

Concepteur Développeur d’Applications

Application de gestion de factures

Table des matières

[I. Liste des compétences 3](#_Toc104671909)

[II. Introduction 4](#_Toc104671910)

[a. Résumé 4](#_Toc104671911)

[b. Abstract 5](#_Toc104671912)

[III. Présentation 6](#_Toc104671913)

[a. Gnosys 6](#_Toc104671914)

[b. Le projet 6](#_Toc104671915)

[c. Les technologies utilisées 7](#_Toc104671916)

[a. Les Frameworks 7](#_Toc104671917)

[b. Les langages de programmation 7](#_Toc104671918)

[c. Les outils utilisés 8](#_Toc104671919)

[IV. La conception 10](#_Toc104671920)

[a. Analyse des besoins 10](#_Toc104671921)

[b. Analyse de l’existant 10](#_Toc104671922)

[c. Organisation du travail 10](#_Toc104671923)

[d. Le cahier des charges 13](#_Toc104671924)

[e. Le dictionnaire de données 16](#_Toc104671925)

[f. Le MCD 17](#_Toc104671926)

[g. Le MLD 19](#_Toc104671927)

[h. Diagrammes UML 20](#_Toc104671928)

[Le diagramme de cas d’utilisation 20](#_Toc104671929)

[Le diagramme de séquences 21](#_Toc104671930)

[i. Wireframes 23](#_Toc104671931)

[j. Maquettes en couleurs 24](#_Toc104671932)

[V. Le développement 27](#_Toc104671933)

[a. Présentation de l’arborescence 27](#_Toc104671934)

[b. Modèle MVC 28](#_Toc104671935)

[c. Connexion 29](#_Toc104671936)

[d. Gestion des rôles 30](#_Toc104671937)

[e. Routing 31](#_Toc104671938)

[f. Filtrage 32](#_Toc104671939)

[g. Collection type 34](#_Toc104671940)

[h. Envoi de mail 36](#_Toc104671941)

[i. Templates twig 37](#_Toc104671942)

[j. Mise en place d’un APIRest 40](#_Toc104671943)

[k. Application CSharp 42](#_Toc104671944)

[l. Application Android 48](#_Toc104671945)

[L’arborescence du projet 48](#_Toc104671946)

[VI. Problème rencontré 54](#_Toc104671947)

[VII. La veille technologique 55](#_Toc104671948)

[VIII. La conclusion 55](#_Toc104671949)

[m. Stage 55](#_Toc104671950)

[n. Formation 55](#_Toc104671951)

[IX. Les remerciements 56](#_Toc104671952)

[X. Les annexes 57](#_Toc104671953)

# Liste des compétences

|  |  |
| --- | --- |
| Maquetter une application |  |
| Développer une interface utilisateur de type desktop |  |
| Développer les composants d’accès aux données |  |
| Développer la partie front-end d’une interface utilisateur web |  |
| Développer la partie back-end d’une interface utilisateur web |  |
| Concevoir une base de données |  |
| Mettre en place une base de données |  |
| Développer des composants dans un langage d’une base de données |  |
| Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement |  |
| Concevoir une application |  |
| Développer des composants métier |  |
| Construire une application organisée en couche |  |
| Développer une application mobile |  |
| Préparer et exécuter les plans de test d’une application |  |
| Préparer et exécuter le déploiement d’une application |  |

# Introduction

## Résumé

J’ai fait mon stage dans l’entreprise Gnosys, cette entreprise est dirigée par Monsieur CARLIER Pascal depuis le 1er janvier 2017 située dans la somme plus précisément à Dernancourt, couvre différents secteurs d’activités tels que la conception, le développement Web, l’audit, le jury.

Comme tout entrepreneur dès lors qu’il vend des produits/prestations des services etc… il doit faire appel à la comptabilité.

En effet sur le marché, il existe beaucoup d'outils de gestion de factures tels que facture.net, dolibarr, etc … mais ceux-ci n’intègrent pas toutes les fonctionnalités nécessaires aux micro-entreprises, ceux-ci ne correspondant pas à ses attentes, il souhaite donc créer sa propre application de facturation.

Monsieur CARLIER souhaite ajouter quelques fonctionnalités personnalisées en plus afin de mieux gérer sa facturation tout au long de l'année.

Les principaux axes de l’application seront donc :

De faciliter la déclaration des résultats comptables auprès des organismes financiers.

D’effectuer des rapprochements bancaires, plus facilement et plus rapidement.

D’effectuer une relance de facture au client et d’en avoir la traçabilité.

De pouvoir avoir une vue d’ensemble sur les différentes statistiques adaptées aux micro-entreprises telles que le chiffre d’affaires mensuel ou trimestriel, le nombres de factures payées, le nombres de factures impayées, le taux de variation de l’année N-1 à l’année N.

De pouvoir être alerté en cas de dépassement de seuils de chiffre d’affaires imposés aux micro-entreprises.

J’ai utilisé le Framework Symfony pour faire une application sécurisée et le framework CSS Tailwind permettant de gagner du temps dans la réalisation du Front-End.

Les langages utilisés sont :

* PHP
* JS
* SQL
* HTML/CSS

## Abstract

I did my internship in the company Gnosys, this company is led by Mr CARLIER Pascal since 1st of January 2017, located in Somme more precisely in Dernancourt, covers different sectors of activities such as design, web development, audit, training panel.

Like every contractor, since he sells products/services etc... he must resort to accounting.

There are many billings management tools on the market, such as facture.net, dolibarr, etc. but these do not integrate all the functionalities necessary for micro-enterprises, these do not correspond to his expectations, so he wants to create his own billing application.

Mister CARLIER wants to add a few more custom features to better manage his billing throughout the year.

The main axes of the application will therefore be:

- To make easier the reporting of accounting results to financial institutions.

- To make bank reconciliations easier and faster.

- To send a bill reminder to the customer and to have traceability

- To be able to have an overview on the various statistics adapted to micro-enterprises such as monthly or quarterly turnover, the number of invoices paid, the number of unpaid invoices, the rate of change from year N-1 to year N.

- To be able to be alerted in the event of exceeding turnover thresholds imposed on micro-enterprises.

I used the Symfony Framework to make a secure application and the Tailwind CSS framework to save my time in the realization of the Front-End.

The languages used are:

* PHP
* JS
* SQL
* HTML/CSS

# Présentation

## Gnosys

L’auto-entreprise Gnosys Experimental crée le 1er janvier 2017 est dirigée par Monsieur CARLIER Pascal située dans la somme plus précisément à Dernancourt, couvre différents secteurs d’activités telles que la conception, le développement Web, l’audit, le jury.

Pour les secteurs géographiques, il opère dans les Hauts-de-France, Paris, Reims.

## Le projet

Monsieur CARLIER Pascal, est un autoentrepreneur et comme tout entrepreneur dès lors qu’il vend des produits/des prestations des services etc… Il doit faire appel à la comptabilité.

En effet sur le marché, il existe beaucoup d'outils de gestion de factures tels que facture.net, dolibarr, etc … mais ceux-ci n’intègrent pas toutes les fonctionnalités nécessaires aux micro-entreprises, ceux-ci ne correspondant pas à ses attentes, il souhaite donc créer sa propre application de facturation.

Monsieur CARLIER souhaite ajouter quelques fonctionnalités personnalisées en plus afin de mieux gérer sa facturation tout au long de l'année.

Les principaux axes de l’application seront donc :

* De faciliter la déclaration des résultats comptables auprès des organismes financiers.
* D’effectuer des rapprochements bancaires, plus facilement et plus rapidement.
* D’effectuer une relance de facture au client et d’en avoir la traçabilité.
* De pouvoir avoir une vue d’ensemble sur les différentes statistiques adaptées aux micro-entreprises telles que le chiffre d’affaires mensuel ou trimestriel, le nombre de factures payées, le nombre de factures impayées, le taux de variation de l’année N-1 à l’année N.
* De pouvoir être alerté en cas de dépassement de seuils de chiffre d’affaires imposés aux micro-entreprises.

## Les technologies utilisées

### Les Frameworks

**Symfony :**

Symfony est un Framework écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permette de faciliter le développement.



**Tailwinc CSS :**

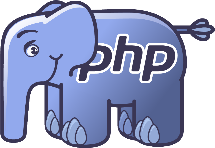
Tailwind CSS est un Framework qui fournit des fonctionnalités afin de faciliter la mise en page web.



### Les langages de programmation

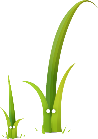
**PHP :**

PHP est un langage de programmation libre principalement utilisé pour produire des pages web dynamique.



**Twig :**

Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le Framework Symfony



**Java Script :**

JavaScript est un langage de programmation de script principalement employé dans les pages web interactives.



**jQuery :**

jQuery est une bibliothèque JavaScript libre et multi-plateforme créée pour faciliter l’écriture de script côté client.

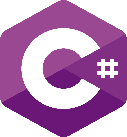


**HTML – CSS :**

HTML est un langage de balisage permettant de construire la structure des pages web. Il est généralement accompagné de CSS, un langage permettant d’appliquer des styles aux éléments HTML.

**C# :**

C# est un langage de programmation objet typé utilisé pour développer des applications desktop, des services web.



**XAML :**

XAML eXtensible Application Markup Language est un langage permettant la mise en place d’interfaces utilisateur. Il complète C# pour le desktop.



**XML :**

XML eXtensible Markup Language est un langage permettant la mise en place d’interfaces utilisateurs. Il peut aussi être une alternative au JSON.



**Java :**

Java est un langage informatique de programmation orienté objet. Dans notre cas, il sera utilisé dans la programmation de l’application Android.



**MySQL :**

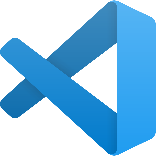
MySQL est un système de gestion de bases de données relationnelles. Il fait partie des logiciels de gestion de bases de données les plus utilisés au monde, autant par le grand public que par des professionnels.



