-

 **DOSSIER-PROJET**

◢

|  |  |  |
| --- | --- | --- |
| Nom de naissance | ▶ | FOUCHERE |
| Prénom | ▶ | Laetitia |
| Adresse | ▶ | 9 |

**Titre professionnel visé**

**Développeur-se logiciel - Niveau III**

 OCUSTAT

Outil statistique de numérisation de documents

**Sommaire**

[Remerciements 4](#_bookmark0)

[Project Abstract 5](#_bookmark1)

[Résumé du projet 6](#_bookmark1)

Lexique……………………………………………………………………….7

Introduction………………………………………………………………….8

[Liste des compétences du référentiel 9](#_bookmark2)

[Cahier des charges 11](#_bookmark4)

1. [Expressions des besoins 11](#_TOC_250054)
2. [Risques et difficultés éventuels 11](#_TOC_250054)

[Spécifications fonctionnelles 12](#_bookmark4)

1. [Modélisation 12](#_TOC_250045)
   1. [Diagramme Use case 13](#_TOC_250044)
   2. [Diagramme d’activité 14](#_TOC_250044)
   3. [Diagramme de classes 18](#_TOC_250044)
2. [Conception de la base de données 19](#_TOC_250045)
3. [Maquette de l’interface 20](#_TOC_250045)

[Gestion de projet 25](#_bookmark4)

1. [Méthodes agiles 25](#_TOC_250045)
2. [Lync 26](#_TOC_250045)
3. [Trello 26](#_TOC_250045)
4. [GitLab 27](#_TOC_250045)

[Réalisation de l’application/Service 29](#_bookmark6)

1. [Vie du projet 29](#_TOC_250045)
   1. [Contexte de travail 29](#_TOC_250044)
   2. [Analyse des données 29](#_TOC_250044)
      * b-1) [Programme WinForm console 30](#_TOC_250044)
      * b-2) [Web Service 30](#_TOC_250044)
      * b-3) [Interface Angular 30](#_TOC_250044)
   3. [Mise en place de la base de données 30](#_TOC_250044)
2. [Langes utilisés 31](#_TOC_250045)
   1. [SGBDR 31](#_TOC_250044)
   2. [Back-end 31](#_TOC_250044)
   3. [Front-end 3](#_TOC_250044)2
3. La Programmation Orientée Objet 32
   1. [Héritage 3](#_TOC_250044)2
   2. [Polymorphisme 3](#_TOC_250044)2
4. Environment de travail 33
   1. [Visual Studio 2015 3](#_TOC_250044)3
   2. [Visual Studio Code 3](#_TOC_250044)3
5. [Réalisation du Service Web WCF 3](#_TOC_250044)3
   1. [La Structure WCF 3](#_TOC_250044)3
      * a-1) [L’interface du service 3](#_TOC_250044)4
      * a-2) [Cheminement 35](#_TOC_250044)
   2. [La connexion à la base 35](#_TOC_250044)
      * b-1) [Interface IDataAccess 3](#_TOC_250044)5
      * b-2) [Les classes (modèles) 36](#_TOC_250044)
      * b-3) [Classe DataAccess 37](#_TOC_250044)
      * b-4) [La classe Service 39](#_TOC_250044)
   3. [Permission d’accès à l’interface utilisateur 42](#_TOC_250044)
6. [Réalisation de l’Interface utilisateur 43](#_TOC_250044)
   1. [Angular-cli 4](#_TOC_250044)3
   2. [Installation 4](#_TOC_250044)3
   3. [Structure et Material Design 4](#_TOC_250044)4
7. [Graphique 51](#_TOC_250044)

[Conclusion 57](#_bookmark7)

**REMERCIEMENTS**

Tout d’abord, merci à La Poste d’avoir mis à disposition de tous leurs agents, la possibilité de se reconvertir au métier de développeur WEB.

Pour la confiance, l’accueil et la bienveillance qui nous a été accordée, je souhaite remercier tous les membres de Simplon.co. Aussi d’avoir instauré un esprit d’équipe dans notre apprentissage.

Merci aux apprenants d’avoir toujours été là pour me soutenir, m’aider à avancer tant techniquement que moralement. Ainsi que pour tous les moments de partage aussi bien dans l’échec que dans la réussite.

Merci à mon tuteur V.JAMIN d’avoir été à l’écoute tout au long de la formation et avoir pris des décisions en fonction de nos échanges pour me mettre dans les meilleures conditions.

Merci à Souleimane et Sophie d’avoir eus un impact considérable sur ma progression, ma compréhension technique et du contexte professionnel ainsi que du métier de développeur au sein du groupe Docapost à NATION. Ils ont été et restent encore pour moi des modèles professionnels.

Enfin, encore merci à tous ces acteurs et autres qui ont eu le courage de me soutenir et de faciliter mon apprentissage durant ces 9 mois de formation. Ma gratitude et ma reconnaissance sont à la hauteur de leurs efforts.

**PROJECT ABSTRACT**

I work for a group called Docapost in their dematerialization department. The dematerialization process consist of scanning documents, archiving them and providing it to customers. According to the customers’ specificities, documents are prepared, then scanned and sent automatically to the WORFLOW of DOCUPOST where they are processed. After DOCUPOST's treatment, deliverables are generated and, depending on the case, integrated into customers’ website, deposited on secured spaces or copied on cd’s.

My application is used to bring back statistics from the DOCUPOST's productive line then improve productivity by giving visibility in order to organize solutions. One of the DOCUPOST steps is to update each processed batch which are some indicators' tables that would be exploited to develop those statistics. The studied tasks are mainly automatic recognition and those who require a human intervention (such as videocoding). To show those statistics, the charts are fed and regulary updated in order to quickly identify malfunctions in order to anticipate or resolved it on a short time.

My app's functioning is intuitive. We connect it with a secured connection window by entering username and password. There are two connection status: « Opérateur » and « Responsable ». Each user is able to access to graph and can personalize it by an option board in the header. Each graph represent a view and we can surf between it as in a carousel. The “Responsable” user is allowed to create a client or an app by accessing to forms. He can also extract data from a .csv file and manipulate it in an excel one. About Backend, the technologies I used are Microsoft WebService in C#, using Visual Studio as IDE. About Frontend, I used Angular, HTML, CSS and Typescript.

**Resume du projet**

Je travaille au sein du groupe Docapost, division dématérialisation.

Le processus de dématérialisation consiste à numériser des documents, les archiver et les mettre à disposition des clients.

Selon les spécificités de chaque client, les documents subissent une préparation avant d’être numérisés puis envoyés automatiquement dans le WORFLOW de traitement d’images DOCUPOST. En sortie de la solution DOCUPOST, des livrables sont générés pour être selon le cas : intégrés sur les sites internet des clients, déposés sur des espaces sécurisés ou copiés sur des supports optique.

Mon application est destinée à faire remonter des statistiques de la chaine de production DOCUPOST afin d’en améliorer sa productivité en donnant de la visibilité pour planifier des solutions. Une étape DOCUPOST met à jour pour chaque lot traité, des tables d’indicateurs qui seront exploitées pour l’élaboration de ces statistiques. Les tâches étudiées sont essentiellement les tâches de reconnaissance automatique et celles nécessitant les interventions opérateurs comme le vidéocodage.

Pour représenter ces statistiques, des graphiques sont alimentés et mis à jour régulièrement pour visualiser rapidement si des dysfonctionnements émergent, dans le but d’anticiper ou résoudre ceux-ci dans un délai réduit.

Le fonctionnement de l'application est intuitif. On y entre par une fenêtre classique de connexion en saisissant ses identifiants. Il existe deux types de connexion : "Opérateur" ou "Responsable". N'importe quel utilisateur peut lire les données sous forme de graphique et les personnaliser via un tableau d'options en tête de page. Chaque graphique représente une vue; on navigue entre elles à la façon d'un carrousel. L'utilisateur ayant pour rôle "Responsable" peut, en plus, accéder aux formulaires de création d'un Client et d'une Application. Il peut aussi extraire des données dans un fichier .csv afin de les manipuler dans Excel.

Les technologies abordées pour le Backend sont le Winform et WebService de Microsoft en langage C# sous l’IDE Visual Studio. Pour le Frontend, Angular.

