# RAPPORT DE STAGE


**PIETKA CEDRIC**

**DWWM 2020/2021**

# PRESENTATION DU PROJET

O’SPEED PIZZA est une pizzéria de vente à emporter et de livraison basée à Esquelbecq. Il s’agit d’une micro-entreprise créée par Fabien Fiers en 2018. Ne disposant que d’un site web sans possibilité de commande en ligne, le projet consiste à créer une application PWA qui donnera la possibilité au client de commander directement en ligne, sur n’importe quel support. La PWA, c’est-à-dire Progressive Web Application, offre entre autres la faculté de placer l’application directement sur son écran d’accueil de téléphone, sans avoir à l’installer comme une application native.

Cette application permettra donc au client, après création d’un compte, de choisir sa pizza, en modifier la composition selon ses goûts, ainsi que des boissons et/ou desserts. La création de compte sera obligatoire, le client pouvant ainsi obtenir des points de fidélités à chaque commande. Une fois son choix fait, il pourra choisir un horaire de livraison, sa commande lui étant confirmée par SMS une fois validée par l’administrateur.

Côté administrateur, l’application proposera au gestionnaire de créer ses recettes de façon simple. La description des pizzas s’accompagnera d’une photo de celle-ci.

Elle permettra également de gérer le stock des boissons et des desserts, une notification alertant l’administrateur dès que le nombre de produits d’un certain type atteindra un seuil prédéfini.

Lorsqu’il recevra une commande, il validera d’un clique et l’application enverra automatiquement un message de confirmation au client. Elle imprimera aussi une étiquette contenant l’adresse de livraison, simplifiant ainsi l’organisation.

De plus, l’historique des commandes donnera la possibilité de connaître les différentes données, comme par exemple les pizzas ayant le plus de succès.

# COMPETENCES EVALUEES

| | | |
|---|---|---|
| Front-end | **Maquetter une application** | ✔ |
| | **Réaliser une interface utilisateur web statique et adaptable** | ✔ |
| | **Développer une interface utilisateur dynamique** | ✔ |
| | **Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce** | ✔ |
| Back-end | **Créer une base de données** | ✔ |
| | **Développer les composants d’accès aux données** | ✔ |
| | **Développer la partie back-end d’une application web ou web mobile** | ✔ |
| | **Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce** | ✔ |

# Table des matières

competences evaluees ..... 2

Presentation du projet ..... **Error! Bookmark not defined.**

I. Technologies utilisées ..... 7

II. maquettes et Diagrammes ..... 9

1. Maquettes ..... 9

2. Diagrammes ..... 10

III. La base de données ..... 13

3. Le dictionnaire de données ..... 13

4. le modele conceptuel de données ou mcd ..... 13

5. le modele logique de donnees ou mld ..... 14

6. la base de donnees ..... 15

IV. Structure Et technique ..... 18

7. la partie Front ..... 18

8. La gestion des images ..... 21

9. recupération des informations de la checkbox ..... 23

# I. CAHIER DES CHARGES

## Coté administrateur

L'application permet à l'administrateur d'ajouter, de modifier ou de supprimer une pizza.

Pour ajouter une pizza, il doit :

- Lui donner un nom
- Sélectionner les ingrédients
- Ajouter une photo
- Indiquer le prix
- Valider la recette

Lui donner un nom et indiquer son prix se fait de façon traditionnelle, comme dans un formulaire.

Les ingrédients à ajouter se présentent sous forme d'icônes sur lesquelles il faut cliquer.

Ils sont regroupés par type : les bases(tomate ou crème), les viandes, les légumes, les fromages et les poissons.

Il suffit de cliquer dessus pour les sélectionner et de cliquer de nouveau pour les désélectionner.

L'application permet ensuite de rajouter une photo. Elle ouvre l'appareil photo du téléphone et reliera directement la photo à la recette.

Une fois la recette composée, il suffit de valider. On est alors dirigé vers une page de récapitulatif, qui une fois validée enregistre la recette.

Pour modifier une recette, le principe est le même. L'administrateur arrive sur une page qui présente la pizza avec les différents ingrédients présentés sous forme d'icônes. En cliquent sur les images, on sélectionne les ingrédients à enlever. Pour en ajouter on agit comme dans l'ajout pizza. Une fois les ingrédients enlevés ou ajoutés, on peut modifier la photo et le prix.

Une fois la modification effectuée, on valide en cliquent sur le bouton modifier. On est ainsi

redirigé sur la page de récapitulatif ou il suffit de valider pour enregistrer la pizza.

Pour supprimer, il suffit de sélectionner une pizza et de confirmer la suppression.

Une fois la commande effectué par le client, l'administrateur reçoit une notification. Lorsqu'il valide la réception de la commande, un SMS est envoyé automatiquement sur le téléphone du client.

L'application imprime également une étiquette contenant le numéro de commande, l'adresse de livraison et le nom du client.

## Côté client

Lorsque le client arrive sur l'application, il doit, si c'est la première fois, se créer un compte. Pour cela il doit remplir un formulaire avec son nom, prénom, adresse postale et adresse mail et mot de passe.

Une fois cela, il se connecte à l'aide de son adresse mail et de son mot de passe. Il arrive sur la page d'accueil et se dirige vers la page de sélection de pizza, de boisson ou de dessert selon l'onglet sur lequel il clique.

La page sélection pizza présente les différentes pizza disponibles. Elle se compose du nom de la pizza, de sa photo et de sa description. On passe d'une pizza à l'autre grâce au swap vers la droite ou la gauche. Lorsque le client sélectionne une pizza, il se retrouve sur la page de description pizza. C'est sur cette page qu'il peut modifier la recette selon ses goûts. En cliquent sur un ingrédient, il le sélectionne pour l'enlever de la recette. Il peut en ajouter de la même manière, en cliquant sur l'ingrédient se trouvant sur la liste des suppléments.

Si il ajoute un ingrédient pour en remplacer un, le prix ne change pas. Si il en rajoute un sans en enlever, le prix augmente selon le tarif du supplément.

Lorsqu'il valide, la pizza est envoyée dans le panier. Il peut continuer sa commande en choisissant une autre pizza ou en se rendant dans l'onglet de choix de boissons ou de desserts.

Ces pages se composent d’une liste des boissons/desserts. Il suffit de cliquer sur l’icône correspondant à son choix pour sélectionner une ou plusieurs boissons/desserts. De la même façon que pour les pizzas, en validant ils sont envoyés dans le panier.

Quand la commande est terminée, le client se retrouve sur la page panier qui récapitule l’ensemble de la commande, ainsi que le prix. Il peut choisir le créneau horaire souhaité pour la livraison. Une fois validée, la commande est envoyée à l’administrateur. Lorsque celui-ci la valide à son tour, le client reçoit un SMS de confirmation.

# III. TECHNOLOGIES UTILISEES

Pour mener à bien le projet, j'ai utilisées les différentes technologies suivantes.


