Introduction

Nous avions pour mission de créer un site web qui facilite la mise en relation entre les producteurs locaux de fruits et légumes et les consommateurs désireux d'acheter des produits frais et locaux. Ce site web permettra aux producteurs de mettre en avant leur offre de fruits et légumes, et aux consommateurs de trouver facilement des produits locaux de qualité. En favorisant la vente directe entre les producteurs et les consommateurs, notre site web vise à soutenir l'agriculture locale et à promouvoir une alimentation saine et durable.

Après la première soutenance, nous avons attribué des lots de travail à chaque membre du groupe, via une matrice RACI.

			Léo VESSE	Cosimo UNGARO	Thomas LERUEZ	Olivia AING
HTML	Page d'accueil	Généralités		RA		
		Carte des producteurs		RA		
		Fil d'actualité			RA	
	Interface producteur	Produits disponibles				RA
		Revenus de vente	RA			
		Messagerie instantanée			RA	
	Interface utilisateur	Interface de commande				RA
		Interface de suggestions	RA			
		Messagerie instantanée			RA	
BASE DE DONNÉES		Structure BD			RA	
CSS	Design	Charte graphique				RA
		Création des icônes				RA
		Création du logo				RA
	Programmation	Organisation des pages		A		R
ALGORITHMIE		Sélection de recettes	R	A		
		Parcours de donnés				
		Problème du voyageur				
		Dijkstra				

R : Responsable
A : Autorité
C : Conseiller
I : Informé

FIGURE 1 – Matrice RACI du projet

Web

1 Communication

Il nous a semblé nécessaire, pour des ventes de proximité, que clients et producteurs puissent directement discuter entre eux. Dans cette partie, nous allons voir comment les outils de communication entre les utilisateurs ont été mis en place. Il y a deux éléments principaux de communication : les messages et le fil d'actualité.

1.1 Messagerie

La messagerie reprend les codes d'une messagerie de base. Chaque utilisateur, qu'il achète ses produits ou qu'il les vende, a accès à son espace messagerie une fois connecté.

1.1.1 Réception (/messagerie/)

Une fois sur la page de sa messagerie, l'utilisateur peut voir la liste des mails présents dans sa boîte de réception. Les messages sont récupérés dans la base de données via une jointure entre la table "messages" et la table "utilisateurs" (voir volet BD). Si le message n'a pas été lu, la ligne commence par [NON LU] écrit en rouge. Ensuite vient le pseudonyme de la personne qui a envoyé le message, l'heure d'envoi, puis l'objet du message indiqué par l'expéditeur. Pour consulter un message, l'utilisateur clique sur l'objet : c'est un lien cliquable qui ajoute /id à la fin de l'URL, avec id l'identifiant du message (voir volet BD). De cette manière, si l'utilisateur a effectivement accès à ce message (vérification effectuée dans la requête SQL pour éviter qu'un utilisateur ne puisse lire tous les messages du site), le message sur lequel il a cliqué lui est montré dans la même fenêtre que sa boîte de réception. Le message est automatiquement marqué comme lu via une requête UPDATE.

Depuis sa boîte de réception, l'utilisateur peut également sélectionner plusieurs messages et les supprimer. J'ai fait en sorte que les messages ne soient pas réellement supprimés de la base de données, mais plutôt masqués pour l'utilisateur, en vue d'un éventuel ajout d'une corbeille. Cet

ajout n'a pas été fait, j'ai cependant pris la décision de garder le fait de masquer les messages plutôt que de les supprimer, pour des raisons de modération. Si l'on veut réellement supprimer ces messages, il suffit de remplacer la requête UPDATE par une requête DELETE.

La boîte de réception comporte également un bouton "Écrire un message", qui permet à l'utilisateur d'être redirigé vers la page d'envoi de messages. Il peut également cliquer sur "Répondre" sur le message qu'il est en train de lire s'il en lit un, ce qui va le rediriger vers la page d'envoi de messages avec l'objet automatiquement rempli : "Re : " suivi de l'objet du message de base.

1.1.2 Envoi (/ecrire)

Sur la page d'envoi, l'utilisateur qui souhaite envoyer un message doit indiquer le pseudonyme de la personne à qui il souhaite envoyer le message (s'il n'est pas correct, l'utilisateur aura un message d'erreur), ainsi qu'un objet et le corps du message. J'ai volontairement mis un input de type "text" pour l'écriture du corps du message, plutôt qu'un textarea comme on pourrait s'y attendre, dans le but d'encourager l'utilisateur à écrire des messages plus courts, dans l'esprit de ceux que l'on peut envoyer sur les réseaux sociaux sur une seule ligne, pour réduire la taille de la base de données et diminuer la quantité d'informations à envoyer. Une fois son message écrit et tous les champs nécessaires remplis, l'utilisateur peut appuyer sur le bouton "envoyer". Flask va récupérer les données envoyées par le formulaire avec la méthode POST, et les inscrire dans la base de données.