**Lexique**

**Glossaire et Définitions**

**Lot :** Ensemble de documents d’un client, à numériser.

**Code Application :** Code qui détermine le type de traitement d’un modèle Docupost.

**Modèle Docupost :** Nom du modèle qui va subir le traitement de numérisation.

**Famille de modèle :** Type de document (Facture, Accusé de réception, pièces jointes…)

**Référentiel :** Informations concernant les clients que va interroger DOCUPOST lors de la reconnaissance.

**Workflow :** Processus d’automatisation des tâches d’une application (numérisation).

**Utilisateur :** Une personne habilitée à utiliser mon application.

**Reconnaissance :** La reconnaissance est le moment de la numérisation ou le logiciel DOCUPOST lit, reconnait et redirige un document.

**WCF :** Windows Communication Foundation. Infrastructure pour la création d’applications orientées service.

**IDE :** Environnement de développement intégré. Ensemble d’outils pour améliorer la productivité des programmeurs.

**Framework :** Ensemble de bibliothèques accompagné du plan de l’architecture cible.

**IHM :** Interface Homme Machine. L’ensemble des moyens utilisés par l’Homme pour communiquer avec la machine.

**SGBDR :** System de Gestion de Base de Données Relationnelles.

**SQL :** Structured Query Language. Langage pour faire des requêtes dans les bases de données.

**URL :** Uniform Resource Locator. Adresse d’un site ou d’une page hypertexte sur internet.

**Wireframe :** Représente le schéma d’une page Web.

**HTML :** Hypertext Markup Langage. Langage informatique fait de balises.

**CSS :** Cascading Style Sheets. Langage informatique de mise en forme de contenu HTML.

**SERES :** Entreprise, basée à NANTES, qui développe l’outil DOCUPOST et livre ses données ses aux utilisateurs.

**Introduction**

Dans un souci de cohérence avec notre société moderne qui se doit d’être performante, innovante à la pointe de la technologie en restant à l’écoute de chacun. La Poste met en place un processus de **digitalisation** de l’ensemble de ses services

Ainsi, pour répondre à ces besoins susnommés, l’idée d’un partenariat entre Simplon (école de formation professionnelle spécialisée dans les métiers du numérique) et La Poste, est devenue évidente. Offrant la possibilité à de futurs apprenants développeurs informatiques, web et mobile, une reconversion professionnelle : tout en donnant un accès (ouvert à tous les collaborateurs de La Poste) via le passage du titre professionnel de  développeur logiciel.

Durant cette formation de neuf mois en alternance avec un service informatique de la Poste, les 19 personnes qui ont été  acceptées  pour  une  formation  en  développement, orientée ​Java​, vont collaborer à cet objectif en créant une application Web.

Me concernant, l’équipe Docapost de la DSI de Nation m’a accueillie.

En me formant, elle a participé à renforcer mes acquis, elle m’a aidé dans la compréhension des concepts abordés en formation, elle m’a permis de découvrir des technologies non abordées en formation comme le langage C# car adepte des technologies Microsoft et ainsi que d'élargir mes domaines de compétences.

Je vais vous présenter maintenant mon projet, mon application WEB fonctionnelle basée sur l’agrégation de données (statistiques).

**LISTE DES COMPÉTENCES DU RÉFÉRENTIEL**

Chacune des compétences citées ci-dessous, relatives au protocole de la rédaction d’un projet, sont accompagnées de précisions sur les technologies abordées avant d’expliquer leur utilisation dans les chapitres suivant.

**Maquetter une application**

La modélisation de mon projet s’est faite avec l’outil en ligne https://www.draw.io/.

Les wireframes de mon application ont été réalisés avec Power point et Excel afin d’être animés avec OBS.

**Concevoir une base de données**

La création et / ou matérialisation de ma base de données relationnelle a été faite avec MySQL Worbench.

**Mettre en place une base de données**

A partir de sa modélisation. Le script SQL généré par Workbench à été exécuté dans SQLyog, logiciel utilisé en interne.

**Développer une interface utilisateur**

Le développement de l’interface utilisateur est fait avec Angular sous Visual Studio Code utilisant les modules d’intégrations de Material Angular ainsi que sa bibliothèque Javascript.

**Développer des composants d’accès aux données**

J’ai développé mes méthodes dans un service WEB selon les règles architecturales requises pour interroger ma base de données normalisée. Un second programme en WinForm est en cours de développement pour agréger les données des bases existantes vers ma base de données.

**Développer des pages web en lien avec une base de données**

Mon application permet, selon le rôle de l’utilisateur, de consulter, modifier et ajouter des données dans la base de données, depuis son navigateur web.

**Mettre en œuvre une solution de gestion de contenu ou e-commerce**

N/A : Mon chef d’œuvre n’est pas une solution de gestion de contenu ou e-commerce.

**Développer une application simple de mobilité numérique**

J’ai choisi d’utiliser Material Angular pour sa capacité à gérer les différents types d’appareils (pc, tablette, mobile).

**Utiliser l’anglais dans son activité professionnelle en informatique**

La norme professionnelle impose d’utiliser l’anglais lors de la conception des programmes contrairement aux commentaires et au nommage des bases de données. La plupart des tutoriaux et documents techniques utilisés pour la réalisation du projet sont en anglais.

**Compétences transversales de l’emploi type :**

* **Actualiser et partager ses compétences en développement informatique**

La réalisation de ce projet a été pour moi l’occasion d’approfondir ce que j’avais vu en cours et le transposer dans un autre langage : le C#.

J’ai partagé mes connaissances à mon entourage professionnel pendant nos réunions hebdomadaires et aux apprenants de ma promotion lors des présentations (Lightning Talks) ainsi que des échanges quotidiens.

Pour organiser mon travail durant l’activité professionnelle, j’ai utilisé Trello (gestion de projets), Lync (plateforme d’échange interne) et GitLab (pour partager et conserver mon code).

**CAHIER DES CHARGES**

# Expressions des besoins

Les utilisateurs de l’application DOCUSTAT pourront consulter en temps réel les statistiques sur la numérisation des documents produite sur les deux sites de production:

* Ce qui a été produit la veille. Un rendez-vous quotidien pour donner une visibilité sur le travail effectué jour après jour.
* Le ratio entre contenants et contenus, pour avoir une idée rapide de la densité du flux traité sur une période souhaitée.
* L’évolution du taux de reconnaissance pour les documents de chaque client ou par type de documents numérisés et ainsi cibler et contrôler les dysfonctionnements de la production.
* Un tableau qui aide à déterminer sur quelles applications vont être dispatchés les collaborateurs.

De plus, les utilisateurs désignés comme « responsable » bénéficieront d’autres fonctionnalités :

* Affilier le nom d’un client à son code dans la base de données depuis le formulaire.
* Voir le détail d’un client ou d’une application et modifier si nécessaire.
* Extraire des données d’un client pour les manipuler dans un fichier Excel afin de faire des démonstrations aux clients lors de réunions de réévaluation du contrat.

1. **Risques et Difficultés éventuels**

Les difficultés ou risques associés à ce projet que j’ai pu rencontrer sont :

* De façon personnelle :
  + - L’apprentissage d’un autre langage et IDE.
    - Se faire aider en formation suite à cette différence.
* Durant la conception :
  + - L’analyse de Bases de données existantes non normalisées.

**SPÉCIFICATIONS FONCTIONNELLES**

# Modélisation

# Présentation des diagrammes UML (Langage de Modélisation Unifié) : l’UML est un langage de modélisation graphique à base de pictogrammes normalisés.

# Grâce aux UML, je peux mettre en forme la conception du projet et les la modifier au fur et à mesure des évolutions et améliorations de mon application.

# Le diagramme des Use Case donne une vision d’ensemble des capacités du système et l’interaction avec les acteurs qui l’utilisent.

# Les diagrammes d’activité, toujours sous forme graphique, permettent de décrire visuellement et par étapes, les fonctionnalités introduites dans le diagramme des uses case. Des actions de l’utilisateur aux réponses du système en passant par les fonctionnalités internes, ils offrent aux développeurs un support très détaillé pour coder aisément.