Tout d'abord le PHP. Acronyme pour *Hypertext Preprocessor*, il s'agit d'un langage spécialement conçu pour le développement web. Utilisé côté serveur, il permet de générer du HTML(entre autres) qui sera envoyé au client et ainsi produire des pages web dynamiques.

Wamp est un acronyme pour *Windows Apache MySQL PHP*. Il s'agit d'une plateforme de développement web utilisant le serveur Apache, et permettant de faire fonctionner les script PHP. On l'utilise surtout pour développer son application, ou l'améliorer, tout se faisant en local.


MySql est un système de gestion de base de données open source. Il stocke toutes les données de l'application. Il supporte le langage SQL, qui signifie *Structured Query Language* c'est-à-dire *Langage de Requête Structurée*. Le langage SQL permet entre autre la manipulation des données ; autrement dit on peut ajouter, modifier ou supprimer des données.


Le HTML, soit *HyperTextMarkup Language* (langage de balise pour l’hypertexte en français) est le langage utilisé pour représenter une page web. Il ne s’agit pas d’un langage à proprement parler mais plutôt d’un système de balises servant à structurer la page.

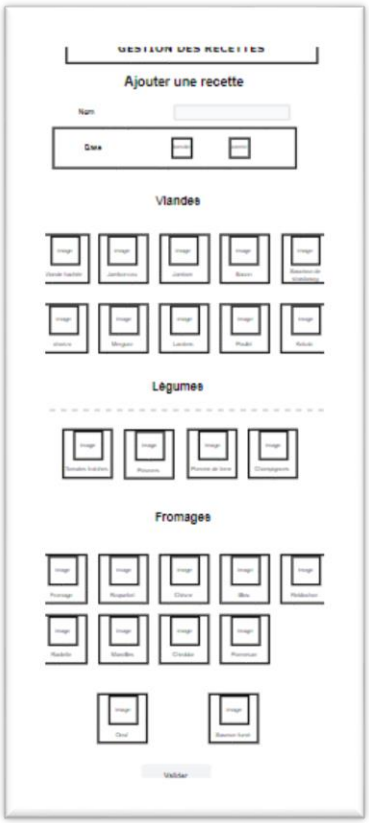
Le CSS, *Cascading Style Sheets*(feuilles de style en cascade) sert à ajouter du style à la page web(couleur, format de police, Il est standardisé par le W3C.

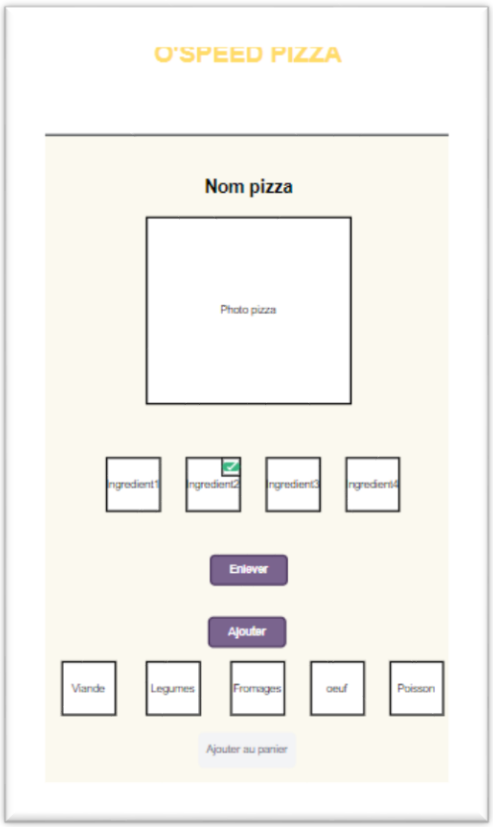

Le JavaScript est un langage principalement utilisé pour rendre une application dynamique. Il est très peu typé, permet d’agir directement sur le DOM(*Document Object Model*), c’est-à-dire la structure même de la page web.

# IV. MAQUETTES ET DIAGRAMMES

## 1. MAQUETTES

Après une première réunion avec Fabien, pour définir ses attentes, j’ai commencé par produire des maquettes. Sa principale exigence étant la simplicité d’utilisation, nous avons opté pour une interface qui serait presque totalement graphique, à base d’icônes cliquables.




La première maquette ci-dessus représente la page d’ajout de recette, donc côté administrateur. La page est structurée par type de produits qui composeront la recette. Les produits sont représentés par des icônes sur lesquelles il suffit de cliquer pour les ajouter. Le but étant qu’il y ait le moins de texte possible à écrire.

Du côté client, la maquette représente la page de description de la pizza sélectionée. Ici encore, elle est composée d’icônes cliquables permettant d’enlever ou de rajouter des ingrédients selon ses goûts.

## 2. DIAGRAMMES

L'étape suivante fut de conceptualiser l'application grâce à la méthode UML(*Unified Modeling Language*). Tout d'abord, le diagramme des cas d'utilisation permet de visualiser les interactions des différents acteurs avec l'application. Dy a ans mon cas, il n'a que deux acteurs, l'administrateur et le client.

