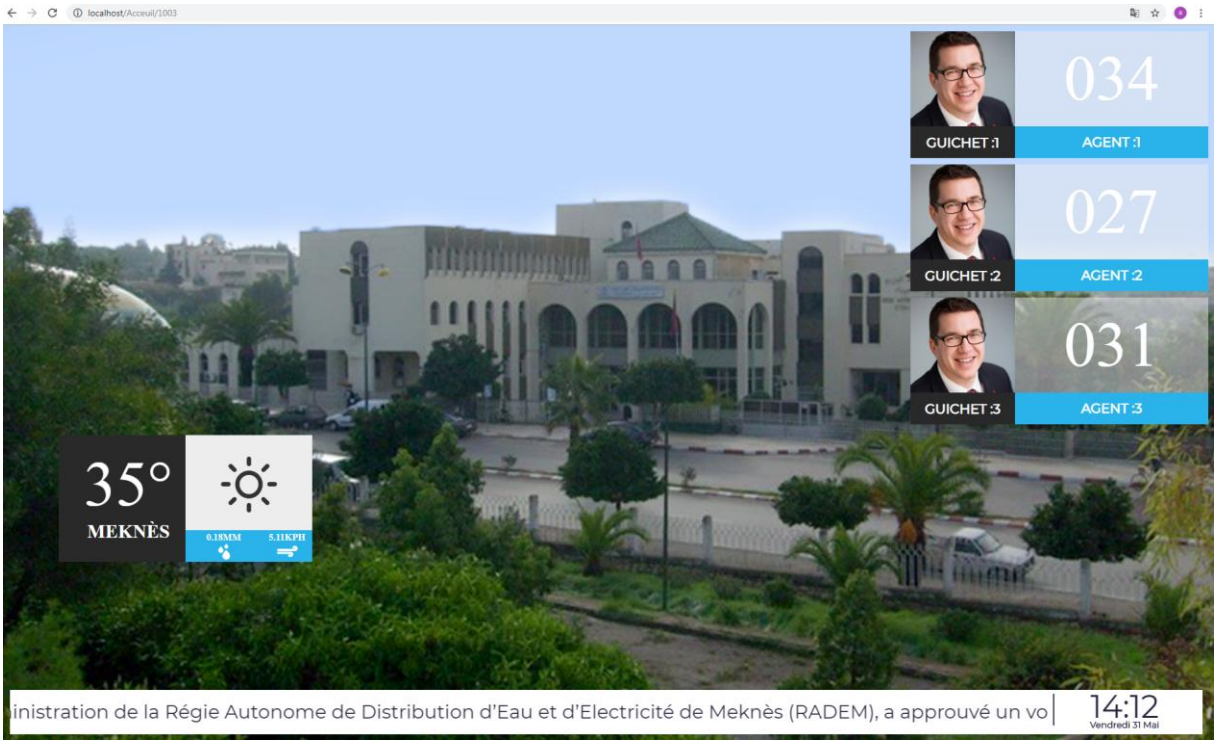

Figure 32: interface –accueil 2

Figure 33: interface –accueil 3

Chaque interface est caractérisée par un code ; par exemple, l'URL pour l'accès à l'application depuis le siège est : 192.168.100.55/Acceuil/1000 comme le montre les deux figures suivantes :

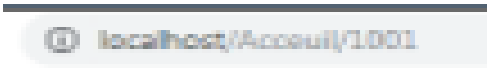

Figure 34 : URL 1

Figure 35 : URL 2

Une fois allumée, l'écran de l'agence envoie son code au serveur,

- Ce dernier le récupère et le lui renvoie en cookie accompagné des informations concernant cette agence.

```
app.get('/acceuil/:code', function (req, res) {
    let site = req.params.code;

  getAgence(site).then(function(agence){
    res.cookie("code", site);
    res.cookie("agence",agence)
     res.sendFile(__dirname + '/public/index.html');
     });
});
```

Figure 36 : Code 1

Sur l'interface d'accueil, on trouve à droite les agents de l'agence avec le numéro du ticket de client en cours de traitement.