# Le diagramme de classes lui, a pour vocation de modéliser les liens entre les classes métier ce qui donne un aperçu de la structure globale du code avant d’y intégrer les méthodes.

# Le diagramme des séquences n’est pas utilisé pour modéliser mon application. Le principe d’interaction entre les différents acteurs avec le système et la notion de représentation du temps n’est pas adapté dans mon cas.

## Diagramme USE CASE

## L’assignation du nom d’un client à son code et l’ajout d’informations sur une application se font depuis un formulaire. Une application, dans mon projet, est définie par un code qui détermine la façon dont sera traité un lot de documents pour un client.

## 

* 1. **Diagrammes d’activité**

1. **Connexion :**

Contexte : L’utilisateur souhaite accéder à l’application

Acteur(s) : Tous les utilisateurs ayant un droit d’accès



1. **Consultation et personnalisation des données :**

Contexte : L’utilisateur est entré dans l’application depuis la page d’accueil

Acteur(s) : Tous les utilisateurs ayant un droit d’accès



1. **Assigner le nom d’un client à son code :**

Contexte : L’utilisateur est entré dans le formulaire depuis la page d’accueil.

Acteur(s) : Utilisateur dont le rôle est Responsable.



1. **Ajouter des informations sur une application.**

Contexte : L’utilisateur est entré dans le formulaire depuis la page d’accueil.

Acteur(s) : Utilisateur dont le rôle est Responsable



* 1. **Diagramme des classes**



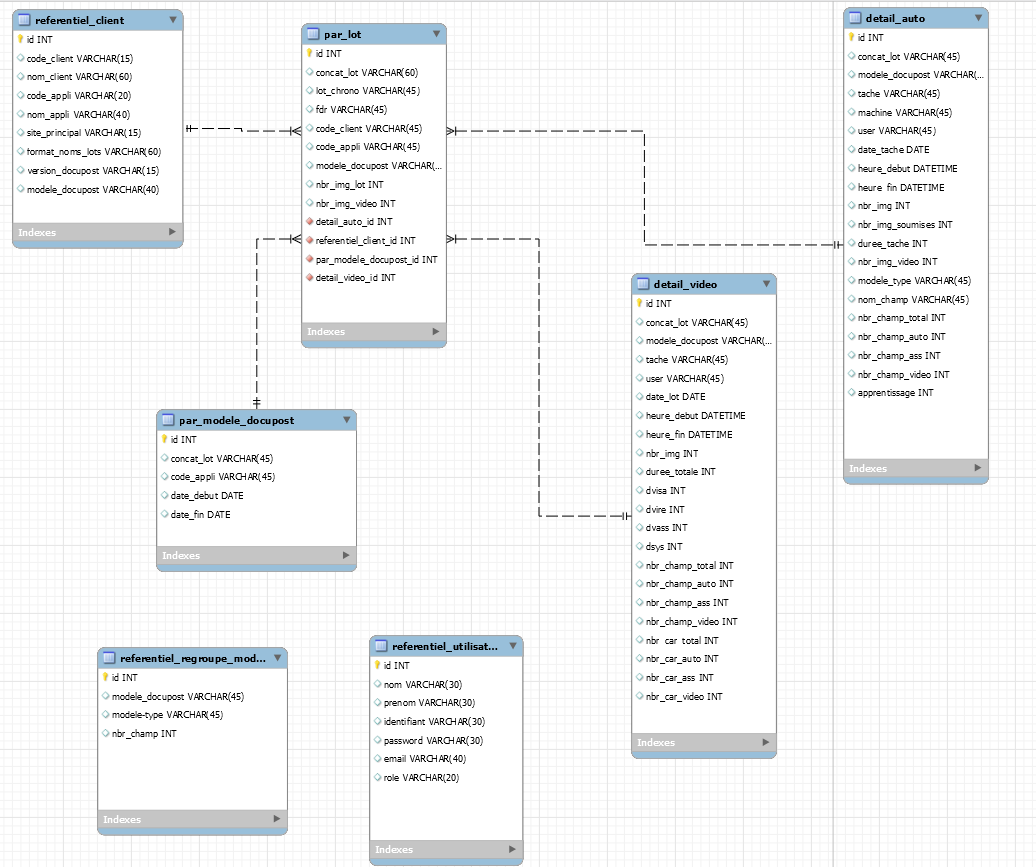
1. **Conception de la base de données**

La conception d’une base de données passe d’abord par sa modélisation (EER Diagram => Enhanced entity–relationship model => Modèle d'entité-relation amélioré).

Elle a pour principe de représenter de façon structurée, les données qui seront utilisées par l’application.

Ce modèle conceptuel décrit la sémantique de ces données en utilisant ici, l’outil SQL Workbench qui est un système de gestion de bases de données relationnelles connecté à un serveur.

Je me suis basée sur le diagramme des classes pour créer le model de ma base de données : stat\_num. J’y ai défini ses relations par le biais des flèches représentant les « many to one » ou « one to many ». Puis, j’ai paramétré les colonnes des tables en signifiant quel type de données y est attendu.  
  
Toutes les contraintes n’y sont pas paramétrées à ce jour car cette base de données sera modifiée lorsque le contact avec SERES sera établi.



1. **Maquette de l’interface**

Une fois les fonctionnalités (principales) déterminées, j’ai réalisé la maquette de l’interface avec PowerPoint. J’ai préparé la représentation des graphiques dans Excel car il est possible de les personnaliser :

* Type de graphique
* Abscisses
* ordonnées
* Type de valeur

Dès lors, J’ai pu les intégrer aux wireframe.

L’intérêt d’avoir utilisé PowerPoint fut de dupliquer facilement les slides pour y ajouter uniquement des détails de navigation, ceci pour aboutir à un enchainement tel une bobine de cinéma afin, justement, de les animés avec l’outil en ligne : <https://invisio.com/>.

**Pourquoi avoir animé la maquette de l’application ?**

Les différents interlocuteurs avec qui je travaille, n’étant jamais situés au même endroit au même moment, auraient été difficiles à réunir. Aussi, il n’était pas envisageable d’aller voir chacun d’eux. J’ai donc eu idée de partager cette vidéo pour que chacun puisse la regarder en temps voulu.

Cette maquette intègre des commentaires. Il était pratique pour les lecteurs de cette vidéo, d’avoir un aperçu concret de la navigation, de la mettre en pause pour prendre le temps de lire les commentaires ou revenir en arrière pour approfondir les remarques à m’envoyer en retour.  
  
Une fois toutes les remarques reçues par mail, j’ai pu apporter des corrections et produire une nouvelle vidéo, maquette version 2. Voici maintenant, quelques slides (wireframe) qui ont contribuées à cette animation :

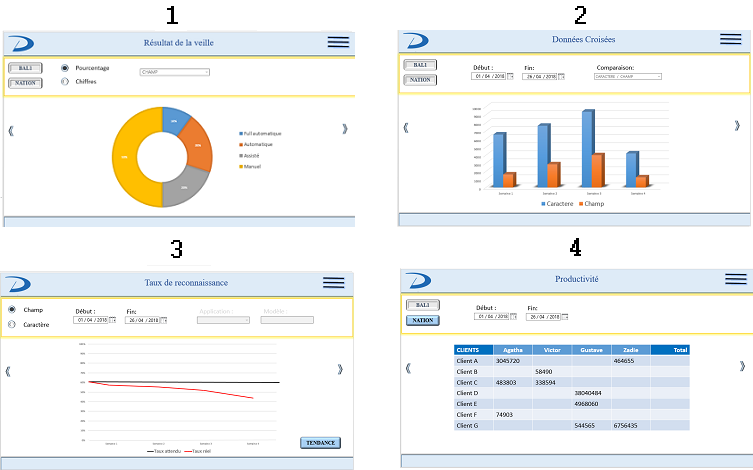
* Page de connexion à l’entrée de l’application qui a pour objectif de vérifier les identifiants des utilisateurs et de récupérer leur rôle.



* Une fois connecté, la page d’accueil apparait avec ses 5 boutons de redirection.