### Les outils utilisés

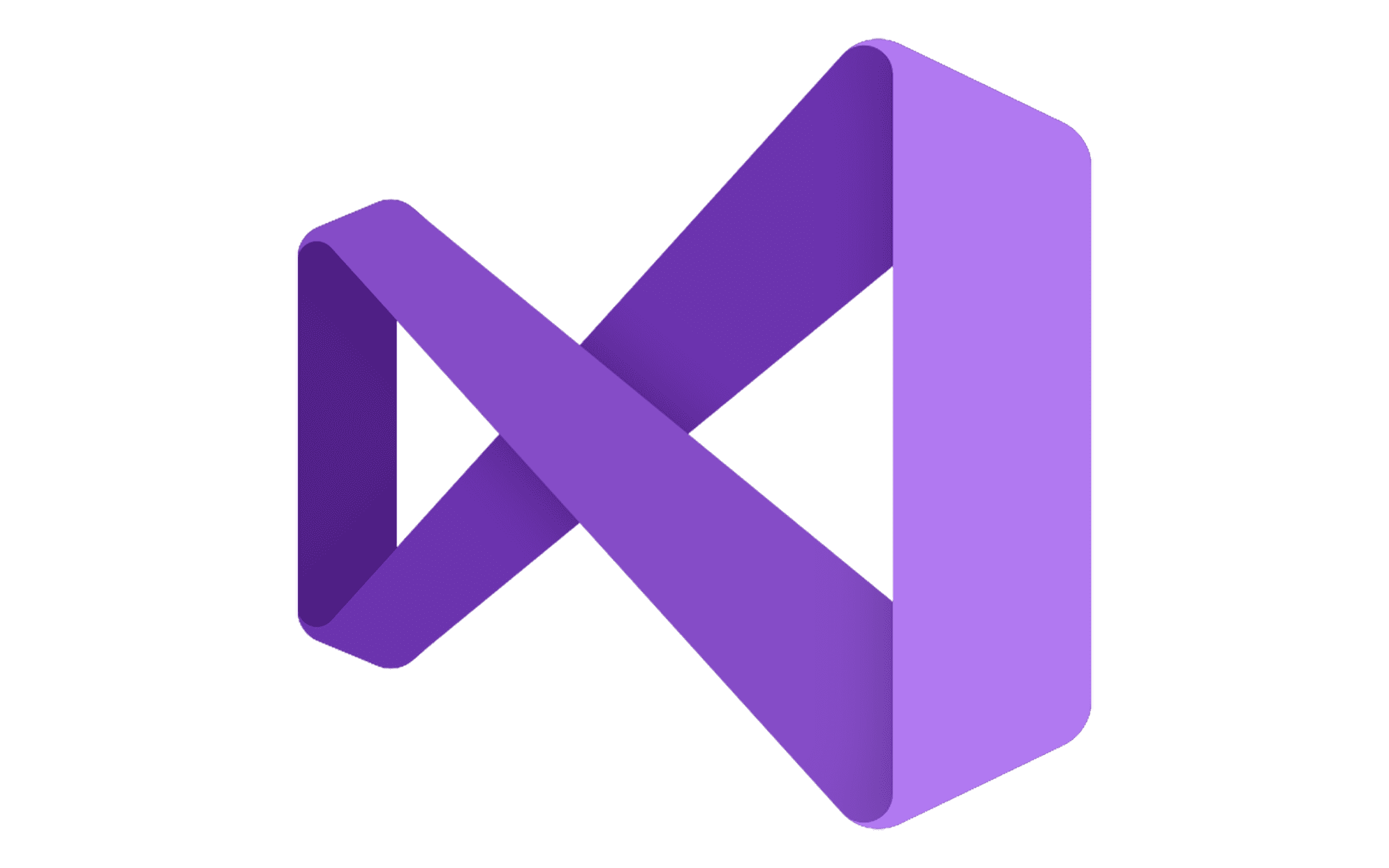
**Visual Studio Code :**

Vs code est l’éditeur de code sur lequel j’ai travaillé quotidiennement pour coder l’application Symfony.



**Visual Studio  :**

VS est un ensemble d’outils permettant de générer des applications desktop, mobile ou des services web.



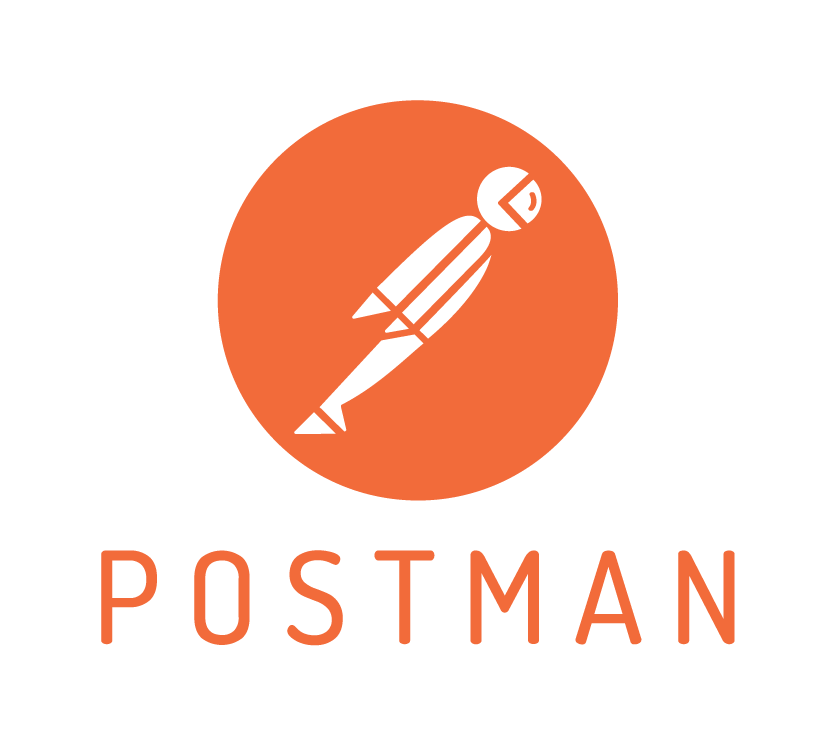
**Android Studio :**

Android Studio est un environnement WYSIWYG de développement pour développer des application android.



**Postman :**

Postman est le logiciel avec lequel j’ai pu tester les requêtes api.



**FileZilla**

FileZilla est un logiciel FTP qui permet le déploiement de l’application sur le serveur.



**Putty**

Putty est émulateur de terminal doublé d’un client pour le protocole ssh. Permet de lancer des commandes sur le serveur.



# La conception

## Analyse des besoins

Dans la conception, l’analyse des besoins est indispensable, c’est même la première étape de la conception et la plus déterminante sur la faisabilité du projet.

À ce moment précis, c’est le plus souvent un rendez-vous avec le client et c’est l’étape qui requiert le plus d’informations possibles afin de comprendre ce que veut réellement le client. On va pouvoir déterminer les coûts approximatifs, le temps de travail pour la réalisation du projet, les technologies à utiliser ou non.

Au cours de cet entretien, la découverte de l’existant peut être faite afin de comprendre le type d’interface souhaite le client et donc faire quelques wireframes avec lui afin de valider le style d’interface.

Elle permet donc après coup de rédiger le cahier des charges afin de pouvoir le faire valider au client et lui faire prendre conscience de ce qu’on a compris et lui faire comprendre ce qui sera mis en place lors du projet.

## Analyse de l’existant

Dans notre cas, l’étude de marché s’est faite en direct avec le client, comme il utilisait déjà une application web pour facturer ses clients, j’ai pu analyser celle-ci avec lui et on a pu discuter des choses qui étaient à reprendre et des choses qui n’existait pas et qu’il voulait voir apparaître sur sa nouvelle application.

Cette étape m’a permis de tester l’application afin de comprendre ce qui était mis en place, mais aussi de voir les inconvénients de celle-ci. On peut en profiter pour savoir si l’interface de l’application déjà existante est convenable pour le client, s’il souhaite réutiliser celle-ci afin de ne pas être perturbé en changeant d’application ce qui permet de gagner beaucoup de temps sur le maquettage.

## Organisation du travail

Afin d’améliorer l’organisation du travail dans ce projet, mettre en place des habitudes était le meilleur moyen de communiquer avec le client. Pour communiquer facilement avec Monsieur CARLIER, nous avons utilisé plusieurs outils de communication à distance :

**Zoom:** Cette application payante que presque tout le monde connaît, crée par Microsoft, n’a plus besoin de faire ses preuves, elle est utilisée par beaucoup d’entreprises et a été encore plus sollicitée durant la crise sanitaire.

Une image contenant texte, personne, capture d’écran, équipement électronique

Description générée automatiquement

**Whereby :** C’est une application web, permettant de faire des réunions sans télécharger de logiciel qui est assez similaire à d’autres outils sur le marché finalement.

Cette application nous a servis, pour des réunions à courte durée et non-formelles.



Avec ces deux outils de communication, on faisait 1 à 2 réunions par semaine, sans compter les mails et les sms envoyés qui sont une façon de communiquer rapidement.

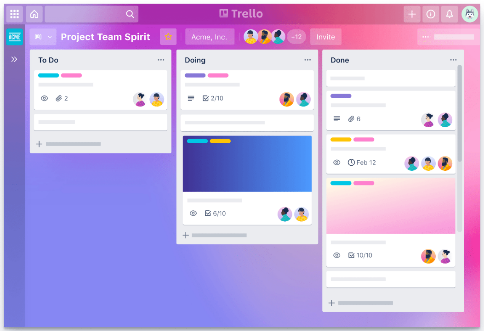
De notre côté, nous avons utilisé plusieurs outils afin d’organiser notre travail sur le projet.

**Trello:** Cette application web et desktop qui permet d’organiser les taches à faire. Il permet d’assurer un suivi permanent sur ces taches.

C’est un outil intéressant pour montrer, ce qui est fait et ce qu’il reste à faire.

Il fait partie de la méthode agile, qui consiste à répartir les tâches à faire du projet et de déterminer qui va prendre cette tâche, afin de ne pas refaire ce que quelqu’un a déjà fait et faire ce qu’il reste à faire.

Le projet est ainsi fragmenté en plusieurs sous-parties que l’équipe qui en a la charge se doit d’atteindre progressivement en réajustant si nécessaire les objectifs pour répondre le plus possible aux attentes du client. Elle permet donc de renforcer les liens entre les membres de l’équipe, en l’occurrence Bruno et moi.



*(vous pouvez retrouver une capture d’écran en annexes)*

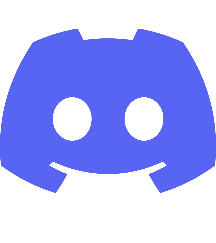
**Git:** Cet outil va simplement faire des versions afin de pouvoir revenir en arrière si besoin, mais il permet aussi de partager les fichiers avec d’autres collaborateurs.



Pour les outils de communication entre Bruno et moi, nous avons décidé d’utiliser **Discord.**

**Discord:** C’est un outil de communication qui a su faire ses preuves en quelques années.

Les fonctionnalités de celui-ci sont différentes des autres outils de communication comme zoom, slack, etc … Il propose de créer un serveur gratuitement afin d’inviter des collaborateurs, amis… et propose de créer des canaux écrits ou vocaux avec les personnes invitées sur le serveur.



Tous les jours, nous étions donc connectés à la même heure, et ce, pour toute la journée.

Ce qui nous à permis de communiquer en direct et de s’aider via des partages d’écrans.

Pour finir, la planification du projet est un élément important pour gérer son temps.

**Gantt Project:** Gantt Project est un logiciel qui permet d’établir un diagramme de Gantt.

Ce diagramme de gantt permet de visualiser le temps que cela pourrait prendre et de voir si nous sommes dans les temps ou non.

Il permet donc d’anticiper sur les taches en cas de manque de temps ou non..

Il peut être faux, mais permet donc de le rectifier et d'estimer le temps le plus optimal par la suite.

L’estimation du temps et importante, elle permet donc d’estimer le coût plus précisément.