Figure 37 : Liste agents

- La fonction qui retourne les informations des agents est :

```
app.get('/informationagent', function (req, res) {
    let site = req.query.site;
    axios.get(process.env.urlrest + '/getAgentListByStatus', {
            params: {
                codeSite: site,
                status: '0',
                _type: 'json'
            }
        })
        .then(function (response) {
            res.send(response.data);
        });
});
```

Figure 38 : Code 2

- En exécutant le lien, le retour du serveur pour une agence donnée (en utilisant Postman) est le suivant :


```
{
    "name": "Agent",
    "id": 3,
    "connected": true,
    "firstName": "2"
},
{
    "name": "Agent",
    "id": 2,
    "connected": true,
    "firstName": "1"
},
{
    "name": "Agent",
    "id": 6,
    "connected": true,
    "firstName": "5"
},
{
    "name": "Agent",
    "id": 5,
    "connected": true,
    "firstName": "4"
},
```


Figure 39 : Réponse – code 2

Pour chaque agent, nous trouvons le numéro de ticket du client en cours de traitement. La fonction qui nous retourne cela est la suivante :

```
app.get('/lastticket', function (req, res) {
    let site = req.query.site;
    let station = req.query.station;
    axios.get(process.env.urlrest + '/getIndicator', {
            params: {
                codeSite: site,
                stationId: station,
                _type: 'json'
            }
        })
        .then(function (response) {
            res.send(response.data);
        });
});
```

Figure 40 : Code 3

Le retour du serveur à cette requête est le suivant :


```
{
    "callableVisitorNumber": 0,
    "rdvScheduledNumber": 0,
    "lastService": "Espace Reclamation",
    "symbolicPage": "stateWait",
    "waitingVisitorNumber": 0,
    "myVisitorNumber": 0,
    "lastTicket": "084",
    "rdvWaitingNumber": 0,
    "concurrentAccess": false,
    "siteVisitorsNumber": 86,
    "anticipatedVisitorPresent": false,
    "needPopUpAlert": false,
    "ticketDisabled": false,
    "stateLevel": -1,
    "ticketMode": 0,
    "alertNumber": 0,
    "siteWaitingVisitorsNumber": 0,
    "logout": false,
    "needUpdate": false,
    "stateTime": "00:03:15",
    "siteOpen": true,
    "postPonementCallableVisitorNumber": 0,
    "rdvScheduledNumberAllServices": 0,
    "postPonementWaitingVisitorNumber": 0
}
```


Figure 41 : Réponse – code 3

On remarque également la météo ; cette dernière a été traité du coté serveur et du coté client :

Grace à une API de météo, nous récupérons toutes les informations de météo que nous souhaitons :

```
app.get('/meteo', function (req, res) {
    axios.get('https://api.darksky.net/forecast/27622eda4cfd8121cb5939f3639f50fa/33.8978,-5.532?units=si')
        .then(function (response) {
            res.send(response.data);
        });
});
```

Figure 42 : Code 4

En exécutant url, le serveur nous retourne beaucoup d'informations dont nous n'aurons pas besoin comme le montrent les deux figures suivantes:


```
{
    "time": 1557381600,
    "summary": "Clear",
    "icon": "clear-day",
    "precipIntensity": 0,
    "precipProbability": 0,
    "temperature": 12.1,
    "apparentTemperature": 12.1,
    "dewPoint": 10.5,
    "humidity": 0.9,
    "pressure": 1018.12,
    "windSpeed": 0.96,
    "windGust": 1.06,
    "windBearing": 62,
    "cloudCover": 0.02,
    "uvIndex": 0,
    "visibility": 16.09,
    "ozone": 326.12
},
```