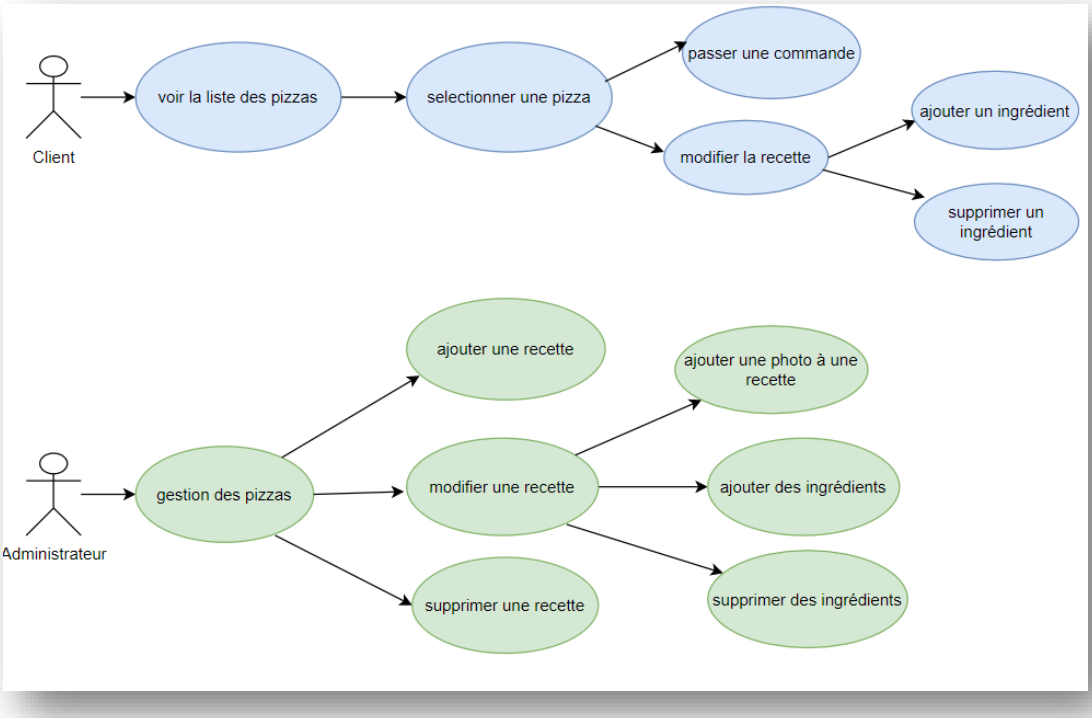



*Figure 1-Diagramme cas d'utilisation*

Une fois visualisé les fonctions permises aux différents acteurs, je mets au point le diagramme de classes. Le diagramme de classes permet de décrire le comportement d'un ensemble d'objets. On peut ainsi découper une tâche complexes en plusieurs tâches simples. Cette modélisation nous permet d'organiser le développement à venir. Les différentes classes se retrouveront en tables de la bases de données. Ici, comme on peut le voir sur la figure ci-dessous, j'ai établi  dix classes, qui donneront donc dix tables dans la base de données.

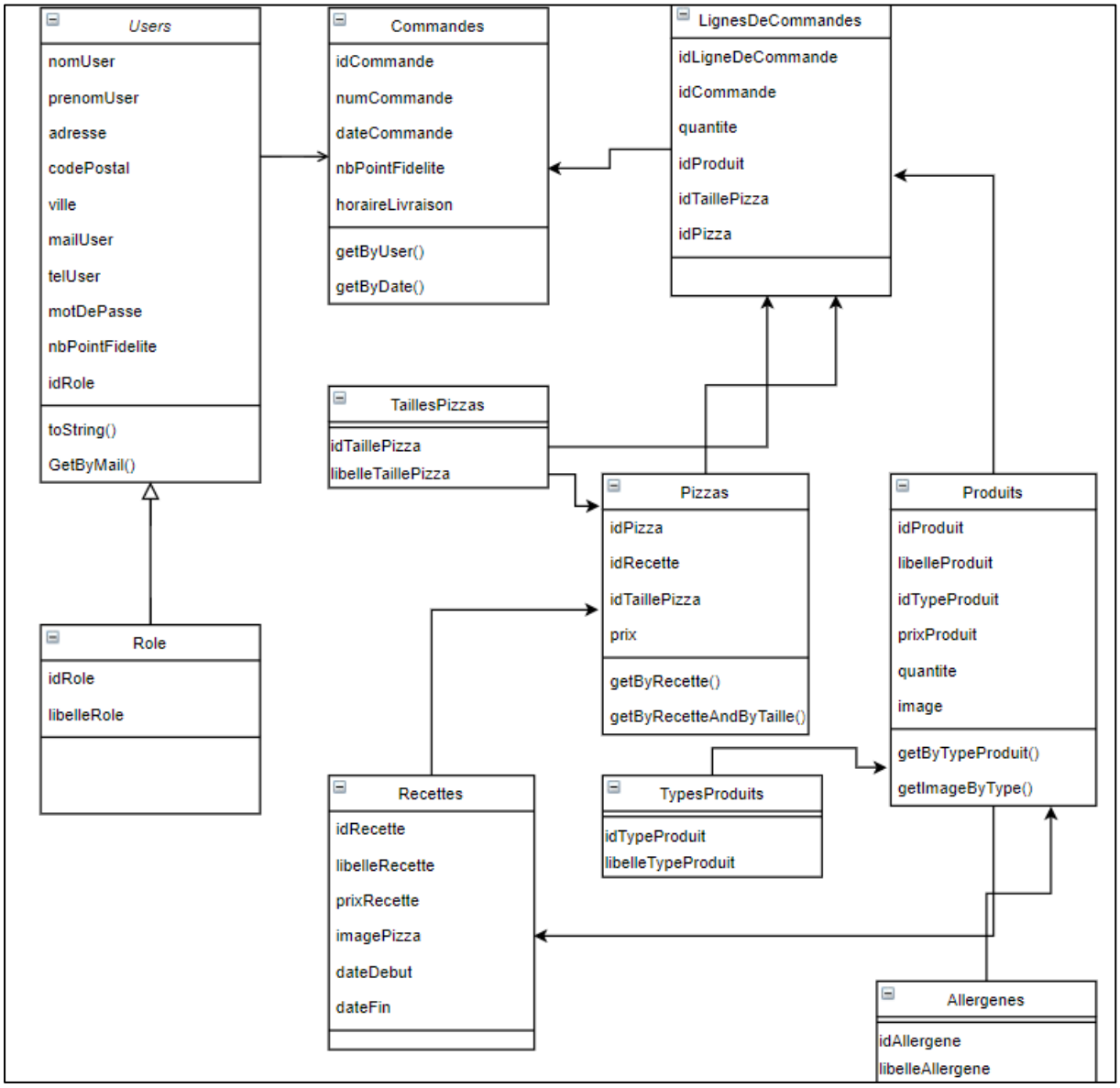



*Figure 2-diagramme de classes*

Enfin, j’ai fini par établir le diagramme de flux, qui représente le fonctionnement de l’application