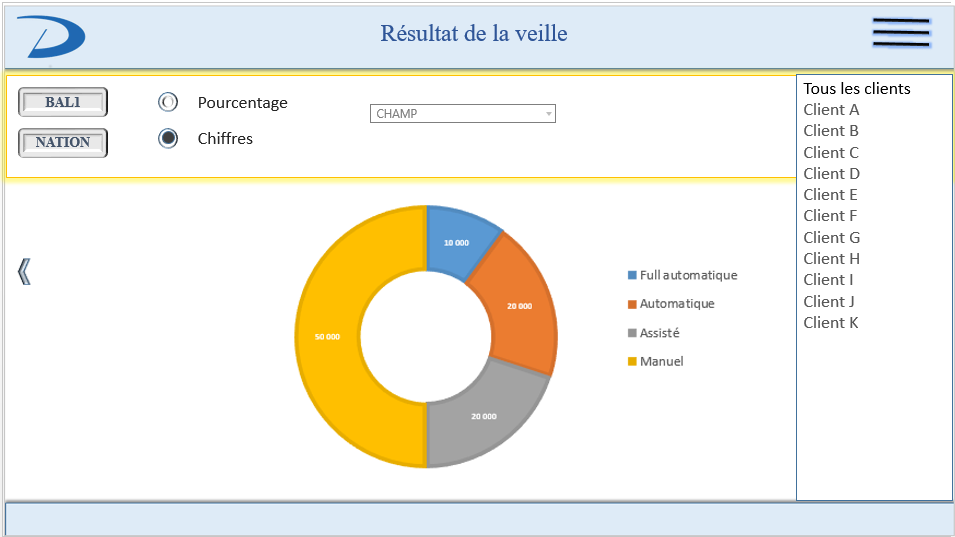


* Mosaïque des quatre vues principales :



Tous ces graphiques ont des données par défaut lorsque vous entrez dans l’application. Ces données par défaut dépendent selon la vue choisie mais un point commun les lient, elles portent sur TOUS les clients.

Si l’on souhaite avoir ces mêmes données mais pour un seul client, il faut cliquer sur le menu burger et sélectionner le client ciblé dans la SideNav.



Aussi, chaque vue est équipée d’un encadré orange munis d’options à sélectionner ou désélectionner. Cela permet de personnaliser les graphiques en fonction de nos recherches. Leurs valeurs par défaut :

1. Résultat de la veille : consiste à visualiser le flux de champs traités la veille pour tous les clients, sur les deux sites de production de façon à dissocier le traitement automatique du traitement manuel en pourcentage.
2. Données croisées : Démontre le ratio entre le nombre de caractères saisis par rapport au nombre de champs traités en vidéocodage pour le mois en cours, ce qui indique la densité de travail pour cette période.
3. Taux de reconnaissance : Affiche une courbe d’évolution du pourcentage de reconnaissance pour le mois en cours et ainsi mesurer l’efficacité du Workflow de Docupost selon les évolutions apportées aux applications. Pour obtenir le pourcentage de reconnaissance, je prends, le nombre de champs reconnus  multiplier par 100 et je divise par le nombre total de champs numérisés. Exemple sur le calcul pour comparer 2mois de production, ayant ajouté lors de ce deuxième mois, un nouveau modèle à numériser pour le client A :

Nbr de champs reconnus = 5000 / Nbr de champs numérisés = 10 000

5000 x 100 / 10 000 = 50%

Cela me donne un premier point sur le graphique à hauteur de 50%, puis pour le deuxième mois :

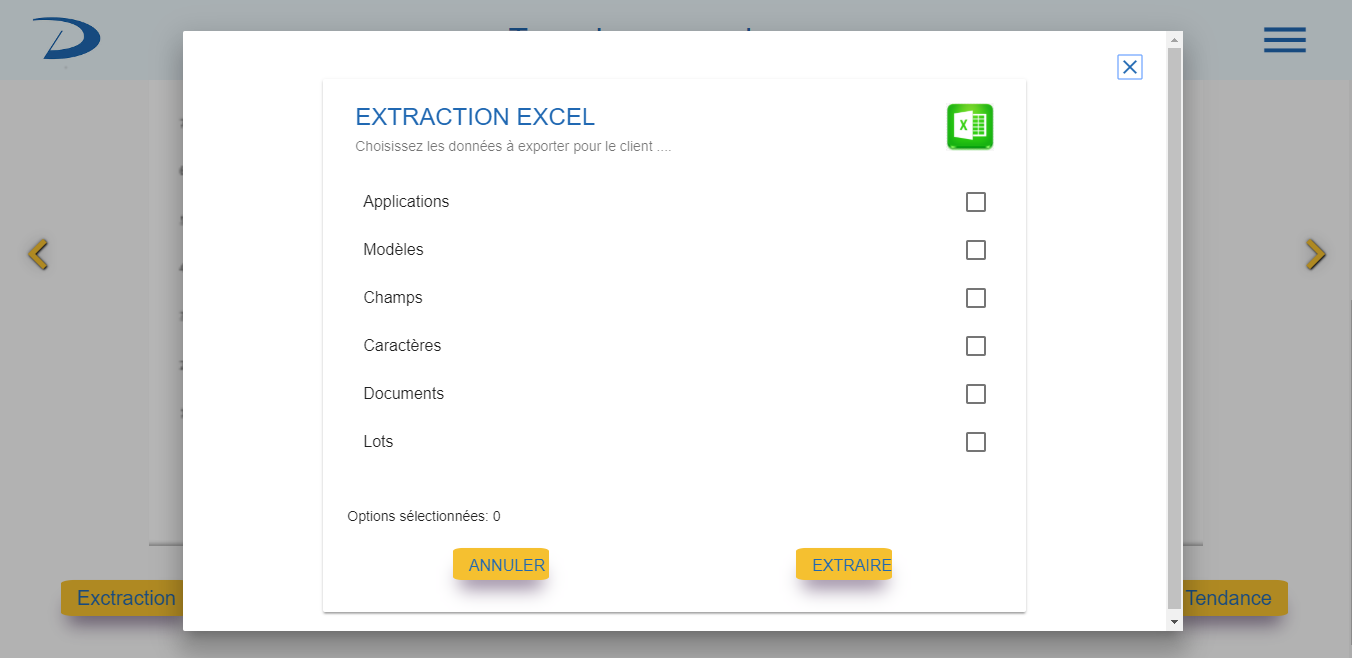
Nbr de champs reconnus = 5000 / Nbr de champs numérisés = 12 000

5000 x 100 / 12 000 = 41,6%

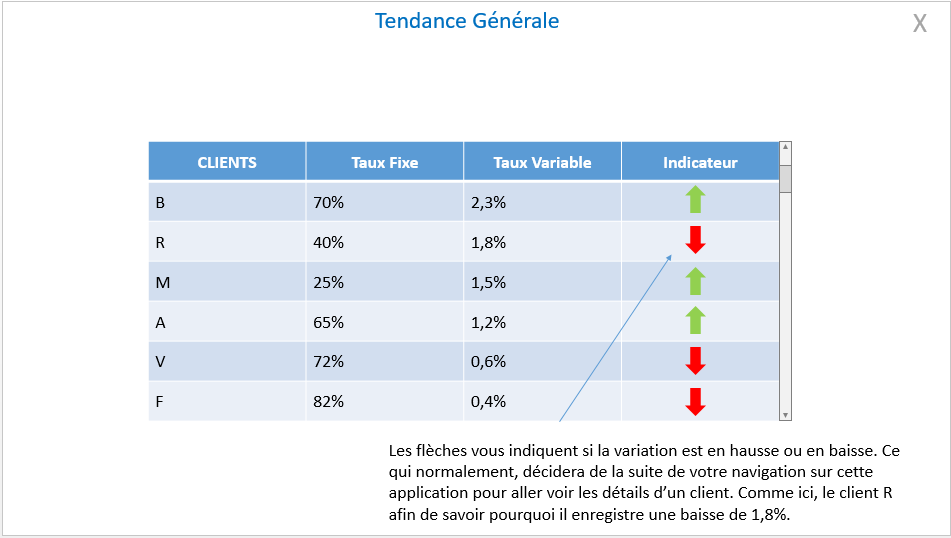
On constate que depuis l’arrivée de ce nouveau modèle pour ce client, la courbe de reconnaissance est en dérive, ce qui indique que le référentiel de Docupost manque de référencement pour ce modèle, qui par conséquent, va envoyer ces modèles non reconnus aux opérateurs de vidéocodage.