Figure 43 : Réponse 1 – code 4


```
{
    "time": 1557320400,
    "summary": "Partly Cloudy",
    "icon": "partly-cloudy-day",
    "precipIntensity": 0,
    "precipProbability": 0,
    "temperature": 25.22,
    "apparentTemperature": 25.22,
    "dewPoint": 12.73,
    "humidity": 0.46,
    "pressure": 1017.81,
    "windSpeed": 2.95,
    "windGust": 3.39,
    "windBearing": 319,
    "cloudCover": 0.33,
    "uvIndex": 8,
    "visibility": 10.01,
    "ozone": 327.79
},
```


Figure 44 : Réponse 2 – code 4

Du coté client, nous récupérons les informations nécessaires à notre développement en utilisant le JQuery pour mieux filtrer la réponse du serveur :

```
function getMeteo() {
    return new Promise(function (resolve, reject) {
        $.get('http://192.168.100.55/meteo', function (res) {
            console.log(res.currently);
            var meteo = {
                icon: '',
                temp: res.currently.temperature,
                humi: res.currently.humidity,
                speed: res.currently.windSpeed
            };
            if (res.currently.summary == 'Drizzle') {
                meteo.icon = 'wi-day-fog'
            } else {
                if (res.currently.summary == 'Partly Cloudy') {
                    meteo.icon = 'wi-day-cloudy'
                } else {
                    if (res.currently.summary == 'Possible Light Rain') {
                        meteo.icon = 'wi-day-rain'
                    } else {
                        if (res.currently.summary == 'Clear') {
                            meteo.icon = 'wi-day-sunny'
                        }
                    }
                }
            }
            resolve(meteo);
        });
    });
}
```

Figure 45 : Code 5

La réponse est affichée sur l'écran grâce à la fonction :

```
socket.on('meteo', function (data) {
    console.log(data);
    getMeteo().then(meteo => {
        console.log(meteo);
        $('.temp span').text(meteo.temp.toFixed(0));
        $('.conditions i').addClass(meteo.icon);
        $('.rain').text(meteo.humi + 'MM');
        $('.wind').text(meteo.speed + 'KPH');
    })
});
```

Figure 46 : Code 6

- Et ce sous la forme :

Figure 47 : Météo

Nous avons réalisé quelques fonctions sur le coté client pour pouvoir régler la forme de la date afin qu'elle soit adapté par rapport au format CSS :

```
updateDateTime();
  function updateDateTime(){
        $('.date').text(getDate());
        $('.time').text(getTime());
}
// date Update
setInterval(function () {
    updateDateTime()
},60000);
```

```
updateDateTime();
function updateDateTime(){
    $('.date').text(getDate());
    $('.time').text(getTime());
}
// date Update
setInterval(function () {
    updateDateTime()
},60000);
// le nom du jour
function getNomDay() {
    var date = new Date();
    var tabJ = new Array("Dimanche", "Lundi", "Mardi", "Mercredi",
    "Jeudi", "Vendredi", "Samedi")
    return tabJ[date.getDay()];
}
// le nom du mois
function getNomMonth() {

    var date = new Date();
    var tabM = new Array("Janvier", "Fevrier", "Mars", "Avril", "Mai", "Juin",
    "Juillet", "Aout", "Septembre", "Octobre", "Novembre", "Decembre")
    return tabM[date.getMonth()];
}
```

Figure 48 : Code 7

La dernière chose que nous trouvons sur l'interface est l'actualité ; cette dernière est également récupérée de la base de données :

```
app.get('/news', function (req, res) {
        let info = req.params.info;
        let sql = `select *  from news ;`;
        con.query(sql, function (err, result) {
            if (err){
                console.log(err);
            }else{
                res.send(result);
            }
          });
});
```

Figure 49 : Code 8

L'actualité s'affiche sur l'interface grâce à la fonction sur le coté client :

```
    $.get('http://192.168.100.55/news', function (res) {
       $('marquee').text(res);
    });
});
```

Figure 50 : Code 9

1.2. Application d'authentification :

Cette application a été réalisé avec le langage Java, elle sert à l'authentification des agents de la RADEM et à faire appel au client suivant.