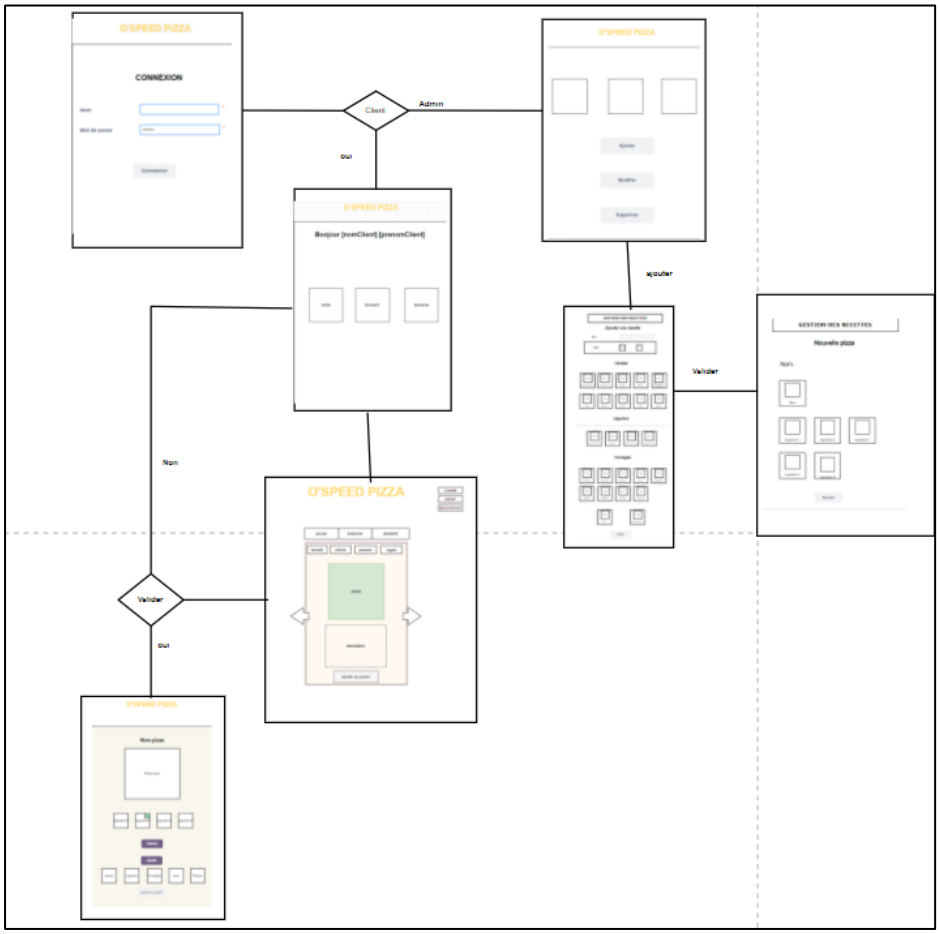



*Figure 3 diagramme de flux*

Ce diagramme nous montre comment va réagir l’application pour chaque action de l’utilisateur. Ainsi, on peut voir par exemple que lors de la connexion, si le rôle de l’utilisateur est client, il va être redirigé sur la page de menu, alors que s’il est administrateur, il va se retrouver sur la page de gestion de l’application. On a donc ici une idée visuelle du fonctionnement de l’application.

Une fois les diagrammes terminés, je me suis consacré à la création de la base de données.

# V. LA BASE DE DONNEES

## 3. LE DICTIONNAIRE DE DONNEES

Pour mettre en place la base données, il faut commencer par créer le dictionnaire de données. Il s'agit du référentiel de l'application, qui décrit toutes les données importantes. Il définit le vocabulaire de l'application . C'est sur lui que l'on se base pour créer toute l'application.

## 4. LE MODELE CONCEPTUEL DE DONNEES OU MCD

Après avoir déterminé les données qui composeront la base, j'ai mis en place le MCD. Il s'agit du schéma **entités/relations**. Il permet de visualiser graphiquement comment les données regroupées en entités sont reliées entre elles par des relations. Pour mon projet, j'ai élaboré mon application autour de sa fonction principale qui est la prise et la gestion de la commande. J'ai donc créée une entité LignesDeCommandes, qui sera en quelque sorte le «panier » du client, et établi ensuite les entités qui devraient être en relation avec elle. Ainsi, elle devra contenir les différents produits qui y seront ajoutés, donc les pizzas, mais aussi les boissons et les desserts. J'ai donc créé les entités Pizzas et Produits, qui regroupera tous les produits apparaissant dans l'application. Pour différencier les produits vendus séparément(boissons et desserts) et les ingrédients nécessaire à la fabrication des pizzas, il fallait une entité que j'ai appelée TypesProduits, et qui trie de façon exhaustive tous les types de produits présents. Ensuite, comme les prix des pizzas sont fixés en fonction de leur taille, j'ai tout d'abord déterminé une entité Recettes, qui contient le nom de la pizza, les ingrédients utilisés et le prix de base, puis TaillesPizzas, qui définit juste la taille de la pizza et le prix des suppléments. Les deux sont reliées à l'entité Pizzas pour définir le prix de la pizza commandée. Pour finir, j'ai établi l'entité Commande, qui récapitulera la commande du client et définira la date et l'heure de livraison ; puis les entités Allergenes, Users et Roles.

Pour chaque relation, il faut ensuite définir les cardinalités. C'est-à dire préciser combien de fois au minimum et au maximum une occurrence d'entité peut être liée à une autre occurrence d'entité.

Une fois le MCD terminé, il ne reste plus qu'à générer le MLD.

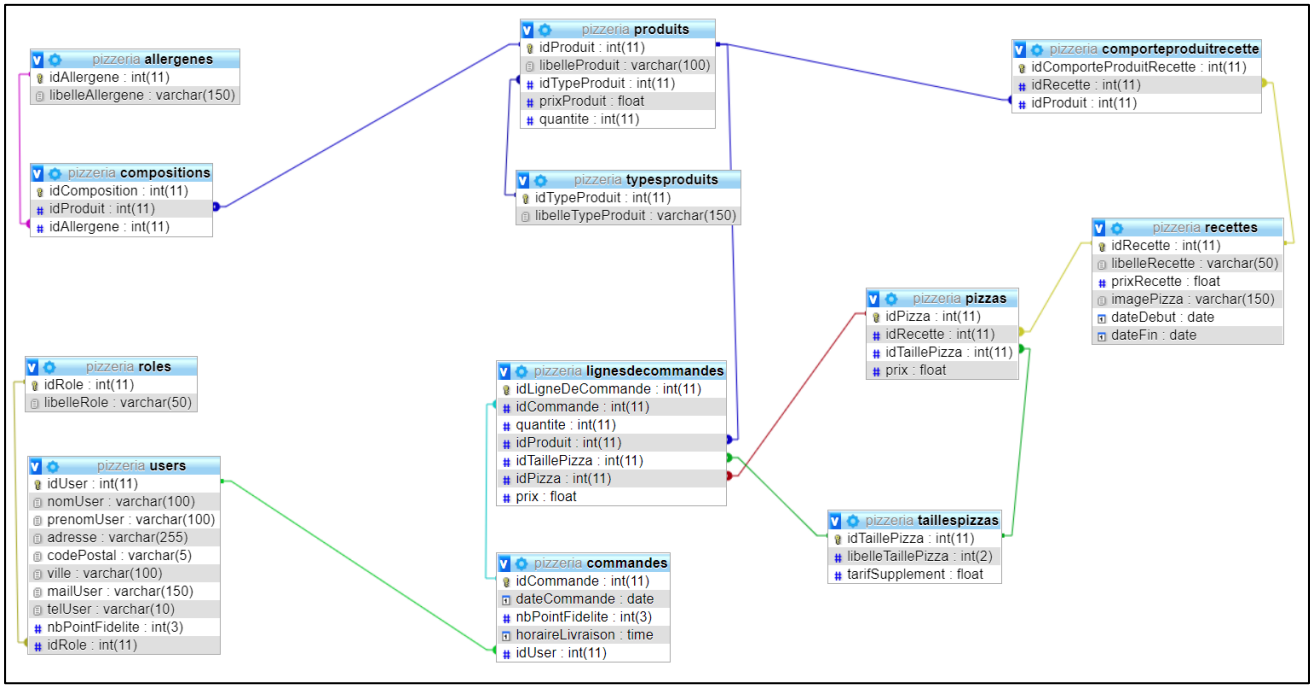
## 5. LE MODELE LOGIQUE DE DONNEES OU MLD

Pour créer la base de données, il faut prendre en compte la nature de l'outil logicielle avec lequel elle sera implantée. Pour cela, on utilise le Modèle Logique de données. Il en existe plusieurs sortes. Ici j'utilise le modèle relationnel.