1. Productivité : Affiche le nombre de caractères saisis par un opérateur pour chaque client. On sait donc quel opérateur travail sur le client A et pouvons déduire que lorsque le nouveau modèle de ce client sera mis en place, il faudra former un nouvel opérateur.

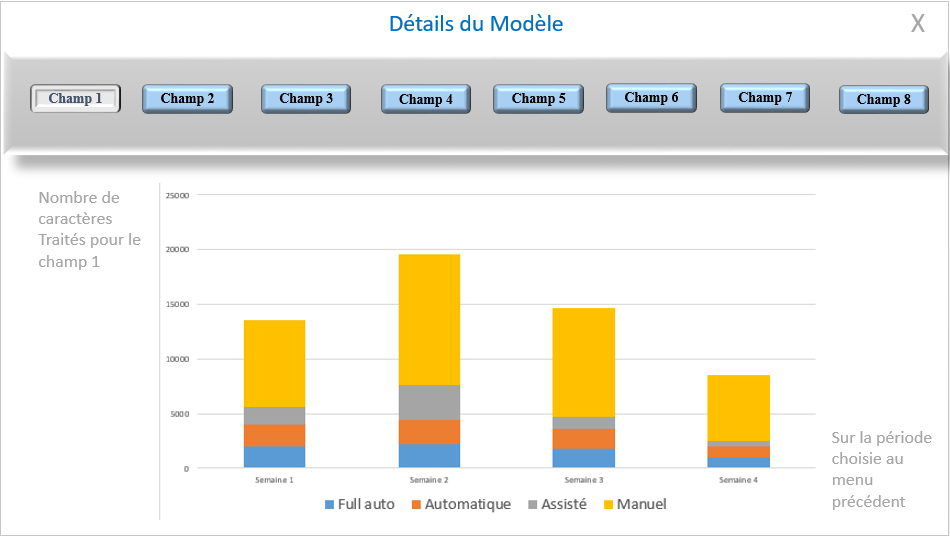
Depuis les vues numéro 1, 2 et 3, un bouton « Extraction » est actif après avoir sélectionné un client. Celui-ci redirige vers une fenêtre listant les informations de production de ce client. On sélectionne les informations à extraire ce qui va générer un fichier csv pour manipuler ces données dans Excel. Cette fonctionnalité a été rajoutée après conception de la maquette, voici une capture d’écran de l’interface en cours de réalisation.



Le bouton « Tendance », qui se trouve dans la vue numéro 3, ouvre aussi une fenêtre et n’apparait que lorsque les données portent sur tous les clients, sinon il se transforme en « Détails ».



Le bouton « Détails » permet de cibler les champs des modèles Docupost ayant un pourcentage de reconnaissance bas, faisant chuter le taux global pour un client. Ce graphique démontre la façon dont ont été traités ces champs (automatique, manuel…) pour la période sélectionnée sur la vue précédente.



**Gestion de projet**

## Méthodes agiles

## Pour obtenir les informations essentielles au développement, il est nécessaire d’avoir une communication bien huilée avec les futurs utilisateurs du produit. J’ai donc eu l’occasion de travailler en méthodes agiles\* avec mes collègues.

## \*Les méthodes Agiles sont des pratiques qui servent à piloter et réaliser un projet. Celles-ci reposent sur les valeurs fondamentales visant tout d’abord à placer l’humain au centre de la conception.

## C’est en pensant ainsi qu’il est jugé primordiale d’obtenir une interaction permanente entre tous les acteurs du projet, qu’ils soient utilisateurs, collaborateurs ou développeurs de la solution. L’évolution même du projet réside dans ces interactions.

## En prenant en compte les idées, les propositions et les demandes de chacun d’eux, plutôt que de se référer à des documentations lourdes qui définissent souvent un cadre restrictif, les changements ici, sont gérés de façon plus souple.

## Que ce changement soit des fonctionnalités à ajouter, des technologies à utiliser ou encore de la gestion de personnel manquant, c’est bien la communication qui permettra sont acceptation pour continuer le travail sans encombre.

## Cette liberté accordée par ces méthodes agiles n’a de sens que par le cadre qui la contient. Encore une fois, c’est bien l’humain de le définir. Pour que cela fonctionne, un cadre est mis en place en donnant un rôle à certains de ces acteurs, il existe un Product owner (propriétaire du produit) qui fera le lien entre le client et les développeurs tout au long de la conception. Le Scrum master lui, organise (planifie) l’avancement du projet en répartissant les tâches et en prenant en compte les retours du Product owner ainsi que l’état moral de l’équipe. Puis les développeurs de proposer, selon leur expérience, de quoi améliorer la productivité du projet.

## Enfin, pour assurer ces pratiques, des outils sont mis à disposition comme ceux que je m’apprête à définir.

## Lync

## 

## Définition :

## Lync est un logiciel interne à DOCAPOST qui permet aux utilisateurs d’envoyer des messages instantanés aux membres du groupe. Il est aussi possible de partager l’écran ou encore passer des appels vidéo.

## Exemple :

## 

## Trello

## Définition :

## Trello est un outil de gestion de projet en ligne. On y créé une liste de tableaux dynamiques qui permettent de visualiser l’état d’avancement d’un projet (sur des colonnes) et de décrire et répartir les tâches (sur des post-it ou cartes). Ces cartes se déplacent par drap and drop (glisser-déposer) dans les colonnes pour indiquer leur état.

## Exemple :

## 

## GitLab

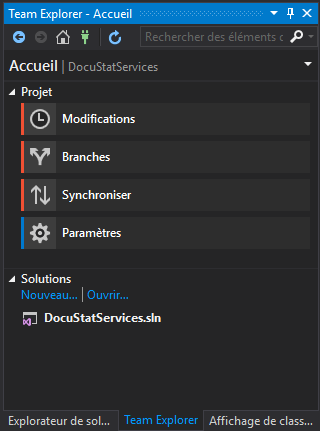
## Définition :

## GitLab est une plateforme (professionnelle dans notre cas) qui permet de déposer un projet et y ajouter les modifications apportés par chaque développeur travaillant sur celui-ci. Un développeur désigné pourra alors gérer les conflits potentiels sur des modifications qui impactent une même fonctionnalité. Toutes modifications apportées par un développeur étant recensées dans l’historique du projet, il devient facile d’effectuer une manœuvre arrière en cas de problème ou de versionner au fur et à mesure des évolutions.

## Exemple :

## 

J’ai la possibilité d’ajouter mes modifications au projet avec les lignes de commande ou directement via une petite interface dédiée dans les IDE : Visual Studio pour le WEB Service et VS Code pour l’IHM :



**RÉALISATION DE L’APPLICATION**

# Vie du projet

# Contexte de travail

# Le cahier des charges présenté dans le chapitre du même nom, est ma retranscription de la demande de mon tuteur. En effet, après m’être imprégnée du sujet : du cycle de numérisation d’un document, mon tuteur, directeur du pôle dématérialisation, a exprimé son besoin d’obtenir une application pour interpréter les données de ce cycle afin d’en améliorer la productivité.

# Il m’a laissé l’opportunité de faire des propositions via la modélisation de l’application, ce qui nous a donné un support pour échanger sur certains aspects de conception.

# Suite à cela, j’ai diffusé mon premier Mockup animé aux futurs utilisateurs de l’application. Prenant leurs remarques en compte, un second Mockup fut partagé en indiquant qu’il serait celui sur lequel je commencerai à développer.

# Analyse des données

# Au lancement du projet, j’ai travaillé sur des bases de données existantes. Celles-ci proviennent de l’entreprise SERES basée à NANTES. Je n’ai donc aucune possibilité de retoucher leur format.

# Elles sont toutes différentes selon les versions de Docupost. Certaines de leurs tables ont un nommage de colonnes peu explicite.

# N’ayant pas la possibilité de contacter l’entreprise SERES durant ma formation, j’ai produit divers fichiers répertoriant les difficultés de lecture pour les soumettre à mes collègues expérimentés et ainsi nous réunir et mettre au point l’approche la plus juste.