Figure 51 : Interface – Authentification 1

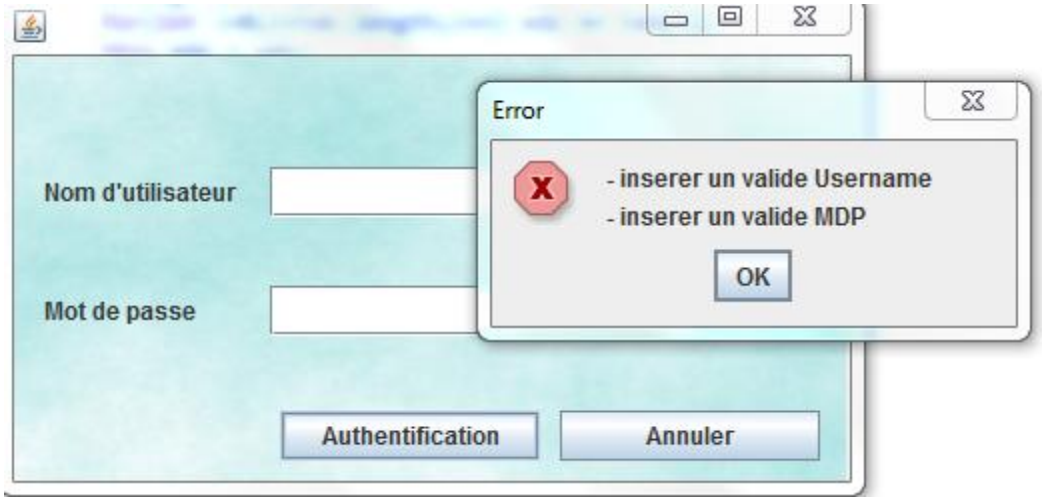

Figure 52 : Interface – Authentification 2

Ces deux figures représentent la première interface de l'application desktop, qui consiste à demander au client de saisir son nom d'utilisateur et son mot de passe.

Cette deuxième interface permet à l'agent de connaitre le ticket du client en cours de traitement ainsi que celui du client suivant.

Figure 53 : Interface – Appel client suivant

1.3. L'application de l'administrateur :

Cette application n'est accessible que par les administrateurs, et leur permet d'avoir un aperçu direct sur les informations des agences, à savoir le rondement des agences, le nombre d'agents disponibles, le temps de travail moyen des agents par semaine ....

Figure 54 : Interface – Application admin 1

Nous avons trouvé judicieux de permettre à chaque administrateur d'avoir sur son application son propre agenda, comme le montre la figure suivante :