Dans ce modèle, on traduit les entités du MCD par des **tables** du même nom. Les colonnes des tables correspondent au propriétés des entités. L'identifiant de chaque table est appelée **clé primaire**. Les associations avec les cardinalités (0,1)-n (0,1)-n et qui ne contiennent pas de données sont transformées en tables, qu'on appelle **tables associatives** et dont le nom est la concaténation des noms des deux tables reliées. Ces tables prennent deux **clés étrangères**, qui sont les clés primaires des deux tables reliées par l'association. Quand elles portent des données, elles prennent une clé primaire qui est la concaténation des deux clés primaires des tables reliées.

Ainsi, j'ai deux tables associatives : ComporteProduitRecette et Compositions, qui sont les associations entre respectivement les tables Produits et Recettes, et les tables Produits et Allergenes.

Le MLD étant généré, j'avais tout ce qui était nécessaire pour créer la base de données.



## 6. LA BASE DE DONNEES

Pour développer, j'utilise WampServer. Je code le script à l'aide du MLD puis je l'importe dans phpMyAdmin. Je créai donc ici la base de données que j'ai nommée pizzeria, contenant les tables du MLD. J'ai de plus créé trois vues. Les vues sont des tables virtuelles qu'on crée à partir de plusieurs tables, dont on prend les colonnes dont on a besoin pour facilités la récupération d'information.

J'ai ainsi créé la vue ProduitsRecettes, jonction des tables Produits, Recettes et ComporteProduitRecette, qui comporte les colonnes idProduit, libelleProduit, idRecette, libelleRecette, prixRecette. Ensuite la vue AllergenesProduits, qui contient les colonnes idProduit, libelleProduit, idComposition, idAllergene et libelleAllergene. Elle est la jonction entre les tables Produits, Allergenes et Compositions. Et enfin la vue CommandePizza, jonction des tables Commandes et LignesDeCommandes, qui comporte les colonnes idCommande, numCommande, dateCommande, idUser, idLigneDeCommande, quantite et idPizza.

Voici le script SQL pour les tables Produits et Recettes :

```
DROP DATABASE IF EXISTS pizzeria;


CREATE DATABASE IF NOT EXISTS pizzeria DEFAULT CHARACTER SET utf8 COLLATE utf8_gen
eral_ci;
USE pizzeria;


DROP TABLE IF EXISTS produits;
DROP TABLE IF EXISTS recettes;


CREATE TABLE produits(
    idProduit               INT  AUTO_INCREMENT  NOT NULL  PRIMARY KEY,
    libelleProduit          VARCHAR(100)  NOT NULL,
```

```
    idTypeProduit           INT  NOT NULL,
    prixProduit             FLOAT    ,
    quantite                INT,
    image                   VARCHAR(255)
)ENGINE=InnoDB, CHARSET = UTF8;

CREATE TABLE recettes(
    idRecette               INT  AUTO_INCREMENT  NOT NULL  PRIMARY KEY,
    libelleRecette          VARCHAR(50)  NOT NULL,
    prixRecette             FLOAT  NOT NULL,
    imagePizza              VARCHAR(150),
    dateDebut               DATE,
    dateFin                 DATE
)ENGINE=InnoDB, CHARSET = UTF8;
```

Et voici la vue ProduitsRecettes créée à partir de ces deux tables :

```
CREATE VIEW produitsrecettes AS
SELECT
p.idProduit,
p.libelleProduit,
cpr.idComporteProduitRecette,
r.idRecette,
r.libelleRecette,
r.imagePizza
FROM
produits AS p
INNER JOIN comporteProduitRecette AS cpr
ON
    p.idProduit = cpr.idProduit
INNER JOIN recettes AS  r
ON
    cpr.idRecette = r.idRecette;
```

Pour commencer le script, j’utilise l’instruction **DROP DATABASE IF EXISTS** pour effacer la base de données pizzeria si elle existe déjà. Alors seulement je crée la BDD, en précisant **IF NOT EXISTS**. L’instruction **USE** permet de préciser quelle base de données va être utilisée pour la suite.

Pour créer la vue, je sélectionne les colonnes que j’ai besoin pour chaque table, puis l’instruction INNER JOIN va permettre d’effectuer la jointure entre les tables.

# VI. STRUCTURE ET TECHNIQUE

Pour la structure de l'application, j'ai utilisé le framework construit pendant la formation. Mais j'ai quelque peut réorganisé les dossiers, pour plus de clarté. Ainsi, dans la partie PHP, j'ai divisé le dossier CONTROLLER en deux partie, une qui contient les fichiers ACTIONS, et la deuxième les CLASSES. J'y ai aussi mis le fichier ***Outils***.

Dans le dossier MODEL se trouvent les API et les MANAGERS. Et enfin dans le VIEW j'ai mis les FORM(les formulaires), les LISTES ainsi que les fichiers de front(dans le dossier GENERAL).

## 7. LA PARTIE FRONT

Pour la partie front, il fallait que l'application soit ***responsive***, c'est-à-dire qu'elle s'adapte à tous les écrans. Pour cela j'ai inséré la ligne suivante :

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

La balise ***viewport*** permet au navigateur de gérer les différents types d'écrans. Ici, on indique que la largeur du contenu doit être égale à la largeur de l'écran.

La plus grande difficulté fut d'afficher la checkbox, qui contient les différents ingrésients de la pizza, en fonction de leur type. En effet, quand le client sélectionne une pizza, il est redirigé vers une page qui affiche ladite pizza avec son nom, les types d'ingrédients, et sous chaque type les ingrédients de ce type.