*(vous pouvez retrouver une capture d’écran en annexes)*

## Le cahier des charges

*(L’intégralité du cahier des charges de l’application se trouve en Annexe p.)*

Le Cahier des charges est un document très important dans la conception de l’application. Il est rédigé en collaboration avec le client est doit donc être compris par les deux parties. Celui-ci a pour fonction de récapituler les besoins fonctionnels ou non ainsi que les moyens déployés et les conditions de réalisation du projet.

Il débute en plaçant le contexte ainsi que les objectifs de l’application ainsi que la description de l’existant.

On retrouve les critères d’acceptabilités qui définissent les conditions que l’application doit remplir pour être acceptée par le client.

Critères d'acceptabilité du produit

Le produit est acceptable lorsque :

* Toutes les fonctionnalités demandées sont respectées.
* L’expérience utilisateur et l’interface utilisateur seront favorisées.
* L’application sera responsive au minimum sur tablette.

La partie suivante va fixer les différentes contraintes du projet.

* Les contraintes financières recensent les différents achats (matériel, logiciel, serveur etc..).
* Les contraintes de délais indiquent les délais de livraison de l’application et des différentes fonctionnalités.
* Les autres contraintes dans cette partie, on placera toutes les autres contraintes notamment celles liées au RGPD.

Le cahier des charges sert de document de référence particulièrement lors de la phase de la réalisation afin de réduire au maximum les écarts entre les attentes du client et les fonctionnalités développées. On retrouve par exemple la charte graphique qui recense les différentes couleurs de l’application, la normalisation des boutons ainsi que les différentes polices utilisées.

*(Extrait de notre charte graphique)*

Définition de la police d’écriture

* Dans toute l’application
* Les titres H1, H2, H3, H4, H5, H6 : Cairo
* Le texte principal : Roboto / Helvetica

Définition de la taille de police

* Dans toute l’application
* Les titres H1 : 20px
* Le texte de base : 16px





Dans le cahier des charges, il est stipulé également le Déroulement du projet de manière à détailler la planification du travail.

*(Extrait de la partie déroulement de notre cahier des charges)*

Déroulement du projet

Planification

La réalisation du projet est constituée de plusieurs phases :

* Recherche des besoins du client
  + Entretiens écrits ou vocaux
* Conception
  + Réalisation du cahier des charges
  + Modélisation de la base de données
  + Conception des différentes maquettes
  + Réalisation des spécifications détaillées
  + Réalisation des analyses fonctionnelles
* Le développement
* Création du projet
* Mise en place du pattern MVC

Le cahier des charges sert de contrat et a donc une valeur juridique. A la fin de celui-ci, on indique les coordonnées des parties liées par le cahier des charges. Le Maitre d’ouvrage qui est la partie qui crée la demande. Et le maître d’œuvre qui est la partie chargée de réaliser le travail.

*(Extrait du cahier des charges)*

Responsabilités

Maitrise d’ouvrage

Le maître d'ouvrage aussi appelé MOA est monsieur Pascal CARLIER dirigeant de l’entreprise GNOSYS ayant le numéro de SIRET : 82455455400017.

Maitrise d’œuvre

Les maîtres d'œuvre aussi appelé MOE sont messieurs Quentin BALAIR et Bruno MAYEUX stagiaires AFPA en formation de Concepteur Développeur D’Applications.

Le cahier des charges a été réalisé à l’aide du logiciel Word.

## Le dictionnaire de données

Le dictionnaire de données est une première approche de la conception de la base de données.  
Elle permet de réunir toutes les entités et attributs dont on pense avoir besoin, son édition n’est pas décisive, elle permet simplement de mettre en place les données dont l’appli aura besoin avec une description sur les attributs qui ne sont pas explicites.

Il a été créé à l’aide d’Excel, inutile de le présenter tout le monde connaît.

|  |  |  |
| --- | --- | --- |
| Entité utilisateur | | |
| Attributs | types | description |
| id | int auto\_increment |  |
| adresse\_mail | varchar(100) |  |
| mot\_de\_passe | varchar(50) |  |
| nom | varchar(75) |  |
| prenom | varchar(50) |  |
| adresse | varchar(240) | adresse postale |
| ville | varchar(75) |  |
| code\_postal | varchar(10) |  |
| numerosTelephone | Text |  |
| ComplementsAdresse | Text |  |
|  |  |  |
|  |  |  |
| Entité entreprise | | |
| Attributs | types | description |
| id | int auto\_increment |  |
| raison\_sociale | varchar(75) |  |
| numero\_siren | varchar(20) |  |
| numero\_siret | varchar(20) |  |
| numero\_tva | varchar(20) |  |
| numero\_activite | varchar(100) | numéro d'activité de l'entreprise |
| code\_naf | varchar(20) |  |
| adresse | varchar(240) |  |
| site\_internet | varchar(50) |  |
| ville | varchar(75) |  |
| code\_postal | varchar(10) |  |
| nom\_comptable | varchar(50) |  |
| telephone\_comptable | varchar(10) |  |
| numerosTelephone | Text |  |
| ComplementsAdresse | Text |  |
| type\_entreprise | varchar(50) |  |
| assurance\_pro | varchar(50) |  |
| nom\_etablissement\_payeur | varchar(50) |  |
| adresse\_etablissement\_payeur | varchar(240) |  |

*(Extrait du dictionnaire de données, le reste est en annexes)*

## Le MCD

Le MCD vient de la méthode Merise, elle consiste à concevoir le modèle de la base de données et schématiser la circulation des données dans la base de données en liant les entités entres elles avec les relations.

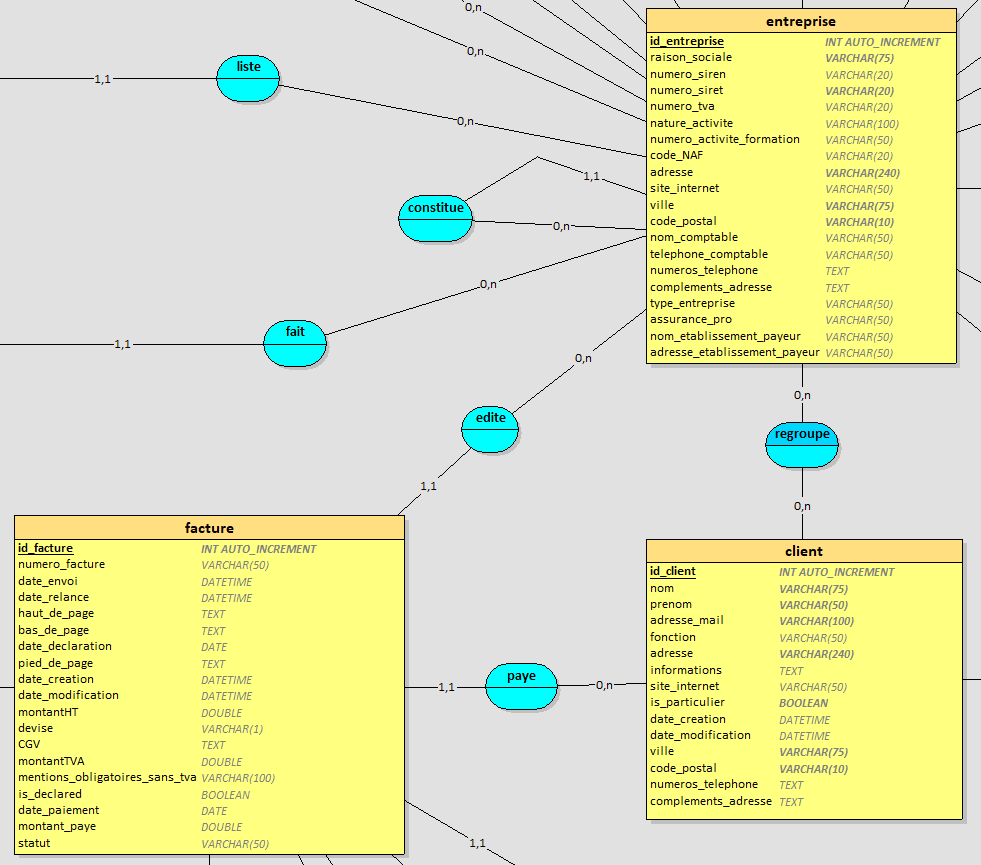
Dans un MCD, on y retrouve plusieurs termes techniques :

* Une entité : Une entité représente un objet qui regroupe des attributs dans un système d’informations (base de données), elle dispose d’une clé primaire qui lui sert d’identifiant unique pour chaque enregistrement de celle-ci.
* Une relation : Une relation c’est une association reliée à deux ou plusieurs entités par des cardinalités, elle peut elle aussi contenir des attributs, elle pourra être transformée en table selon les règles des cardinalités dans le MLD.
* Une cardinalité : Une cardinalité est une liaison entre une entité et une association, elle permet d’interpréter et de déterminer le nombre minimum et maximum de possibilités que chaque entité contient dans la relation.
* Une clé primaire : Une clé primaire peut être composée un ou plusieurs attributs de l’entité
* L’héritage : Une entité hérite des attributs de l’entité mère, et contenir ses propres attributs, l’héritage est utilisé dans le cas où deux entités ont des attributs en commun.
* Un attribut C’est un élément d'information de base attaché à une entité.
* Un type : C’est un type de données qui est attaché à un attribut afin de n’autoriser que celui-ci à être stocké.

**Looping:** C’est un logiciel permettant de créer le MCD et MLD ainsi que générer un script de base de données.

Une image contenant texte, clipart

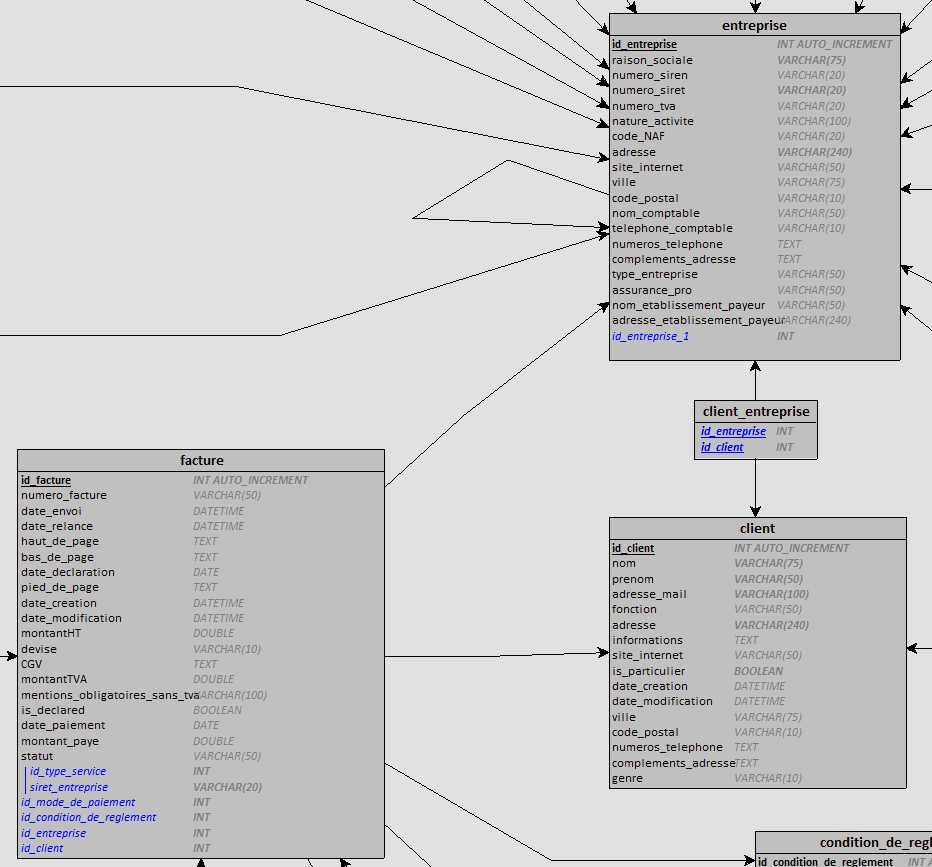
Description générée automatiquement



*(extrait de MCD, vous pouvez retrouver l’intégralité du mcd en annexe)*

## Le MLD

Le MLD (Modèle Logique de Données) est le résultat du MCD, il transforme toutes les entités et quelques relations en tables ou tables associative, les attributs en nom de colonnes et les clés étrangères apparaissent dans les tables respectives.