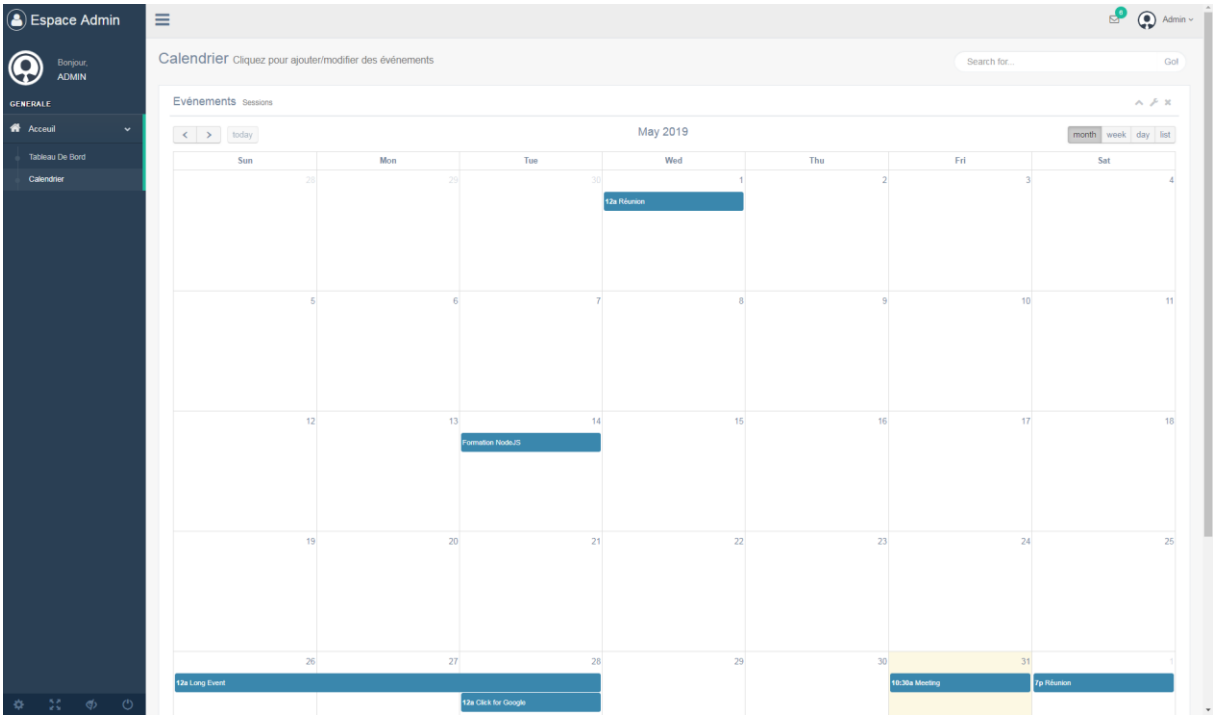

Figure 55 : Interface – Application admin 2

Dans ce dernier chapitre, nous nous sommes intéressés aux langages que nous avons utilisés pour développer nos applications. Nous nous sommes également intéressés à l'application Postman qui nous présentait les réponses du serveur. Pour finalement présenter nos interfaces.

CONCLUSION GENERALE

Notre projet a comme objectif principal la gestion de la file d'attente des agences de la RADEM, ce qui ne nous a pas empêché de nous investir corps et âme dans toutes les tâches qu'on nous a prescrite, à savoir le développement de l'application desktop qui permet la gestion de l'authentification des agents, l'appel du client suivant, et l'application web des administrateurs qui leur permet d'avoir un aperçu sur la performance hebdomadaire des agences, la disponibilité des agents, et bien d'autres données. Cette application facilitera la vie aux agents, aux clients et aux administrateurs et leur permettra de mieux s'invertir dans des tâches primordiales.

Pour la réalisation du projet, nous avons commencé par une documentation afin de mieux connaitre les langages de développement surtout que nous avons utilisé de nouveaux outils comme le Node.JS et le JQuery. Nous avons ensuite passé à l'analyse et la spécification des besoins à travers des échanges avec notre encadrant de stage, chose qui nous a permis de spécifier les différents acteurs interagissant avec l'application. Après cela, nous nous sommes intéressés à la conception du projet dans laquelle nous avons fixé la structure globale de l'application. Et enfin, nous nous sommes penchés sur le développement de notre application.

Nous avons été plutôt déçus de ne pas pouvoir contribuer en la totalité du projet développé au sein du service informatique de la RADEM quoique nous ayons réalisé une belle partie par rapport au temps que nous avions. Malheureusement, la plupart de nos efforts furent vains puisque nous n'avions pas le droit d'accéder à la base de données de la RADEM. Cela a borné notre développement surtout pour les dernières tâches (gestion de l'authentification et l'application admin).

WEBOGRAPHIE

La période durant laquelle nous avons visité ces sites est du mois d'avril au mois de juin de l'année 2019.

- https://fr.vuejs.org/
- https://openclassrooms.com
- https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203585-javascript/
- https://blog.arcoptimizer.com/presente-axios-un-client-http-populaire-base-sur-la-promesse
- https://dantotsupm.com/2009/07/09/besoins-non-fonctionnels-non-functional-requirements/
- https://www.futura-sciences.com/tech/definitions/internet-cookie-469/
- https://ppk.developpez.com/tutoriels/javascript/gestion-cookies-javascript/
- https://socket.io/
- https://expressjs.com/fr/
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise
- https://www.code4corsica.com/blog/technique/quest-ce-que-angular