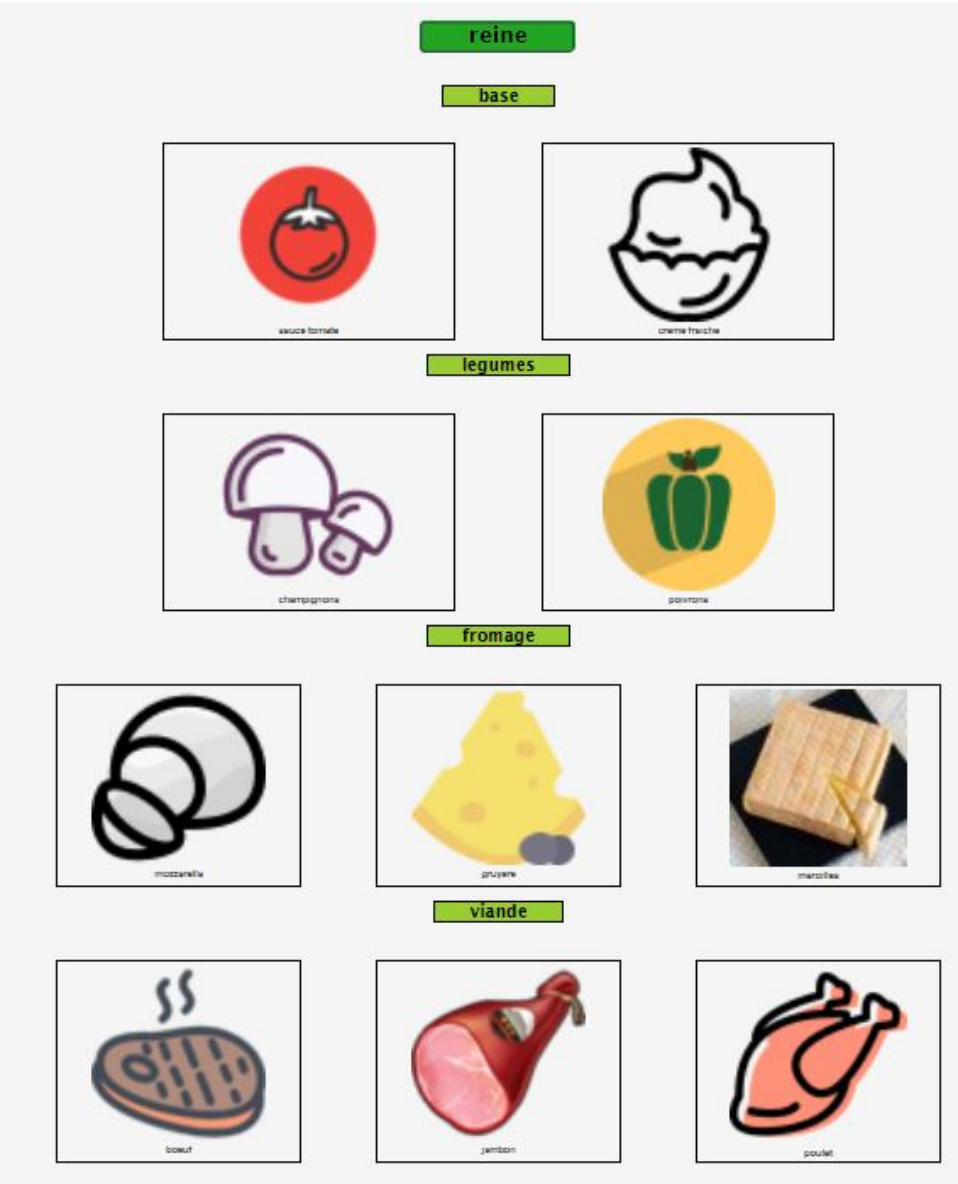
Dans un premier temps, j'avais créé une fonction *afficherCheckbox()* qui prenait en paramètres une liste d'id, une table, et le nom de la classe. Cette fonction retourne un code HTML qui affiche la checkbox.

Pour alterner les noms de type et les checkbox, j'ai retravailler la fonction en ajoutant des paramètres optionnels. La fonction peut donc maintenant prendre une seconde liste d'id, une seconde table, et une seconde classe. En utilisant deux boucles *foreach* imbriquées, j'obtiens bien l'affichage que je recherchais.

```
function afficherCheckBox($listeId, $table, $input, $secondListeId=null, $secondTabl
e=null, $secondInput=null, $mode=null){
    $nomManager= $table.'Manager';
    $disabled= '';
    $nom= substr($table, 0, -1);
    $nomId= 'getLibelle'.$nom;
    $idType= 'getIdType'.$nom;

    $secondNomManager= $secondTable.'Manager';
    $secondNom= preg_replace('/s+/','', $secondTable);
    $secondNomId= 'getLibelleType'.$nom;

    $checked='';
    foreach($secondListeId as $secondId)
    {
        $obj= $secondNomManager::findById($secondId);
        $secondLibelle= $obj->$secondNomId();
        if($obj->$secondNomId()!='boisson' && $obj->$secondNomId()!='dessert')
        {
            echo'
            <div class="double"><h2 class="libelle">'.$secondLibelle.'</h2></div><di
v class="espaceHor"></div><div class="checkbox">
            <div class="double"></div>
            <div class="mini"></div>';
            foreach($listeId as $id)
            {
                $elt = $nomManager::findById($id);
                $image= $elt->getImage();
                $idT= $elt->$idType();

                if($idT == $secondId)
                {
                    echo'
                    <div class="check centre">
                        <input type="checkbox" '.$disabled.' id="'.$elt-
>$nomId().'" name="'.$input.'" class="classcheck" '.$checked.'><label for="'.$elt-
>$nomId().'"><img src="IMG/'.$image.'" alt="'.$elt->$nomId().' class="idImage">
                        <input type="text" name="'.$elt->$nomId().'" value="'.$elt-
>$nomId().'" class="inputCheck" disabled><input name="'.$input.'[]" value="'.$id.'"
type="hidden"></label>

                    </div>
                    <div class="mini"></div>';
                }
            }echo'<div class="double"></div></div>';
        }
    }
}
```

Dans cette fonction, on commence par mettre dans des variables le nom du Manager nécessaire pour récupérer le nom du type de produit, puis les méthode « getLibelle » et « getId » dans deux autres variables. On fait la même chose pour la deuxième table prise en paramètre. Ensuite, pour chaque élément de la deuxième liste d’id , on instancie un objet en le recherchant par son id, puis on récupère son libellé et on l’affiche.

Toujours dans cette même boucle, on récupère, pour la première liste d’id prise en paramètre, on instancie un objet  pour récupérer l’image et l’id. On affiche enfin le tout.



## 8. LA GESTION DES IMAGES

Le but de l'application était de pouvoir faire le maximum avec des icônes. En effet, le patron étant peu à l'aise avec tout ce qui touche à la technologie et voulant quelque chose qui soit le plus simple d'utilisation possible, avec un minimum de texte et qui soit très rapide, il fallait faire avec le moins d'input texte possible.

La plus grande difficulté dans cette partie était de pouvoir rentrer une image en base de données. Pour cela, j'ai créé une fonction *chargerImage()* qui prend en paramètre le *name* de l'input de l'image. Elle modifie le nom de l'image pour éviter les doublons, et le renvoie.

```php
function chargerImage($fileChamp){
    if(is_uploaded_file($_FILES[$fileChamp]['tmp_name']))
    {
        $leNom = uniqid('jpg_') . '.jpg';

        move_uploaded_file($_FILES[$fileChamp]['tmp_name'], 'IMG/' .$leNom);
    }

    return $leNom;
}
```

En JavaScript, j'ai créé trois fonctions pour gérer l'affichage de l'image. La première gère la modification de l'image.

```javascript
function clickModifImage(e) {
    document.getElementById("image").hidden= true;
    document.querySelector("button[type=button]").hidden= true;
    url = window.location.search;
    if(url.indexOf("Produit")>0)
    {
        input = document.getElementsByName("image")[0];
    }
    else
    {
        input = document.getElementsByName("imagePizza")[0];
    }
    input.hidden = false;
    input.type = "file";
    modif = document.createElement("input");
    modif.name= "modifImage";
    modif.hidden = true;
    input.addEventListener("change", choixImage);
    input.parentNode.appendChild(modif);
}
```

On récupère l'id de l'input selon l'url de la page. On crée un élément input de *name modifImage*, et on ajoute l'input avec le l'image de la fonction *choixImage()*.