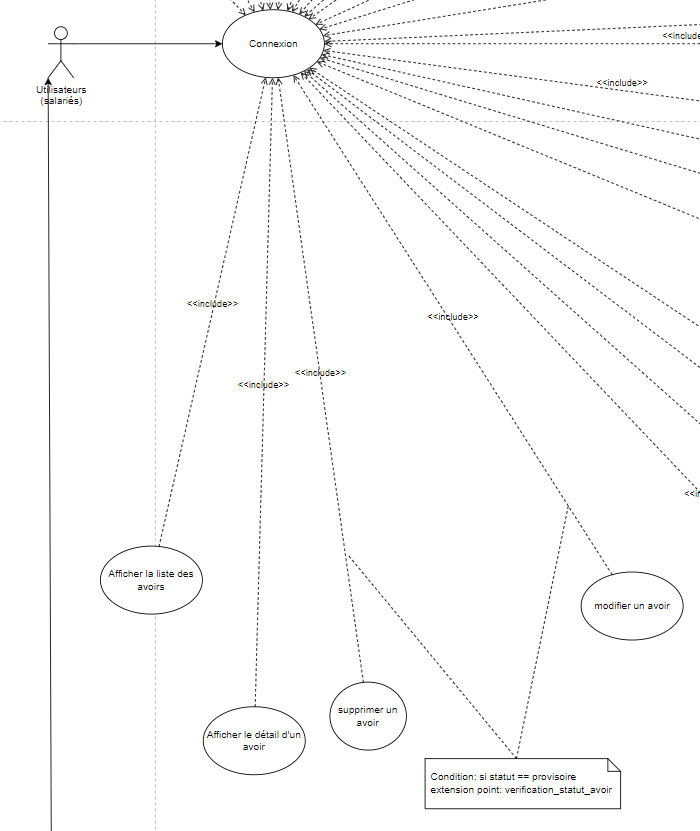
*(extrait de MLD, vous pouvez retrouver l’intégralité du mld en annexe)*

## Diagrammes UML

Le langage UML (Unified Modeling Language, ou langage de modélisation unifié) est un langage de modélisation visuelle.

### Le diagramme de cas d’utilisation

Le diagramme de Cas d’utilisation est un schéma simple. Il doit être compris de tous, même des personnes travaillant en dehors du monde de l’informatique. Il formalise graphiquement les besoins des utilisateurs. Pour cela, il montre les interactions entre les acteurs et le système. Son rôle est de recenser les fonctionnalités principales d’un système, et de montrer ce qu’un utilisateur peut réaliser au niveau de l’application.

***Extrait du diagramme de cas d’utilisation***

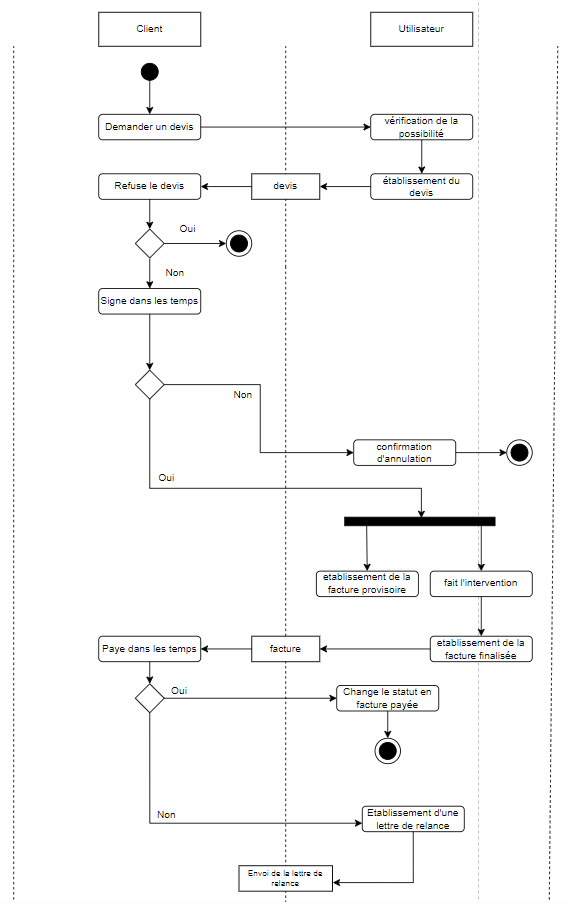
***L’intégralité du diagramme de cas d’utilisation se situe en annexe p.***

Plusieurs éléments composent ce diagramme :

* Personnage ( Salarié ) => Représente l’utilisateur.
* Les ellipse => Représente les actions réalisables.
* Les flèches en pointillé avec include => Indique que l’action précédente est obligatoire.
* Le rectangle => Représente une condition. Cette action sera réalisable que si la condition est vrai.

### Le diagramme de séquences

Le diagramme de séquence sert à décrire le fonctionnement interne du système. Il montre les enchaînements d’opérations qui se produisent lorsqu’un acteur déclenche une action. Il y a un ordre chronologique dans la lecture du schéma qui se fait du haut vers le bas

***Diagramme de séquences***

Plusieurs symboles composent ce diagramme :

* Le point noir => Représente le début de la séquence.
* Le pont noir encerclé => Représente la fin de la séquence.
* Les flèches => Représente les cheminements entre les actions.
* Les rectangles => représentent les différentes actions.
* Les losanges => représentent les conditions.
* La barre horizontale épaisse => Indique que les deux actions suivantes sont déclenchées en parallèle.



Pour réaliser les diagrammes UML, nous avons utilisé le logiciel draw.io.

Il s’agit d’une application qui permet de faire des schémas et du dessin vectoriel. Elle a la particularité d’être une application web. Cette application nous a également permis de réaliser les maquettes et les wireframes.

### Le diagramme de classes

Le diagramme de Classe représente la structure de l’application orientée objet en répertoriant toutes les classes de cette dernière. Une classe est symbolisée par un rectangle pouvant être découpé en trois compartiments. Le premier pour le nom de la classe, le second pour ses attributs et le dernier pour ses méthodes. Les classes sont donc les éléments principaux de ce schéma. Une classe est un ensemble d’attributs et de fonctions permettant de créer des objets à la chaine en leur donnant des caractéristiques et un comportement commun.

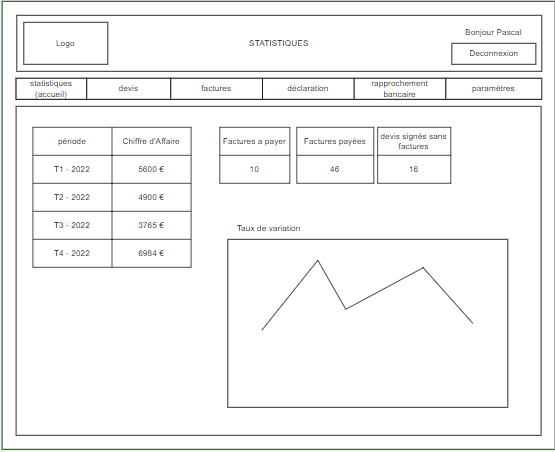
***Extrait du diagramme de classes :*** Une image contenant texte

Description générée automatiquement

## Wireframes

C'est la première ébauche des interfaces de l'application, qui peuvent être faites lors du premier rendez-vous ou après l'analyse de l'existant.

Afin de positionner les différents éléments de l'interface pour faire rendre compte au client, quel aspect aura l'appli. Il sert en priorité à disposer les éléments interactifs de l’application afin de prendre en compte l’expérience utilisateur et de favoriser celle-ci.

Voici quelques exemples :

Une image contenant table

Description générée automatiquement

*(Le reste des wireframes sont en annexe)*

## Maquettes en couleurs

Après avoir fait valider les wireframes par le client et établi la charte graphique, on peut très simplement créer les maquettes, dans le cas où les wireframes sont validés partiellement par le client, les maquettes permettent de rectifier les erreurs en même temps afin d'optimiser le temps.

Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement

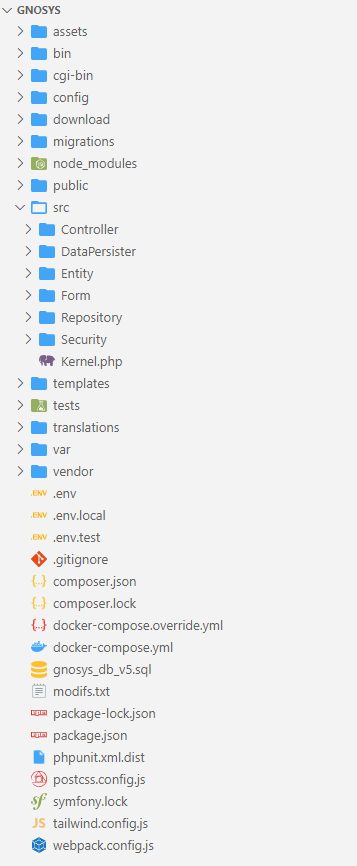
Une image contenant texte

Description générée automatiquement

*(Le reste des maquettes en couleurs sont en annexe)*

# Le développement

## Présentation de l’arborescence



Le dossier config contient les bundles, les routes sa sécurité et les services

Le dossier migrations contient les fichiers permettant à Symfony de construire et modifier la base de données.

Le dossier public contient les sources du projet : Code css, Js, images et vidéos.

Le dossier src contient les classes, controller, entités, formulaires.

Le dossier templates contient les vues de l’application.

Le dossier var contient le cache et les logs de l’application.

Le dossier vendor contient les bundles liés à l’application.