# D’après ces analyses, j’en ai déduit les premières fonctionnalités de ce projet et commencé les UML et la maquette de l’interface. Mais aussi, nous avons décidé de mettre en place un moyen de réunir les bases de données existantes en planifiant cette approche de conception :

# 

b-3

b-2

b-1

# *b-1) Programme WinForm en C#*

# Ce programme sera réalisé sous l’IDE (Integrated development environment) Visual Studio.

# Il aura pour fonction de rassembler les données de chaque table, de chaque version DOCUPOST dans la nouvelle base de données stat\_num. Il permettra de récupérer ces valeurs déjà calculées si nécessaire.

# *b-2) Web Service C# WCF*

# Le Web Service est implémenté dans la technologie WCF car cette technologie permet d’exposer des méthodes via une interface utilisateur. D’autres applications peuvent alors l’appeler pour récupérer les méthodes qui y sont implémentées, comme des cuisiniers qui se servent d’un même appareil offrant les mêmes fonctions pour réaliser divers recettes.

# Le pattern (architecture) d’un Service Web WCF est décrit un peu plus bas.

# *b-3) Interface avec ANGULAR en Typescript*

# J’ai décidé de réaliser cette interface avec Angular pour monter en compétence sur ce framework que nous avons appris à utiliser en formation.

# Mise en place de la base de données

# Ayant modélisé ma base de données d’après le diagramme des classes comme expliqué dans le chapitre des spécifications fonctionnelles. J’ai récupérer le script généré par SQL Workbench pour le lancer dans SQLyog, SGBDR interne.

# Lancement du script :

# 

# Une fois la base crée, il faut lui injecter ses données. Cette base est vouée à évoluer, comme le reste de l’application car elle peut encore être optimisée lorsque le contact avec SERES sera établie.

# Toutefois, le premier programme Winform n’étant pas encore créé, j’ai demandé d’obtenir des données cohérentes à injecter manuellement dans SQLyog afin de tester mes méthodes.

# Langages utilisés

# SGBDR :

# SQL (sigle de *Structured Query Language*, en français => langage de requête structurée) est un langage normalisé servant à exploiter des bases de données relationnelles.

# Back-end :

# C# est le langage de programmation orienté objet créé par Microsoft. Il fait partie du Framework .NET. Ce Framework est dédié à la réalisation d’application Microsoft.

# 

# Front-end :

# Typescript a été coécrit par le principale inventeur de C#. C’est un langage libre et open source qui a pour but d’améliorer et de sécuriser la production de code Javascript. Il permet un typage statique optionnel des variables et des fonctions, la création de classes et interfaces tout en conservant l’approche non-contraignante de Javascript. Le code Typescript est transcompilé en Javascript.

# La Programmation Orientée Objet

# Mon programme utilise le paradigme de programmation orienté objet. Cela consiste à créer des briques logicielles appelées « objets » représentant les données à manipuler. Il est possible de définir un élément dit « objet », dans une classe, depuis n’importe quel concept ayant des attributs pour le définir.

# La manipulation de ces objets est structurée dans un système d’héritage et de polymorphisme.

# Héritage

# Le principe de l’héritage, en programmation, s’identifie par :

# La création d’une nouvelle classe qui hérite des propriétés de sa classe mère.

# De cette classe enfant, créer d’autres classes enfants qui en héritent.

# Les propriétés de ces classes doivent être soumises à un champ de diffusion en fonction cet héritage par des mots clés (public, protected, private…..).

# Polymorphisme

# Le polymorphisme complète l’héritage. Il permet de manipuler des objets sans connaitre leur type. On peut donc donner à une même méthode, plusieurs signatures pour implémenter des comportements différents selon les types des paramètres passés.

# Environnement de travail

# Un IDE est un logiciel qui permet de coder. Grâce à son éditeur de texte intégré, nous serons en mesure de modifier et compiler le code source et générer la solution qui sera déployée sur le server.

# De plus, on peut y ajouter des librairies (bibliothèque logicielles) que l’on peut définir par des bouts de programmes redondants déjà conçues, pour éviter de multiplier les mêmes efforts, que l’on va intégrer dans nos programmes. D’autres fonctionnalités s’ajoutent à cela, comme la possibilité de déboguer notre programme.

# Visual Studio 2015

# Pour le Back-end, je me suis servie de Visual Studio 2015. Version payante de l’IDE conçu par Microsoft. On y code en langage C# (entre autre) et les librairies qu’on y importe sont nommées DLL (bibliothèques redistribuables).

# Visual Studio Code

# Pour le Front-end, j’utilise Visual Studio Code, à ne pas confondre. IDE gratuit qui me permet de coder en Typescript. Il est extensible est permet d’importer des modules.

# Réalisation du Service Web WCF

# Le Web Service a pour objectif d’interroger la base de données (stat\_num) en établissant une connexion à celle-ci pour récupérer les données d’après les fonctionnalités de l’application ; ceci de façon structurée. On délivrera alors un accès aux méthodes d’agrégation de ces données par le biais d’URL..

# La Structure WCF

# Lors de la création d’un nouveau programme dans l’IDE, j’ai choisi, comme convenu, qu’il soit un service Web WCF. En faisant ce choix, Visual Studio importe un squelette du pattern (architecture) WCF.

# WCF = Windows Communication Foundation

# A l’origine, la fondation de communication de Windows (Microsoft) a pour principe d’ajouter une couche au programme qui permettra de communiquer avec d’autres applications via une interface utilisateur.

# Je vais m’en servir pour programmer mon Service, ce qui permettra à d’autres types d’interface utilisateurs de l’utiliser. Alors comment définir cette couche supplémentaire ?

# a-1) L’interface du service

# En « décorant » les classes et interfaces pour les définir en tant qu’éléments contribuant à cette communication.

# 

Décoration qui détermine que nous somme dans l’interface du service

Décoration qui détermine une méthode du contrat du service

# Dans cette interface, on y dépose les méthodes dont notre application aura besoin, en se référant aux Use case. La décoration [WebInvoke], que l’on aperçoit dans l’exemple ci-dessus, contiendra le verbe (ici « GET ») qui détermine la façon dont on va agir sur les données, et l’adresse (l’URL) à laquelle elle peut être appelée.

# Par exemple, pour la méthode qui renverra la liste des utilisateurs, ici GetUsers(), il faudra interroger l’URL suivante : http://localhost:50480/Service.svc/users

# La partie orange est obligatoirement placée en préfixe de chaque URL. Ce chemin est déterminé dans le fichier de configuration du service.

# a-2) Cheminement

# Les méthodes du contrat dans l’interface sont perçues comme l’objectif à atteindre. Dès que ces méthodes sont testées via Postman, je peux nourrir mon application. Je vais donc avoir besoin d’obtenir des données de la base stat\_num et pour cela, de m’y connecter.

# La connexion à la base

# b-1) Interface IDataAccess

# Pour implémenter les méthodes de l’interface IService, il faut récupérer des données sur lesquelles travailler. Pour cela, je dois créer une deuxième interface qui expose les méthodes dédiées à l’interaction avec la base de données. Elles ont un nommage normalisé pour les reconnaitre, elles se terminent toutes par FromDB() :

# 

# b-2) Les classes (modèles)

# Ces méthodes doivent retourner des objets dont les propriétés ont été définies dans des classes. Si on prend la méthode :

# List<Utilisateur> GetUsersFromDB();

# Cette méthode aura pour but d’aller chercher la liste de tous les (objets) utilisateurs dans la base de données. Voici la classe utilisateur sur laquelle se repose la définition d’un utilisateur :

# 

Exemple d’attribut de la classe

Décoration qui détermine que c’est une propriété du modèle

Décoration qui détermine que cette classe est un modèle

# b-3) Classe DataAccess

# Ayant les objets et le nom des méthodes, je créé une classe, qui n’aura pas l’objectif de définir un objet mais, d’implémenter les méthodes de l’interface IDataAccess.

# 

# L’implémentation des méthodes commencera toujours par une connexion à la base de données. Pour cela, j’ai stocké dans des variables, comme on le voit ci-dessus, les éléments nécessaires à une connexion, à savoir : l’adresse IP du server, le nom de la base de données, l’identifiant et le mot de passe.