La fonction *choixImage()* permet de créer la balise *img* pour afficher l'image chargée.

```
function choixImage(e) {
    image = e.target.value;
    if(image.lenght>0)
    {
        imageType = /^image\//;
        fichier = e.target.files[0];
        if(!imageType.test(fichier.type))
        {
            alert("Sélectionnez une image");
            e.target.value;
        }
        else
        {
            img = document.createElement("img");
            img.id="image";
            console.log(img.id);
            img.file = fichier;
            e.target.parentNode.appendChild(img);
            reader = new FileReader();
            reader.addEventListener("load", chargerImage);
            reader.readAsDataURL(img, file);
        }

    }
}
```

Ici, on stocke la valeur de la cible dans la variable *image*. On crée ensuite l'élément *img*, on lui donne l'id *« image »*, puis on place l'élément au bout des enfants de la cible. On utilise ensuite l'interface FileReader, qui sert à les sources de données dur l'appareil de l'utilisateur. Suite à l'appel d'événement, on charge l'image grâce à la fonction *chargerImage()*.

On a enfin la fonction *chargerImage()* qui retourne le contenu du fichier.

```
function chargerImage() {
    document.getElementById("image").src = e.target.result;
}
```

Pour gérer l'affichage, on récupère d'abord l'url de la page concernée. On recherche si le mot Produit ou le mot Recette est contenue dans l'url. Si c'est le cas, on recherche ensuite si elle contient le mot ajout ou modif. On peut ensuite soit ajouter soit modifier l'image grâce à un écouteur d'événement.

```
url =window.location.search;
if(url.indexOf("Recette")>0)
{
    recette = document.getElementsByName("idRecette")[0];
    if(url.indexOf("ajout")>0)
    {
        document.querySelector("input[type=file]").addEventListener("change", choixI
mage);
    }
    else if(url.indexOf("modif")>0)
    {
        document.querySelector("button[type=button]").addEventListener("click", clic
kModifImage);
    }
}

url = window.location.search;
if(url.indexOf("Produit")>0)
{
    produit = document.getElementsByName("idProduit")[0];
    if(url.indexOf("ajout")>0)
    {
        document.querySelector("input[type=file]").addEventListener("change",choixIm
age);
    }
    else if(url.indexOf("modif")>0)
    {
        document.querySelector("button[type=button]").addEventListener("click", clic
kModifImage);
    }
}
```

## 9. RECUPERATION DES INFORMATIONS DE LA CHECKBOX

Pour récupérer les informations des checkbox, j'ai rencontré quelques difficultés. La plus importante étant que mon POST ne récupérait que le derniers des inputs. Même après des recherches sur différents forums et l'essai de différentes méthodes, je ne réussissais pas à récupérer tous les produits checkés.

Finalement, sur le forum **stackoverflow**, j'ai trouvé une solution qui semble fonctionner. Il m'a suffit de rajouter des [] au *name* de l'input de type hidden des checkbox, pour récupérer tous les id dans un tableau.

```
<input name="'.$input.'[]" value="'.$id.'" type="hidden">
```

Dans la page FormPizza, on peut ainsi récupérer les différents id par une boucle *foreach*.

```
$inputs = !empty($_POST['idProduit']) ? $_POST['idProduit'] : array();

 foreach($inputs as $value) {

    $idProduit[]= $value;

 }
```

# VII. LA PWA

La PWA (Progressive Web Application) est une technologie qui permet à l'utilisateur de profiter d'application web de la même manière qu'une application native. On y accède de la même manière qu'un site web classique, directement sur un navigateur. Une fois sur le site, elle propose d'installer une icône sur l'écran d'accueil du smartphone ou de la tablette pour pouvoir y accéder directement. Les avantages par rapport aux applications native est qu'on n'est pas dépendant des App-Stores, et une utilisation moindre de la mémoire de l'appareil.

Les principes pour qu'une application soit identifiée comme une PWA sont les suivants :

- Elle doit être *discoverable,* c'est-à-dire qu'on doit pouvoir la trouver sur un moteur de recherche.
- Elle doit être *installable*, disponible sur l'écran d'accueil.
- *Linkable*, on doit pouvoir la partager simplement, en envoyant un lien.
- *Network independant*, elle doit pouvoir fonctionner hors-ligne ou avec une mauvaise connexion internet.
- *Progressive*, elle doit être utilisable sur les anciens navigateurs et complètement fonctionnelle sur les nouveaux.
- *Re-engageable*, des notifications doivent pouvoir être envoyées lorsque du nouveauc contenu est disponible.
- *Responsive*, elle doit être fonctionnelle sur n'importe quel appareil et résolution d'écran.
- *Safe*, la connexion doit être sécurisée.

Pour qu'une application soit considérée comme progressive, elle doit présenter plusieurs caractéristiques techniques.

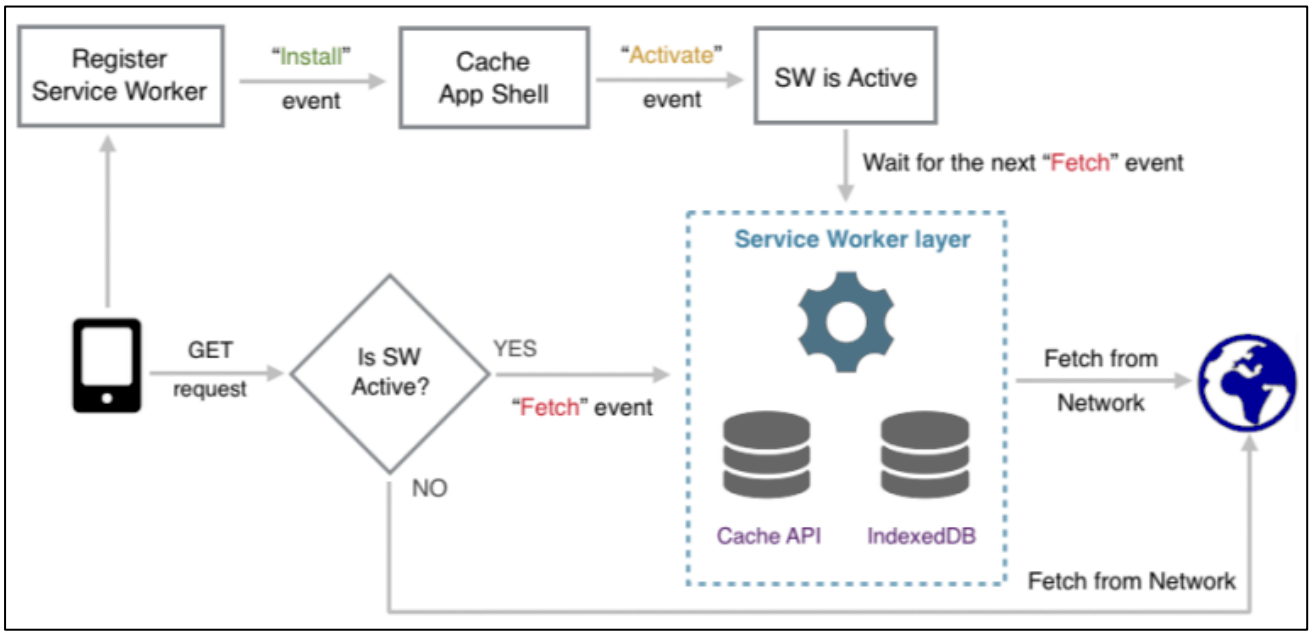
Tout d'abord, elle doit être *Secure contexts(htpps)*. C'est-à-dire qu'elle doit être servie sur un réseau sécurisé. Cela fait de l'application une application de confiance. Les fonctionnalités de la PWA ne sont disponibles que si le réseau est sécurisé.