Le fichier .env contient les paramètres de connexion à la base de données et au serveur de mails.

## Modèle MVC

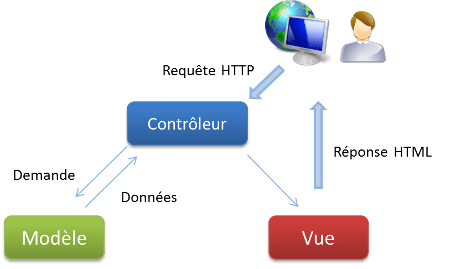
Pour la conception de l’application, j’ai utilisé une architecture MVC (Model – View -Controller) qui permet de bien organiser le code source. Le but du MVC est de séparer la logique du code en 3 parties :

**Model** : Cette partie gère les données du site. Son rôle est d’aller récupérer les informations dans la base de données, de les organiser et de les assembler pour qu’elles puissent ensuite être traitées par le controller. On y trouve donc entre autres les requêtes SQL.

**Vue** : Cette partie concentre le code qui gère l’affichage du site. On y retrouve le code écrit en twig.

**Controller :** Cette partie joue le rôle d’intermédiaire entre le Model et la vue, il demande les données au model, il les analyse et les retourne à la vue afin de réaliser l’affichage. C’est lui qui gère la partie login. Dans notre cas cette partie ne contient que du code écrit en PHP.

**Schéma représentant l’architecture MVC**



## Connexion

Avec Symfony vous pouvez gérer la connexion à votre site / application web assez facilement.

Vous pouvez exécuter des commandes avec composer pour qu’il génère les classes et templates automatiquement, c’est d’ailleurs recommandé de laisser Symfony gérer cette connexion, en intégrant la sécurité que celui-ci propose. Vous pouvez tout de même modifier certains fichiers pour effectuer certaines actions personnalisées.

Tout d’abord, quand un utilisateur souhaite accéder à l’application, il doit se connecter.

Pour se connecter, il sera redirigé sur cette méthode.

    /\*\*

     \* @Route("/", name="app\_login")

     \*/

    public function login(AuthenticationUtils $authenticationUtils): Response

    {

        if ($this->getUser()) {

            return $this->redirectToRoute('app\_home');

        }

        // get the login error if there is one

        $error = $authenticationUtils->getLastAuthenticationError();

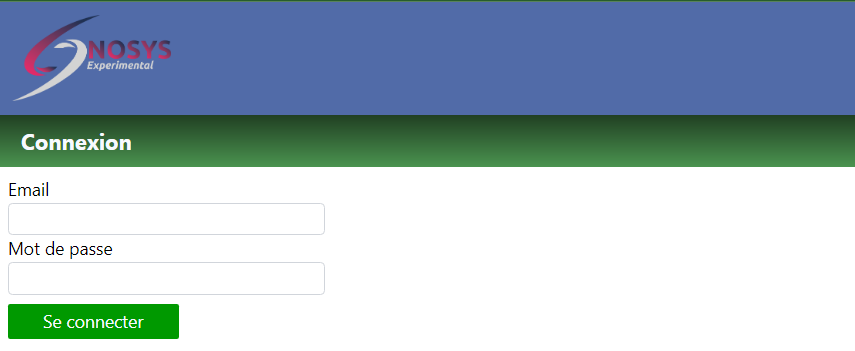
        // last username entered by the user

        $lastUsername = $authenticationUtils->getLastUsername();

return $this->render('security/login.html.twig', ['last\_username' => $lastUsername, 'error' => $error]);

    }

Cette méthode va dans un premier temps vérifier que l’utilisateur ne soit pas déjà connecté, si c’est le cas, il sera redirigé vers la page d’accueil.

Si l’utilisateur n’est pas connecté alors il sera redirigé vers la page de connexion ci-dessous.

Une fois que l’utilisateur a rempli le formulaire, la classe AppAuthenticator.php est la classe qui va gérer l’authentification d’un utilisateur.

public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response

    {

        if($token->getUser()->getFirstConnection())

        {

        return new RedirectResponse($this->urlGenerator->generate('app\_change\_password'));

        }

        // For example:

        return new RedirectResponse($this->urlGenerator->generate('app\_home'));

        // throw new \Exception('TODO: provide a valid redirect inside '.\_\_FILE\_\_);

    }

La méthode ci-dessus permet de rediriger un utilisateur, dans le cas présent si c’est la première connexion de l’utilisateur sur cette application, on le redirige directement une page pour changer son mot de passe parce que celui-ci était un mot de passe généré par l’application.

## Gestion des rôles

La gestion des rôles est indispensable pour mettre en place des applications web qui comprend des rôles pour les utilisateurs de celles-ci.

Quand on créer la table utilisateurs dans Symfony, la colonne « rôle » référence les rôles de l’utilisateur.

Dans le fichier security.yaml, on peut créer des rôles afin de pouvoir les utiliser par la suite.

role\_hierarchy:

ROLE\_USER: ROLE\_USER

ROLE\_CHEFENT: [ROLE\_USER]

ROLE\_ADMIN: ROLE\_ADMIN

Le ROLE\_USER, c’est le rôle par défaut assigner à tous les utilisateurs inscrits.

On peut faire hériter de ses droits à d’autres rôles, comme le rôle « ROLE\_CHEFENT », cet héritage permettra d’avoir les droits de l’utilisateur + ses propres droits à l’application.

## Routing

Lorsque l’application reçoit une requête, elle appelle une action du contrôleur pour générer la réponse. Les routes servent à définir l’action à exécuter pour chaque URL entrant. Dans notre application, les routes sont sous forme d’annotation au-dessus des fonctions.

Pour une meilleure organisation du routage, nous avons assigné une route à chaque contrôleur.

***Exemple de route afin d’ajouter un devis :***

Route au niveau du contrôleur.

/\*\*

\* @Route("/devis")

\*/

class DevisController extends AbstractController

Route au niveau de la méthode.

/\*\*

\* @Route("/add", name="app\_devis\_add")

\*/

public function adddevis(Request $request, EntityManagerInterface $entityManager,EntrepriseRepository $entrepriseRepository): Response

{

/ . . . . . /

}

Dans cet exemple la route « app\_devis\_add » est appelée lorsque que l’application reçoit :  /devis/add . Ce qui va entraîner l’exécution de la méthode add.

Si des paramètres doivent être passés à la fonction, cela doit être défini également au niveau de la route. Dans l’exemple situé ci-dessous, la route

/\*\*

\* @Route("/details/{id}", name="app\_devis\_details")

\*/

public function detaildevis(Devis $devis): Response

{

/ . . . . . /

}

Au niveau sécurité, il est possible d’autoriser ou de restreindre l’accès à une route à un utilisateur en fonction de son rôle. Pour cela, il suffit de définir les rôles pouvant accéder à cette route au niveau du fichier « Security.yaml »

***Voici un exemple tiré de l’application.***

access\_control:

        - { path: ^/home, roles: ROLE\_USER }

        - { path: ^/entreprise, roles: ROLE\_USER }

        - { path: ^/client, roles: ROLE\_USER }

        - { path: ^/devis, roles: ROLE\_USER }

        - { path: ^/facture, roles: ROLE\_USER }

        - { path: ^/admin, roles: ROLE\_ADMIN }

        - { path: ^/register, roles: [ROLE\_ADMIN, ROLE\_CHEFENT] }

        - { path: ^/declarations, roles: [ ROLE\_CHEFENT] }

        - { path: ^/admin, roles: ROLE\_ADMIN }

        - { path: ^/profil, roles: ROLE\_USER }

        - { path: ^/register, roles: [ROLE\_ADMIN, ROLE\_CHEFENT] }

Dans cet exemple, si nous prenons la première ligne du fichier, la route « /home » sera accessible aux utilisateurs ayant le « ROLE\_CHEFENT » car ce dernier hérite du rôle ‘ « ROLE\_USER ». La route « /register » quant à elle, ne sera accessible pour les rôles  « ROLE\_ADMIN et ROLE\_CHEFENT » .

## Filtrage

Pour pouvoir filtrer les devis / factures, on a fait appel a de l’AJAX (Asynchronous Javascript And XML) avec du javascript.

Pour ce faire, on récupère les boutons de filtres pour pouvoir poser des EventListeners, qui vont servir à écouter les évènements déclenchés par l’utilisateur.

const filtres = document.querySelectorAll("#filters a");

filtres.forEach(elm => {

    if (elm.hasAttribute("statut"))

    {

        elm.addEventListener("click", function (e) {

            getFiltre(e);

        });

    }

});

On pose les events listeners que sur les filtres qui ont les attributs « statut ». Pour éviter de poser des listeners sur d’autres éléments.

function getFiltre(e) {

    // on recupere l'url de la page

    const Url = new URL(window.location);

    let action = (e.target.getAttribute("statut")).toLowerCase();

    //on lance la requete AJAX

    let destination = Url.pathname + "?filter=" + action + "&ajax=1";

    fetch(destination, {

        headers: {

            "x-Requested-With": "XMLHttpRequest"

        }

    }).then(response => response.json())

        .then(data => {

            const content = document.getElementById("content");

            content.innerHTML = data.content;

            // on recharge le menu\_contextuel

            $.getScript('../../js/menu\_contextuelv2.js', function() {

                init();

            });

        })

        .catch(e => alert(e));

}

Cette fonction permet d’envoyer la requête au controller de la vue qui utilise le filtrage avec des paramètres en plus tels que le filtre (provisoire, signée, payée) et un paramètre ajax qui permet au controller de savoir si on vient d’un appel AJAX ou qu’on veut faire un affichage normal.

La méthode d’un controller ci-dessous

    /\*\*

     \* @Route("/", name="app\_facture")

     \*/

    public function index(FactureRepository $factureRepository, EntrepriseRepository $entrepriseRepository, Request $request): Response

    {

// some code ...

   if ($request->get("ajax")) {

            if ($request->get("filter") != "all") {

                $listeFacture = $factureRepository->findBy(["entrepriseUtilisateur" => $entrepriseUser, "statut" => $request->get("filter")]);

            }

            return new JsonResponse([

                "content" => $this->renderView('facture/\_content.html.twig', [

                    'listefacture' => $listeFacture,

                ]),

            ]);

        }

La première condition permet de savoir si on vient d’un appel en AJAX

Le deuxième permet de savoir si on veut afficher quelque chose en particulier.

Ensuite on retourn un objet JsonResponse qui va permettre d’être envoyé à la vue \_content.html.twig avec la liste des factures en paramètres et celle-ci pourra mettre en forme cette liste.

Ce qui permet donc de rafraîchir une partie de la page afin d’optimiser le temps de réponse.

## Collection type

Le collection type est un type d’input en Symfony qui permet de faire des formulaires imbriqués.

Dans notre exemple, l’application gère des devis et ces devis disposent de lignes.