# Ces variables sont sécurisées, elles font appel au fichier de configuration qui détient les informations :

# 

# Voici un exemple de méthode du DataAccess :

# 

5

4

7

6

1

2

3

# Une variable qui contiendra le résultat et une variable qui contient les informations de connexion en paramètre.

# Dans le try => catch qui test la méthode et renvoi un message d’erreur en cas d’échec :

# Variable qui contient la requête SQL pour obtenir la liste des objets utilisateurs.

# Ouverture de la connexion.

# Variable qui contient toutes les données telles qu’elles existent dans le SGBDR.

# Itération sur chaque ligne de données pour les structurer en associant les attributs d’un utilisateur aux colonnes de la table.

# Ajout de chaque utilisateur à la liste.

# 7- Fermeture de la connexion et retour du résultat.

# b-4) La classe Service

# Elle implémente des méthodes de l’interface IService (avec leur chemin URL assicié) en prenant les données que l’on vient de récupérer depuis le DataAccess.

# Pour la liste des utilisateurs, ma méthode du Service consiste à appeler la méthode du DataAccess.

# 

# Mais c’est bien dans le Service que je peux manipuler les données. En effet, je peux par exemple, affiner une méthode pour obtenir un nouveau résultat avec les expressions Lambda, langage Linq. C’est en fait un langage très proche du langage SQL puisqu’il utilise les mêmes mots d’actions.

# Exemple :

# Si je souhaite obtenir les utilisateurs ayant pour rôle « opérateur », je me sers de la même connexion à la base de données par la méthode GetUsersFromDB() mais j’y inclue une expression Lambda qui saura dissocier le rôle des utilisateurs.

# 

# Dans ce cas précis, il aurait été préférable de dissocier le rôle de l’utilisateur par la requête SQL dans une méthode du DataAccess. En effet, je récupère ici Toutes les données d’une table, pour finalement me servir d’une seule partie d’entre elle.

# Je me servirai du langage Linq, plutôt pour dissocier deux éléments à l’intérieur d’une méthode pour les intégrer à son type de retour, comme un Dictionary <Key, Value>.

# Un autre aspect pour obtenir une information de façon plus complexe , qui demande des paramètres en entrée => Je souhaite récupérer le nombre de champs reconnus automatiquement par DOCUPOST pour une période qui sera déterminée par l’utilisateur.

# Je vais donc devoir faire une première méthode qui récupère toutes les informations en fonction d’une date de début et une date de fin. Cette méthode est « private » car je l’utilise uniquement à l’intérieur de cette même classe.

# 

# Noter l’expression lambda et le changement de format de la date avant qu’elle soit retournée.

# Maintenant je souhaite récupérer uniquement mon information sur le nombre de champs reconnus. Je vais donc mettre en paramètre à cette méthode, les dates pour déterminer le calcul et faire appel à ma méthode privée pour définir la période.

# 

# Cette méthode fait partie du contrat pour l’UI, je vais pouvoir la tester en l’appelant depuis son URL. Comme mon UI n’est pas encore construite, je me sers du logiciel Postman. Il suffit de lancer la compilation du projet dans l’IDE et de passer l’URL de la méthode avec le bon verbe d’action sur la base :

# 

# 

# J’ai entrée des dates directement dans l’URL car on ne peut pas encore le faire depuis l’interface utilisateur mais on voit bien le résultat retourné est en format JSON avec le nom de la méthode invoquée.

# Permission d’accès à l’interface utilisateur

# Une fois que je sais que mes méthodes fonctionnent via leur URL, je dois donner l’accès à l’IHM Angular qui se lance sur l’adresse locale : http://localhost :4200. Pour cela, je créée une classe Global.asax pour paramétrer les droits d’accès.

# 

# Ainsi que dans le fichier de configuration du programme :

# 

# Réalisation de l’UI avec Angular

# L’IHM (Interface Homme Machine) de mon projet a été réalisé avec Angular. L’objectif est de concrétiser les wireframe de mon Mockup animé.

# Angular-cli

# Angular est un framework Javascript créé par Google et sert à créer des pages Web dynamiques en utilisant le principe de développement « single page » = plus de temps de chargement de page.

# Angular-cli signifie *Command Line Interface* (interface en ligne de commande). En effet, l’interaction avec la machine se fait par les lignes de commandes extérieures ou intégrées dans l’IDE, qui sera ici, Visual Studio Code.

# C’est par cette pratique que l’on va pouvoir ajouter du contenu (composants, modules, services…) et le modifier avec le langage balisé HTML pour structurer sémantiquement nos pages, les stylisées avec CSS puis les enrichir de fonctionnalités avec le langage Typescript (Javascript typé).

# Installation

# Pour utiliser Angular, il faut au préalable avoir installé Node.js et npm. Node.js est un environnement qui permet d’utiliser le langage Javascript (ou Typescript) coté server et de l’exécuter très rapidement.

# On peut voir un projet en construction comme une voiture dont Node.js serait le moteur prêt à se lancer, dès qu’on lui en donne l’ordre et à chaque fois qu’on lui ajoute de quoi avancer, comme des roues, ici représenté par des modules. Et c’est npm (Node Package Manager) qui détient ces modules (packages).

# *Lancement de mon projet*

# J’ouvre la fenêtre de ligne de commande depuis mon espace de travail dédié et j’y écris :

# 

# Cela va générer un dossier contenant la structure globale et basique d’un projet Angular nommé « docustat ».

# Structure et Material Design

# Structure Globale

# Décomposition des fichiers importants d’un nouveau projet :

# 

Fichiers qui répertorient de quoi est fait le projet (nom du projet, versions, liste des packages intégrés...)

Dossier qui contient les modules que l’on importe

Dossier qui contient les composants

# Dans le dossier src (source) ci-dessus :

# 

Feuille CSS qui impact l’aspect et organise ce que contiendra l’index.html

Composant (component) de base

Point d’entrée du projet, page htlm basique

# Dans le composant de base (app-component) ci-dessus :

Feuille CSS de ce composant, uniquement

# 

Page html de ce composant, uniquement

Fichier Typescript de ce composant, c’est ici que l’on code pour rendre sa page html dynamique

Module qui répertorie tous les imports nécessaires à la construction du projet. Qu’ils soient de nature Component, Service, Material… ; il est essentiel qu’ils y paraissent TOUS sous peine de ne pas pouvoir lancer le projet dans le navigateur

# Tous les composants que j’ajouterai par la suite seront, à l’initialisation, identiques à celui-là, sans le app.module.ts ; mais devront être importés dans ce fichier.

# Structure de mon projet

# Tout d’abord, il faut savoir qu’un projet Angular se lance sur le navigateur de notre choix en entrant l’URL : <http://localhost:4200>, via la commande : ng-serve

# Par défaut, un nouveau projet se lance depuis l’Index.html qui contient dans son <body>, la balise donnant accès au composant de base : app.

# Cet appel, comme tous les autres ensuite se fait ainsi : Dans des balises qui indiquent le nom défini pour le composant appelé. Ce nom se récupère dans le fichier Typescript du composant à afficher (ici : app Component.ts) :

# 

# Et c’est justement depuis l’appComponent.html (nettoyé de son code par défaut) que je vais construire l’architecture (Single Page) de mon projet en faisant appel, de la même manière, aux composants qui la forme.

# J’ai décidé, par rapport à mon Mockup, d’avoir sur mon modèle single page, un entête (header) contenant une barre d’outils fixe et le reste de la page (body) mobile durant la navigation. Donc, dans le composant de base (qui est lui-même appelé par l’index.html) j’insère les appels de ces deux composants à l’origine de ma structure :

# 

# Ces composants ont été créés à l’aide de la ligne de commande suivante:

# 

# Et contiendront les 4 fichiers de base comme expliqué ci-dessus

# Arborescence de mon projet

# Rappel : Tous les fichiers créés, le sont depuis la racine du composant de base et doivent être importés dans son module.ts. Je vous montre donc l’arborescence après le nœud « app » :

# 

COMPOSANTS

Exemples de services.

Api = service général;

Taux-reco = service spécifique au composant taux-reco

Exemple d’un composant ouvert pour montrer les 4 fichiers qui le forme.

Composant Header qui contient la barre d’outils.