Elle doit comporter un ou plusieurs *Service Workers*. Le Service Worker permet d'intercepter les requêtes entre l'application et le serveur, et de contrôler la mise en cache des ressources. De ce fait, la navigation est fluide et fiable. De plus il permet d'utiliser l'application hors-ligne.

Elle doit comporter un fichier *Manifest*. Il s'agit d'un fichier JSON qui contrôle comment l'application apparaît à l'utilisateur.

## 10. LE SERVICE WORKER

Le *Service Worker(sw)* est un script Javascript qui agit comme un serveur proxy. Il intercepte les requêtes entre l'utilisateur et le serveur.




Ce schéma nous présente le fonctionnemant du *sw*.

La première étape est de l'enregistrer. On entre un bloc de code dans le fichier JS principal de l'application.

```
if ('serviceWorker' in navigator)
{
    navigator.serviceWorker.register('sw.js');
};
```

Une fois le *sw* installé, l'événement *install* se déclenche, et remplit le cache local du navigateur avec les ressources nécessaires pour faire fonctionner l'application en hors-ligne.

Une fois cela fait, l'événement *fetch* permet de contrôler ce que le *sw* doit faire avec ce qui est mis en cache. Il se déclenche chaque fois qu'une requête est retournée.

On peut rattacher un *listener* sur l'événement *fetch* et appeler la méthode *respondWith()* pour détourner les réponses http.

## 11. LE FICHIER MANIFEST

Le fichier *manifest* est un fichier JSON qui contient les informations nécessaires pour la mise en place d'une PWA. Ces informations sont le nom de l'application, la couleur du thème, les icônes, l'url de l'application…

```
{
    "shortname": "pizzeria",
    "name": "o'speed pizza",
    "theme_color": "#f5f5f5",
    "background_color": "#f5f5f5",
    "display": "standalone",
    "Scope": "/",
    "orientation": "portrait",
    "icons": [
        {
            "src": "IMG/Pizza_icon.png",
            "type": "image/png",
            "size": "48x48"
        },
        {
            "src": "IMG/Pizza_icon2.png",
            "type": "image/png",
            "size": "96x96"
        },
        {
            "src": "IMG/Pizza_icon3.png",
            "type": "image/png",
            "size": "128x128"
        }
    ],

    "start_url": "index.php?page=accueil"
}
```

# VIII. PROBLÉMATIQUE

Mon questionnement était de savoir comment stocker les images dans la base de données.

J’ai donc consulté le forum **stackoverflow** pour savoir comment faire.

> Yes, you can store images in the database, but it's not advisable in my opinion, and it's not general practice.
>
> A general practice is to store images in directories on the file system and store references to the images in the database. e.g. path to the image,the image name, etc.. Or alternatively, you may even store images on a content delivery network (CDN) or numerous hosts across some great expanse of physical territory, and store references to access those resources in the database.
>
> Images can get quite large, greater than 1MB. And so storing images in a database can potentially put unnecessary load on your database and the network between your database and your web server if they're on different hosts.
>
> I've worked at startups, mid-size companies and large technology companies with 400K+ employees. In my 13 years of professional experience, I've never seen anyone store images in a database. I say this to support the statement it is an uncommon practice.

Oui, on peut stocker des images dans la base de données, mais ce n’est pas très avisé amon avis, et ce n’est pas une pratique répandue.
Habituellement, on stocke les images dans des dossiers dans l’oragnistion des fichier dans le système de l’application et on stocke les références des images dans la base de données, par exemple le chemin de l’image, son nom, etc… Ou encore, on peut aussi stocker les images dans un réseau de diffusion de contenu et stocker les références pour accéder à ces ressources dans la base de données.
Les images peuvent être assez volumineuses, plus que 1MB. Et stocker des images dans la base de données peut éventuellement mener à un chargement inutile entre sa base de données et son serveur si elles sont sur différents hébergeurs.
J’ai travaillé dans des start-up, des entreprises de taille moyenne et des grandes entreprises de plus de 400k employés. En treize ans d’expérience professionnelle, je n’ai vu personne stocker des images dans une base de données. Je dis cela pour insister sur le fait que ce n’est pas une pratique courante.

# IX. CONCLUSION

Ce stage m’a permis d’appréhender l’organisation et la rigueur nécessaire à l’exercice du métier de développeur web. En effet, se retrouver face à un projet d’une certaine ampleur seul nécessite d’ordonner sa réflexion et son travail de manière très rigoureuse. Malgré quelques problèmes dans ce domaine et des moments parfois difficiles, cette expérience fut très enrichissante, et j’en retire un enrichissement à la fois professionnel et personnel.

Au final, cette formation, en plus de m’avoir permis d’acquérir de solides connaissances, m’a apporter un grand plaisir sur le plan personnel. La cohésion de groupe , l’entraide entre tous les membres de la session a permis de surpasser les moments difficiles, aidé en cela par le soutien permanent de notre formatrice. Les travaux de groupe furent en cela très importants, nous apprenant à nous organiser pour travailler tous ensemble.

# REMERCIEMENTS

Je voudrais en premier lieu remercier Martine Poix, notre formatrice, pour nous avoir supporté et pour son soutien tout au long de la formation.

Merci aussi à tous mes collègues, pour tous les bons moments passés au cours de ces neufs mois, pour leur aide quand ça n’allait pas.

Je tiens aussi à remercier Antoine Reynot, qui a pris le temps de m’aider durant mon stage.

Enfin, merci à ma famille, pour m’avoir soutenu pendant neuf mois.