Les lignes ne sont ni plus ni moins qu’un formulaire avec des données comme le type de service la quantité, le prix, la tva, etc …

Ces formulaires ne sont jamais remplis sans avoir rempli les données du devis, pour rendre la rédaction d’un devis plus facile, il faut donc intégrer le formulaire d’une ligne de devis dans le formulaire du devis.

En ajoutant ces lignes dans le FormType, le FormType c’est ce qui va permettre de construire le formulaire afin de pouvoir le mettre en forme dans la vue.

->add('lignesDevis', CollectionType::class, [

                'entry\_type' => LigneDevisFormType::class,

                'entry\_options' => ['label' => false],

                'prototype' => true,

                'allow\_add' => true,

                'allow\_delete' => true,

                'by\_reference' => false,

                'label' => 'Prestation',

                'mapped' => false,

                'data' => $options['ligneDevis'],

            ]);

Plusieurs attributs importants composent le collectionType :

* Entry type : Permet de sélectionner le formulaire associé aux lignes de devis
* Entry options : Permet d’ajouter des options aux formulaires de ligne de devis
* Prototype : permet de récupérer l’HTML du formulaire d’une ligne de devis afin de pouvoir la dupliquer par la suite par exemple
* Allow\_add et Allow\_delete : permet d’autoriser ou non la suppression/ajout de la liste de lignes de devis
* Data : permet de donner la liste des lignes de devis lié au devis quand on modifiera le devis par exemple.

Une fois le collection type construit, on peut aller l’afficher dans la vue correspondante.

## Envoi de mail

Pour envoyer des mails, on pourrait croire que c'est un calvaire, que c’est compliqué finalement avec symfony, c’est assez simple. Symfony dispose de symfony mailer, un bundle qui est conçu pour envoyer des mails simplement.

    /\*\*

     \* @Route("/sendMail/{id}", name="app\_facture\_sendmail")

     \*/

    function sendMail(Request $request,MailerInterface $mailer,KernelInterface $kernel,Facture $facture,EntrepriseRepository $entrepriseRepository)

    {

// some code ...

$email = (new Email())

               ->from('gnosysfacture@gmail.com')

               ->to($form->get('email')->getData())

               ->subject($form->get('objet')->getData())

               ->text($form->get('message')->getData())

               ->attachFromPath($kernel->getProjectDir().'/download/factures/'.$facture->getNumeroFacture().'.pdf');

            $mailer->send($email);

}

Cette méthode permet d’envoyer un mail.

Elle nécessite de créer un objet Email et celui-ci contient plein de paramètres qu’on peut lui assigner.

**From** : qui va permettre de donner l’adresse mail qui va apparaître au moment de la réception du mail.

**To**: L’email du destinataire qui va recevoir le mail

**Subject**: C’est simplement l’objet du mail

**Text**: C’est le corps du mail

**attachFromPath**: C’est la pièce jointe en l’occurrence ici, c’est le pdf qui a été sauvegardé

Ensuite, on demande au mailer qui est un objet MailerInterface d’envoyer le mail et Symfony s’occupe du reste !

Cette fonctionnalité de mail ne va pas faire en sorte de ne pas faire tomber le mail en spam.

## Templates twig

L’interface graphique de l’application est développée grâce au langage twig. Celui-ci va nous permettre de créer rapidement des templates.

Prenons l’exemple de la vue de base de l’application (base.html.twig). Ce fichier est le template de base qui est appelé a chaque appel à une page afin d’éviter de recopier tout le code de base dans chaque vue.

***Voici une vue du template  base .html.twig***:

Il appelle à son tour deux autres fichiers que l’on appelle les partials et qui sont :

header.html.twig qui contient le code de l’entête de l’application.

Navbarvertical.html.twig qui contient le code de la barre de navigation verticale.

L’autre fonctionnalité que j’ai utilisé dans l’application est l’utilisation de macro. Dans les pages de gestion de factures et de devis, il faut gérer un formulaire qui permet de créer des factures et devis, nous devons ajouter autant de lignes au document que l’utilisateur souhaite. Pour éviter réaliser la génération des lignes en Javascript, j’ai utilisé une macro dans twig . Cette partie de code renferme tout le template permettant la génération d’une ligne de formulaire ou de facture.

***Extrait de la macro utilisé pour la ligne de devis***

{# Prototype de la ligne de devis  #}

{% macro lignesDevis\_prototype(ligneDevis) %}

    <div class="lignecompta flex mt-2" data-order={{ligneDevis.children.numLigne.vars.data}}>

        <div class="invisible w-0">

            {{ form\_widget(ligneDevis.tauxTva)}}

            {{ form\_widget(ligneDevis.uniteService)}}

        </div>

        <div class="encadre w-11/12 p-2 grid grid-cols-8 gap-x-4 items-center text-center">

            {{ form\_label(ligneDevis.dateIntervention)}}

            {{ form\_label(ligneDevis.typeService)}}

            {{ form\_label(ligneDevis.quantite)}}

            {{ form\_label(ligneDevis.uniteService1)}}

            {{ form\_label(ligneDevis.prixHT)}}

            {{ form\_label(ligneDevis.tauxTva1)}}

            <label>Total HT</label>

Dans le formulaire d’ajout du devis, la première ligne du devis affichée par défaut se fait par la ligne suivante :

{{ form\_widget(devisForm.lignesDevis) }}

Quand l’utilisateur veux créer une nouvelle ligne en cliquant sur le lien « +Ajouter », celui-ci déclenche la fonction Javascript « addLigne » qui permet d’insérer une ligne au formulaire. Voici l’extrait du code qui permet l’insertion de la macro dans la DOM ( Document object Model).

let ligne = container.getAttribute("data-prototype").replace(/\_\_name\_\_/g, index);

let content = document.createElement("div");

content.innerHTML = ligne;

container.appendChild(content);

Lors de l’édition du devis ou de la facture, l’affichage des différentes lignes composant le document s’effectue grâce à une boucle for située dans le fichier twig. Si le document comporte des lignes à afficher, on boucle sur ces dernières et on insère autant de fois que nécessaire la macro dans la DOM.

Si le document ne comporte aucune ligne, dans ce cas on ne génère qu’une seule fois la macro pour n’afficher qu’une ligne vide.

***Voici l’extrait de code permettant de réaliser cette boucle.***

{% for ligne in devisForm.lignesDevis %}

    {{ \_self.lignesDevis\_prototype(ligne)}}

{% else %}

    {{ form\_widget(devisForm.lignesDevis) }}

{% endfor %}

***Une image contenant texte

Description générée automatiquementVue de la page d’ajout du devis***

## Mise en place d’un APIRest

Afin de pouvoir interagir avec la base de données depuis une application extérieure, j’ai mis en place une API (Application Programming Interface). Pour mettre en place cette API, j’ai utilisé l’outil API qui va intégrer les outils de gestion de l’API au projet Symfony. Pour mettre en place cet outil, la première étape pour la mise en place, est l’installation du bundle avec la commande suivante :

composer require api

Cette commande a pour action de créer l’interface « swagger » qui va recenser les tables accessibles par l’API et qui donnera la possibilité de tester les différentes actions possibles sur ces tables. Lors de sa création, cette interface est vide. Pour autoriser l’accès à une table, il faut placer l’annotation suivante au-dessus de l’entité.

/\*\*

\* @ORM\Entity(repositoryClass=ClientRepository::class)

\* @ApiResource

\*/

class Client

{

}

L’interface swagger sera la suivante :

Une image contenant table

Description générée automatiquement

Dans notre cas, comme nous avons des collections types, les objets sont imbriqués (par exemple, le client contient une entreprise et l’entreprise contient la liste de ses clients donc le client) Pour éviter de tomber dans une boucle infinie, il est nécessaire d’indiquer à l’API quels sont les attributs de l’entité à faire apparaître dans les requêtes. Pour ce faire, nous allons utiliser les Groupes. Pour les mettre en place, il faut ajouter des nouvelles annotations dans notre entité :

/\*\*

 \* @ORM\Entity(repositoryClass=ClientRepository::class)

 \* @ApiResource(

 \*     normalizationContext={"groups"={"read\_client"}},

 \*     denormalizationContext={"groups"={"write\_client"}})

 \*/

class Client

{

La ligne normalizationContext va indiquer que les requêtes « get » concerneront les attributs annotés par le groupe « read\_client »

La ligne denormalizationContext va indiquer que les requêtes « POST – PUT – PATCH -DELETE » concerneront les attributs annotés par le groupe « write\_client ».

Les attributs sur lesquels l’annotation « @groups » n’apparaît pas, ne seront pas pris en compte par l’API.

L’annotation de l’attribut se fait en ajoutant la ligne suivante :

/\*\*

\* @ORM\Column(type="string", length=75)

\* @Groups({"read\_client","write\_client"})

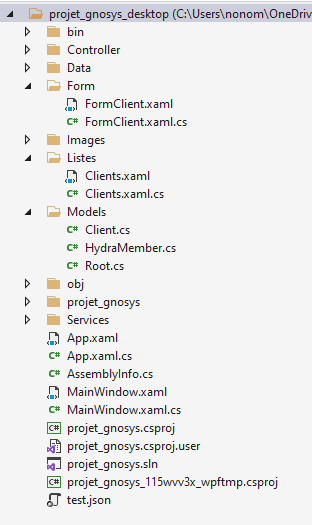
\*/

    private $nom;

## Application CSharp

Toujours dans le cadre de la certification, afin de réaliser l’application en multi-couches, nous avons développé une application en C# qui a pour but de communiquer avec l’API afin de réaliser un CRUD sur la table Client de notre base de données.

***Arborescence du projet :***



Les fichiers MainWindow contient le code de gestion de la fenêtre principale.

Le dossier Services contient le fichier de services de l’entité Client.

Le dossier Models contient les fichiers des différentes classes du projet.

Le dossier Listes contient les fichiers composant la vue affichant la liste des clients

Le dossier Form contient les fichiers composant la vue du formulaire client

Le dossier Data contient les DTO et les profiles de l’entité client

Le dossier Controller contient le Controller de l’entité Client.

Aperçu des différentes vues de l’application

***Page d’accueil de l’application :***



***Page affichant la liste des clients :***

Une image contenant table

Description générée automatiquement

***Page de formulaire :***



Ce formulaire est un formulaire unique qui permet l’ajout la suppression et la modification du client. C’est le clic sur le bouton d’action de la fenêtre « liste de clients » qui permet l’initialisation des paramètres de cette fenêtre tels que le titre, le texte du bouton d’action (Ajouter – Modifier – Supprimer) et mise en lecture seule des inputs lors de la suppression d’un client.

***La partie front :***

La partie front de l’application (interface WPF) est codée en XAML. La base de la page est une Grille (grid ) sur laquelle viennent se greffer les différents éléments qui composent cette page. Les deux fichiers qui composent une vue sont le fichier « \*.xaml » qui contient le code XAML proprement dit et le fichier \*.xaml.cs qui contient le code CSharp qui permet d’interagir avec cette page.