Fichier Typescript contenant mes classes (modèles)

# Pour améliorer cette arborescence, je pourrai regrouper dans des dossiers tous les Services, par exemple. Si je m’exécute, il faudra alors modifier tous les chemins dans les imports qui font appel à eux.

# Les imports et modules Material

# *Imports de composants*

# Pour que les fichiers communiquent entre eux, il faut importer celui détenant l’information (le besoin) dans le composant qui l’utilisera.

# Par exemple, mon composant qui correspondant à la vue qui affiche une courbe d’évolution du Taux de reconnaissance (décrite dans les spécifications fonctionnelles de ce dossier) à un bouton d’extraction qui, lorsque l’utilisateur clique dessus, affiche la fenêtre Popup pour extraire les données d’un client dans le format Excel (csv).

# 

Clique sur le bouton

La fenêtre Popup s’ouvre

# Cette Popup est donc soumise à l’évènement « clique » du bouton qui est dans le composant taux-reco (voir arborescence).

# Le composant *taux-reco-component* a besoin du composant *popup-extraction-component,* donc dans le fichier *taux-reco-component.ts*, je vais insérer l’import du composant de la Popup :

# 

# *Modules Material Design*

# Material Design est une bibliothèque en ligne de modules déjà fabriqués dont on peut se servir, simplement en appelant l’import associé, dans le composant de notre projet qui le contiendra.

# Son adresse : <https://material.angular.io>. On y trouve des composants Javascript facile à intégrer dans Angular, avec la documentation nécessaire pour les incorporer et les modifier.

# *Imports de modules*

# Un exemple avec mon projet. Pour inclure la SideNav (dans le header, pour lister le nom des clients), j’ai récupéré l’import associé sur le site:

# 

# Et inclue dans le *header-component.ts* puis, toujours avec la documentation du site, j’ai pris le code HTML et CSS de ce module que j’adapté dans mon projet :

# 

# On peut voir d’ailleurs qu’il m’a fallu rediriger la balise de mon composant Body à l’intérieur de la SideNav car, ce module à besoin d’entourer ce surquoiil va agir*. Je clique sur le menu Burger pour la faire apparaitre :*

# 

Menu Burger

SideNav

Composant Body

# Routing

# J’ai fait le choix de faire défiler les différentes vues de mon projet, dans le composant Body. La navigation entre elles se fait, comme un carrousel, par des flèches de chaque côté.

# Pour que les vues défilent dans l’espace alloué au composant Body, j’y ai inséré une balise dédié :

# Ce router-outlet interroge le fichier de routing, c’est-à-dire les routes (chemin URL) que j’ai défini pour chaque vue :

# 

Chemins URL de chaque composant

Imports des composants dont j’ai besoin pour définir les routes

# Maintenant, dans les balises html de mes flèches du carrousel, je dois indiquer quelle est la vue à atteindre quand l’utilisateur clique dessus. Voici le code HTML de l’une d’entre elles :

# 

# Le routerLink = « /cross » renvoi l’utilisateur à la vue créée dans le composant Données-Croisées.

# Graphique

# Dans la documentation du site d’Angular Material, aucun graphique n’est proposé. Mais en cherchant sur internet, des tutoriels démontrent qu’il est possible d’en importer depuis, par exemple <https://www.chartjs.org/>

# Ce site internet propose une bibliothèque de graphique conséquente, en Javascript. Ce qui signifie que c’est compatible avec Angular. Le principe est le même qu’avec Material. Le code HTML et Javascript est accompagné d’une documentation pour intégrer un graphique dans nos pages. Il suffit d’adapter ce code à mon projet Angular.

# Pour commencer, je vais charger le package des graphiques. Pour cela, je tape dans ma ligne de commande :

# Ensuite, comme d’habitude, je signale dans le .ts du composant qui va recevoir le graphique, que je vais utiliser cette bibliothèque avec :

# 

# Dans ce même fichier, et comme dans tous les composants.ts à leur création, il existe une méthode d’Angular par défaut. C’est la méthode ngOnInit(). Tout le code inscrit à l’intérieur de cette méthode sera exécuté à l’initialisation du composant (son appel dans l’interface). C’est donc dans cette méthode que je vais écrire le code du graphique que je souhaite faire apparaître. Je déclare une variable qui va contenir le graphique :

# 

# Puis j’appelle cette variable dans la méthode ngOnInit de mon composant en lui passant le code du graphique avec des données fictives (en dur) :

# 

Données fictives pour tester si le graphique fonctionne.

# Et enfin, je vais utiliser l’interpolation (data binding => fichiers d’un même composant qui communiquent entre eux) dans le code HTML, pour appeler ce graphique :

# 

# Je lance mon projet avec la ligne de commande : ng-serve, pour vérifier si le graphique apparait bien dans l’interface :

# 

# Dernière étape, donner les valeurs que je récupère dans les méthodes de mon Web Service WCF.

# Connexion avec le programme, services Angular

# Pour récupérer les valeurs de mes méthodes depuis le Web Service, je vais établir la connexion dans un fichier « service » d’Angular.

# Je vais donc créé ce fichier depuis les lignes de commande :

# 

# La dernière partie de la commande est une instruction qui demande à Angular de placer directement l’import de ce nouveau fichier dans l’app.module.ts, puisque c’est indispensable pour faire fonctionner chaque nouveau fichier.

# Je créé un autre service, cette fois ci de façon plus général (pour qu’il diffuse ses informations à tous les composants), que je vais appeler ApiService. C’est dans ce service que je fais appel à toutes les méthodes du Back.

# Un service général sécurise l’interface, car si je supprime la vue en lien avec son service, j’aurai toujours l’appel de la méthode dans le service général pour m’en resservir. Aussi, si je me sers d’une même méthode dans deux composants différents.

# Je créée la méthode qui va récupérer la valeur du nombre de champs reconnus par l’outil DOCUPOST sur une période donnée.

# Dans le Back, cette valeur est récupérable à l’adresse suivante :

# 

# Je vais donc faire appel à cette adresse dans l’interface en utilisant le module HttpRequest pour écouter les ports.

# J’ai besoin d’une variable qui va contenir la partie de l’URL commune à chaque méthode et une variable qui va contenir l’appel du Back, dans ma méthode :

# 

# J’instancie l’ HttpRequest dans le constructeur :

# 

# 

# Et j’écris ma méthode en lui passant des dates fictives pour la tester :

# 

# Dans le service (tauxReco-service) propre à la vue, je déclare une méthode qui va simplement faire appel à celle de l’ApiService. Pour cela, je dois y importer l’ApiService et l’instancier dans le constructeur pour qu’Angular reconnaisse la méthode que je souhaite récupérer :

# 

# 

# Je peux maintenant faire appel à cette méthode dans le code Javascript du graphique. Je déclare une variable qui contiendra la valeur que renvoie ma méthode et j’appellerai cette variable dans l’axe (ordonnées ou abscisses) qui correspond à cette valeur :

# La variable :

# L’appel de la méthode :

# 

# L’insertion des variables dans le graphique :

# 

**CONCLUSION**

Ce projet a été et sera encore un moyen pour moi de progresser.

En effet, en me confrontant à la réalisation d’une application, de sa conception à l’interaction de chacun des éléments qui la constitue, je réalise à quel point j’ai pu consolider mes connaissances mais aussi et surtout, apprendre de nouvelles notions.

Les difficultés rencontrées durant le développement de mon projet ont à chaque fois apporter un peu plus à mon bagage professionnel dans le domaine de la programmation et compréhension des enjeux de Docapost.

J’en retiens principalement l’importance de créer des bases de données lisibles.

Je souhaite pouvoir aller jusqu’au bout des capacités de mon Web Service. Et pourquoi pas intégrer une nouvelle forme de base de données, en No SQL (*not only*) pour gérer la densité d’informations agrégées depuis toutes les bases existantes.

Il est important de souligner que j’ai beaucoup appris durant cette formation mais que c’est une infime partie du monde de l’informatique. J’y ai appris aussi à m’auto-former, ce qui me permettra de progresser continuellement et me spécialiser dans certains langages.

Enfin, je suis contente d’avoir suivie ce cursus de reconversion et partagé cela avec des personnes passionnées.

.