Voici un exemple tiré du code XAML de la fenêtre « liste des clients » :

<DataGrid x:Name="dg"

Grid.Column="1"

CanUserAddRows="False"

AutoGenerateColumns="False"

CellStyle="{DynamicResource SelectedBackground}">

<DataGrid.Columns>

<DataGridTextColumn Header="Id"

Binding="{Binding Id}"

Width="1.5\*"

Visibility="Hidden"/>

<DataGridTextColumn Header="Genre"

Binding="{Binding Genre}"

Width="0.7\*"/>

<DataGridTextColumn Header="Nom"

Binding="{Binding Nom}"

Width="1\*"/>

***Partie Back :***

La partie Back est écrite en C#. J’ai commencé par crée les Models qui sont les classes permettant de créer le différents objets clients. À la suite j’ai créé le DTO correspondant à ce Model. Les DTO permettent de formater les objets. Dans notre cas, nous avons un DTOIn qui va retourner l’objet avec son Id et le DTOOut qui retournera l’objet sans son Id. Afin de pouvoir lier les différents formats, j’ai créé le profil, qui permet de réaliser le mapping entre les différents formats.

Une fois ces actions réalisées, j’ai créé le contrôleur ainsi que le service. C’est dans le fichier service que nous allons trouver les méthodes qui communiquent avec l’API.

On commence tout d’abord par définir la constante contenant l’URL de l’API :

Cette déclaration dans une constante évite de réécrire l’URL en dur dans chaque fonction.

private const string URL = "https://gnosys.quentinbalair.fr/api/clients";

Puis on y écrit les méthodes qui vont réaliser les différentes requêtes pour Ajouter Modifier et Supprimer.

public async Task AddClientAsync(ClientsDTOOut obj)

{

if (obj == null)

{

throw new ArgumentNullException(nameof(obj));

}

HttpResponseMessage response = await client.PostAsJsonAsync(URL, obj);

}

Cette fonction récupère un objet de type ClientDTOOut qui est le client sans son id. C’est la fonction PostAsJsonAsync qui gère la conversion de l’objet en format JSON et envoie cet objet en méthode POST a l’adresse spécifiée dans la constante URL. Dans le cas où l’objet serait nul, une exception serait levée. Les méthodes pour réaliser la mise à jour et la suppression ont la même structure. La seule différence est la fonction qui envoie la requête.

Dans la méthode update on trouvera :

await client.DeleteAsync(URL+"/"+id);

Dans la méthode delete on trouvera :

await client.PutAsJsonAsync(URL +"/"+ id, obj);

Pour réaliser la récupération des données, j’ai écrit la fonction « GetAllClientAsync » .

public async Task<List<Client>> GetAllClientAsync()

{

var response = await client.GetAsync(URL);

var json="";

client.DefaultRequestHeaders.Accept.Add(new

MediaTypeWithQualityHeaderValue("application/ld+json"));

response.EnsureSuccessStatusCode();

if (response.IsSuccessStatusCode)

{

json = response.Content.ReadAsStringAsync().Result;

}

Root reponse = JsonConvert.DeserializeObject<Root>(json);

List<Client> ListeClients = new();

for (int i = 0; i < reponse.HydraMember.Count; i++)

{

ListeClients.Add(reponse.HydraMember[i].CreateClient());

}

return ListeClients;

}

Cette fonction effectue une requête Get à l’API. Cette dernière lui retourne la liste des clients au format JSON, voici le format de la réponse retourné par l’API.

{

    "@context":"/api/contexts/Client",

    "@id":"/api/clients",

    "@type":"hydra:Collection",

    "hydra:member":

    [

        {

            "@id":"/api/clients/1",

            "@type":"Client",

            "nom":"MAJAX",

            "prenom":"Gérard",

            "email":"gerard.majax@gmail.com",

            "adresse":"40 rue du blockus",

            "ville":"Arras",

            "codePostal":"62000",

            "numerosTelephone":"0321945212",

            "genre":"Monsieur"

        },

        {

            // deuxième objet...

        }

    ]

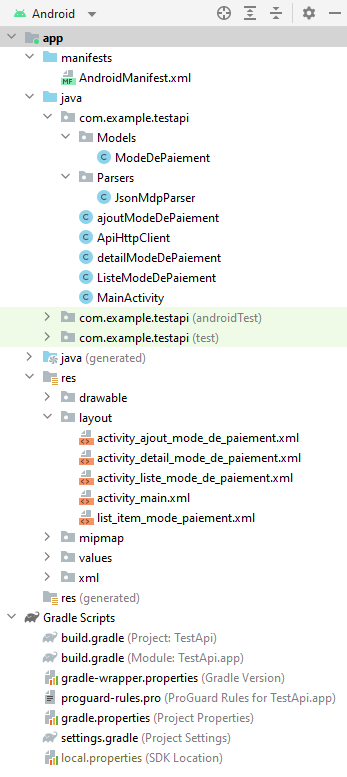
}

Pour pouvoir récupérer les différents clients, la fonction convertit cette réponse en objet Root dans lequel nous pourrons récupérer la liste « hydra member » La boucle sur cette liste, va nous permettre de créer la liste d’objets clients. Une fois que cette liste sera complète, elle sera envoyée directement à la DataGrid afin de réaliser l’affichage.

## Application Android

Dans le cadre de la certification pour correspondre le plus possible aux critères d’évaluations, nous avons développé une application Android qui communique avec l’API web, afin d’interagir avec la base de données.

***L’arborescence du projet***



Le dossiers gradle scripts, c’est un dossier qui sert a configurer notre projet en y important des extensions, en définissant la version d’android minimale requise pour lancer l’application, etc …

Les ressources vont regrouper toutes les ressources de l’application comme les images, les variables contenant des mots/phrases qui peuvent être traduites en plusieurs langues, les vues des activités qu’on crée dans l’application.

Les Activités ce sont les classes qui vont appeler la vue (layout) et vont faire le traitement pour celle-ci

Les parsers, ce sont les classes qui vont transformer le JSON et en créer des objets java

Les models, ce sont les classes qui vont créer des objets java

Le manifest, c’est le fichier de configuration de l’application, on y met le logo de l’application, quelle fenêtre de lancement on va afficher etc …

Pour commencer à développer, une activité de base nommée MainActivity est créé.  
Le fichier activity\_main.xml représente la partie front de l’activité.

Il suffit dans ce fichier, de créer des contrôles.

<Button  
 android:id="@+id/buttonMDP"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="192dp"  
 android:text="@String/btn\_mdp\_principal"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />

Dans le cas présent, c’est un bouton, on lui définit un id pour pouvoir le récupérer dans la partie back afin de pouvoir lui mettre un event listener par exemple.

Pour les autres attributs, ils sont assez explicites, ils servent à donner un style au bouton, mais surtout d’ajouter des contraintes, ces contraintes permettent de définir la position du bouton par rapport aux autres contrôles.

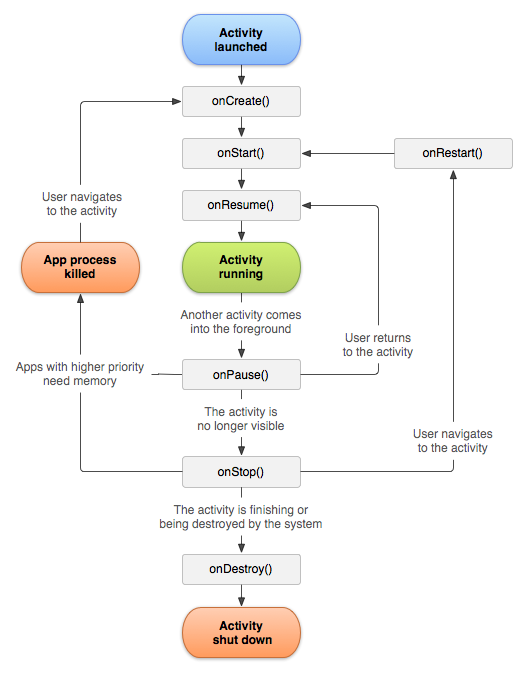
L’attribut text fait appel à une ressource de type string qui permettra de mettre le contenu de celui-ci dans l’attribut text.

<string name="btn\_mdp\_principal">Modes de Paiement</string>

Une fois que la construction de la vue est finie, il faut gérer les interactions entre l’utilisateur et celle-ci.

Dans la classe Java liée à cette vue (layout) on y trouve une méthode « onCreate », cette méthode va être hérité de la classe mère et correspond à un état dans le cycle de vie d’une application.

Le cycle de vie d’une application correspond aux différents états de celle-ci lorsque d’autres applications prennent le dessus par exemple lorsque l’on reçoit un appel, l’application reste en arrière-plan.



Voici un schéma expliquant le cycle de vie d’une application.

On peut voir différents états de l’application qui peuvent être gérés en appelant des méthodes de la classe mère afin de gérer toute exception en cas de déclenchement d’évènements tiers.

En général les deux premières lignes de cette méthode, on ne les change pas, c’est pour appeler la vue qui est liée à cette classe pour former l’activité.

public class MainActivity extends AppCompatActivity {  
  
  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 *// au lancement de l'application* super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);

*// on declare une nouvelle intention et on récupère le bouton*  
 Intent intent = new Intent(this, ListeModeDePaiement.class);  
 Button boutonModePaiement= findViewById(R.id.*buttonMDP*);  
 boutonModePaiement.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 startActivity(intent);  
 }  
 });  
  
 }  
}

On déclare une nouvelle intention, c’est pour préparer la navigation vers l’autre activité.

On passe en paramètre la classe d’où l’intention part et la classe ou l’intention va être traitée.

Ensuite, on récupère le bouton via son id, et on y ajoute un Click listener (un « écouteur d’évènement de type « click » ) et dans le méthode onClick on y ajoute la fonction startActivity qui va permettre de lancer la nouvelle activité.

Et enfin pour faire un appel à l’API en java, il va nous falloir quelques connaissances.

Au moment d’arriver dans l’activité qui va accueillir la liste des modes de paiement, on va lancer la méthode sendRequest.

*/\*\*  
 \* Lancement de la recherche  
 \*/*public void sendRequest(View view) {  
 InputMethodManager imm = (InputMethodManager) getSystemService(Context.*INPUT\_METHOD\_SERVICE*);  
 if (isNetworkAvailable()) {  
   
 if (isOnline()) {  
 AppelAPI task = new AppelAPI();  
 task.execute();  
 imm.hideSoftInputFromWindow(view.getWindowToken(), 0);  
 } else {  
 Toast.*makeText*(view.getContext(), getApplicationContext().getResources().getString(R.string.*no\_online*), Toast.*LENGTH\_LONG*);  
 Log.*d*("DEBUG", "sendRequest: pas d'internet");  
 }  
 } else {  
 Toast.*makeText*(view.getContext(), getApplicationContext().getResources().getString(R.string.*no\_network*), Toast.*LENGTH\_LONG*);  
 Log.*d*("DEBUG", "sendRequest: pas de reseau");  
  
 }  
}

Pour faire simple cette méthode, va vérifier que l’appareil est bien connecté à internet pour pouvoir faire l’envoi d’une requête à l’API, si c’est le cas, elle va demander à la classe AppelApi, d’exécuter ses tâches, c’est donc elle qui va demander à envoyer une requête à l’API et elle va recevoir l’information.

AppelApi va donc appeler la méthode ci-dessous pour afficher la liste, ce n’est qu’un extrait des bouts de code les plus intéressants.

Dans un premier temps dans cette méthode, on déclare les variables dont on va avoir besoin.

Ensuite, on essaie d’ouvrir une connexion http de type GET, avec l’url donnée et on reçoit une réponse, si elle est comprise entre 200 et 299 c’est une réponse avec le statut OK, et cette réponse est donc retournée à la méthode on PostExecute de la classe AppelAPI qui elle va créer une liste d’objet ModeDePaiement afin de l’integrer dans le contrôle listView.

public String getListeMDP() {  
 HttpURLConnection con = null;  
 InputStream is = null;  
 String url = "";  
 int responseCode;

url = *BASE\_URL* + "mode\_de\_paiements/";

try {  
 *// on établit la connexion* con = (HttpURLConnection) (new URL(url)).openConnection();  
 con.setRequestMethod("GET");  
 con.setDoInput(true);  
 con.setDoOutput(false);  
 con.connect();

*// en fonction du code réponse, on récupère les données ou l'erreur* responseCode = con.getResponseCode();  
 if (200 <= responseCode && responseCode <= 299) {  
 is = con.getInputStream();  
 } else {  
 is = con.getErrorStream();  
 }

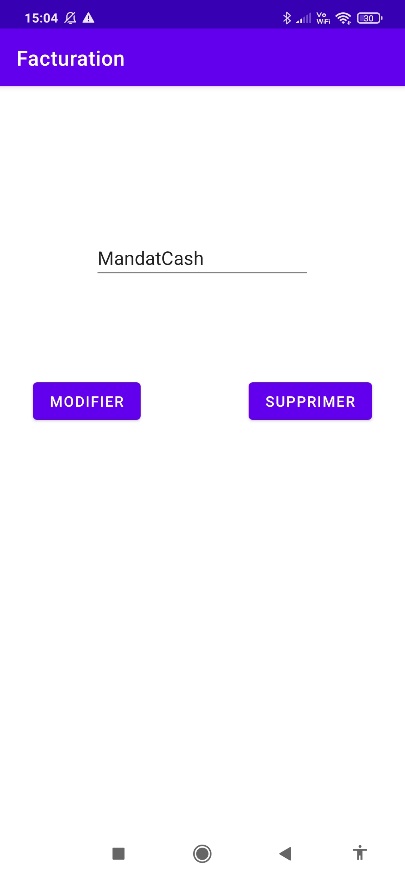
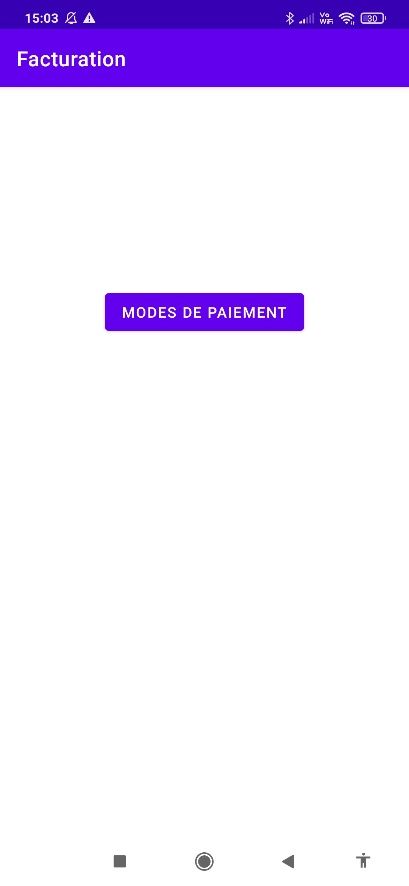
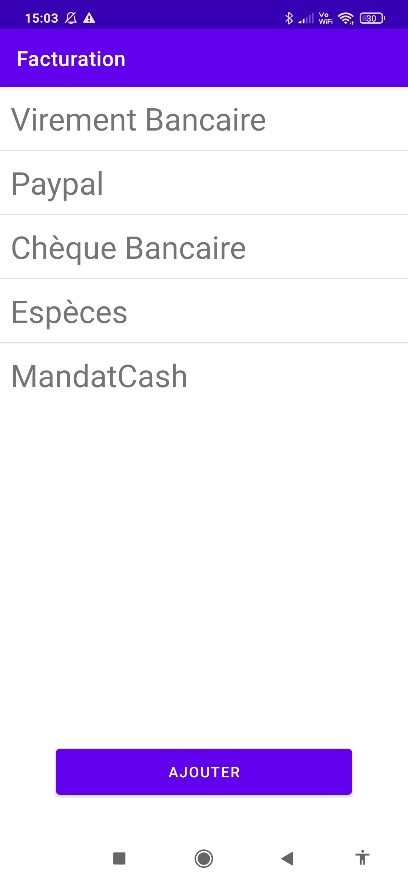
*// on ferme la connexion* is.close();  
 con.disconnect();  
  
 *//on récupère le résultat en chaîne de caractères (ici json)* return buffer.toString();

} catch (Throwable t) {  
 t.printStackTrace();

}

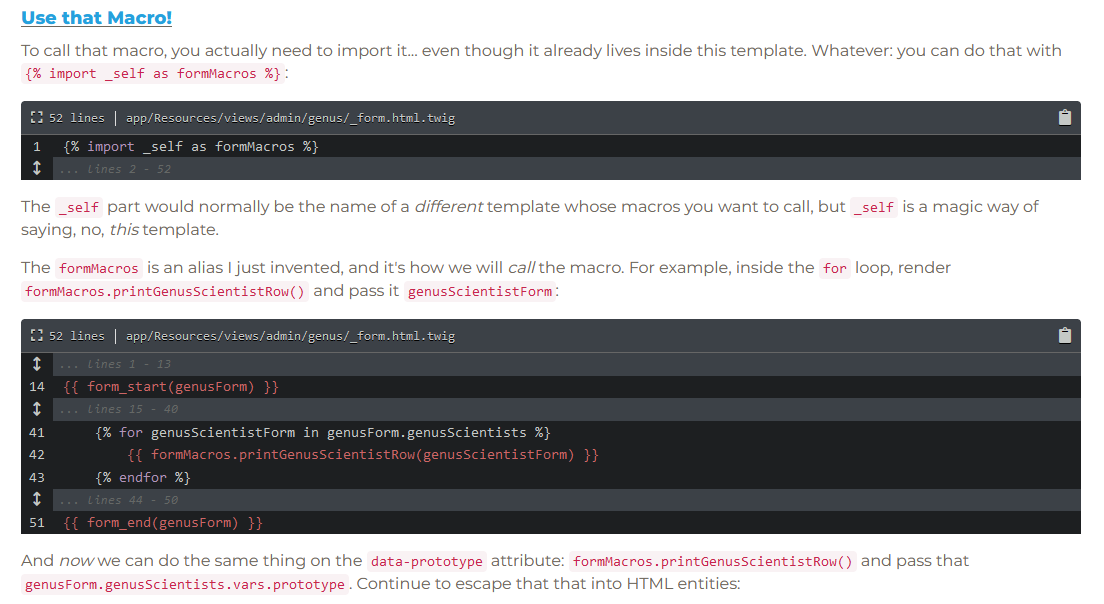
return null;  
}

Et nous avons une application android qui fonctionne, vous pouvez retrouver des captures d’écrans ci-dessous.



# Problème rencontré

Une image contenant texte

Description générée automatiquementLors du développement, j’étais confronté à un problème, je ne savais pas comment mettre les formulaires des lignes de factures dans le formulaire, de facture j’ai donc dû chercher sur les différents.

Dans le cas présent, la personne explique comment faire en sorte d’avoir un formulaire imbriqué stylisé et souhaite donc aussi faire en sorte de pouvoir dupliquer ce formulaire afin que tous les formulaires imbriqués aient le même style pour quelque chose d’homogène.

Ensuite la personne dit que dans la boucle « for » il va appeler le templates (la macro) et il va lui passer l’élément du tableau en paramètres.

# La veille technologique

Les attaques par déni de services sont des attaques qui datent depuis bien longtemps, mais reste toujours d’actualité.

Le principe de cette attaque est d’envoyer un maximum de requêtes au serveur afin de le rendre inaccessible par les utilisateurs, ce qui peut donc entrainer un arrêt temporaire de l’activité.

Prenons l’exemple de Amazon Web Service (AWS), cette entreprise a subi une attaque DDoS en février 2020. Étant une grosse entreprise, elle est forcément vulnérable, cependant elle y était préparée. Habituée à recevoir des attaques à hauteur de 1 terabit/s, celle-ci s’est fait surprendre avec un trafic à hauteur de 2,4 Tb/s. Ce qu’il faut retenir, c’est que nous ne sommes jamais à l’abri d’une faille, même si celle-ci est connue depuis 1999, elle ne reste pas moins dangereuse.

# La conclusion

## Stage

Ce stage m’a permis d’approfondir les connaissances que j’ai pu acquérir durant la formation notamment au niveau de la conception avoir une expérience professionnelle ne peut qu’être bénéfique, elle permet de prendre conscience de l’importance de la conception et du développement d'un projet tout en faisant attention à la sécurité et aux données que l'on peut recueillir.

## Formation

J’ai trouvé la formation très qualifiante, la formatrice Martine POIX est très professionnelle, elle a su transmettre tout son savoir afin de faire de nous de bons concepteurs développeurs d’applications, finalement ça a été intense pour tout le monde. Mais tout le monde était là pour les autres. Je pense que l’apprentissage se fait aussi grâce aux collègues, ils ont su remonter le moral, demander de l’aide mais aussi m’aider car sans eux je pense que ça aurait été très différent. Il était très important de conserver sa motivation pour supporter la difficulté, le stress, la fatigue, la colère et toute autres émotions !

# Les remerciements

Mes premiers remerciements sont attribués à … MARTINE !

Martine POIX, la formatrice par excellence dans le domaine, elle qui a toujours été à l’écoute, qui nous a transmis tout son savoir-faire et qui nous a aidé jusqu’à la fin de la formation, elle a aussi été présente dans les moments difficiles comme dans les moments de joie, elle a su nous rebooster un par un durant toute la formation et aussi mettre un coup de pied pour nous remettre dans le droit chemin !

Un remerciement à tous mes camarades de formation pour l’aide et la confiance qu’ils m’ont accordés, et plus particulièrement à Bruno MAYEUX qui a su m’apporter du soutien et une bonne ambiance durant le stage.

Je remercie Monsieur CARLIER dirigeant de l’entreprise Gnosys Experimental de m’avoir donné l’opportunité d’effectuer mon stage au sein de son entreprise.

Je terminerais en remerciant la région Hauts-de-France pour le financement de ma Formation.

# Les annexes