Ce module d'envoi de messages aurait dû être présent sur les pages des producteurs, mais ce n'est pas le cas. Il est donc présent sur /ecrire, dans la version dans laquelle il aurait dû être sur les pages des producteurs, avec un champ "Destinataire" pour indiquer à qui ce message doit être envoyé.

1.2 Fil d'actualité (/publications)

Une fois connecté, chaque utilisateur peut aller sur son fil d'actualité.

Il peut tout d'abord poster une publication : il entre le message à publier, puis clique sur "Poster". Sa publication est ensuite insérée dans la base de données.

Il y a un système d'abonnement : chaque utilisateur peut "suivre" un autre utilisateur, et verra ainsi ses publications apparaître dans son fil d'actualité. Dans mon code, l'utilisateur se suit automatiquement lui-même, c'est-à-dire que ses propres publications apparaissent dans son fil d'actualité. Cela n'est pas inscrit dans la base de donnés (table "follow", voir volet BD), mais est fait directement via la requête SQL qui sert à récupérer les messages.

Ainsi, pour chaque publication qu'il peut voir, l'utilisateur a accès au pseudonyme de l'utilisateur qui a posté la publication, ainsi que la date de publication, en plus du message.

Il y a également un système de commentaires pour chaque publication : l'utilisateur peut commenter une publication, et son commentaire sera visible par toutes les personnes qui suivent l'individu qui a posté la publication. Pour récupérer les commentaires des publications, j'ai fait une boucle sur chaque publication qui sera récupérée par l'utilisateur, et pour chaque publication, je fais une requête SQL comportant un JOIN entre la table commentaires et la table utilisateurs (voir volet BD) et en filtrant sur l'identifiant de la publication passée en revue. Il me semble important de parler de la complexité de cette méthode ici : je pense que celle-ci est (trop) lourde : on a du $O(n)$ pour la récupération des publications, puis pour chaque publication on fait une jointure entre deux tables et on filtre les commentaires correspondant à la publication. J'ignore si la jointure est refaite à chaque fois ou si celle-ci est stockée en cache ou quelque chose de la sorte, mais cette méthode me semble au moins être en $O(n^2)$. Je n'ai pas trouvé d'alternative à cette méthode, j'ai cependant conscience que si mon évaluation de la complexité est correcte, cette méthode de récupération des commentaires n'est pas viable avec une quantité importante de publications/commentaires.

1.3 Producteurs les plus proches (/jardins)

Cette page a pour but de permettre à l'utilisateur de trouver les producteurs les plus proches de lui. Il entre l'adresse où il se trouve, et on lui retourne les producteurs les plus proches, avec leur pseudonyme et une carte permettant de savoir où se trouve l'utilisateur. Côté Python, l'adresse est récupérée, envoyée à une API du Gouvernement qui retourne des données en JSON : ce qui nous intéresse ici est l'adresse (qui peut être légèrement corrigée), et les coordonnées GPS de l'adresse entrée. On calcule ensuite la distance à vol d'oiseau en kilomètre grâce à une formule trigonométrique. De cette manière, après un tri (détaillé dans le volet algorithmie), l'utilisateur

voit apparaître, dans l'ordre croissant de distance à vol d'oiseau, les producteurs les plus proches de lui dans un rayon de 50 km.

2 Page d'accueil

2.1 HMTL

Il n'y a rien de particulier à noter sur HTML ; la page, par souci de compréhension, n'est constituée que de deux boutons. Seul quelques div ont étées ajoutées afin de pouvoir gérer le 'header'.

2.2 CSS

De la décoration à été ajoutée aux boutons. On étudiera ici le bouton d'inscription, les deux étant codés de la même manière.
On commence par définir les propriétés de base, propre au bouton :

```
    .inscription{
margin-top: 0.25rem;
color:#FFF;
}
```

On définit ensuite les propriétés 'au repos', c'est-à-dire les propriétés du bouton par défaut, sans animation. Pour obtenir l'animation finale, on créé deux cadres verts 'avant' et 'après' l'élément(ceux-ci seront en fait au niveau de l'élément). Pour finir, on indique la durée de la transition, soit ici 0.3s :

```
.inscription::after, .inscription::before{
border: 3px solid #ABD664;
content: "";
position: absolute;
width: calc(200px - 6px);
height: calc(60px - 6px);
left: 0;
bottom: 0;
z-index: -1;
transition: transform 0.3s ease;
}
```

On indique ensuite quelle animation jouer lors du survol par la souris :

```
.inscription:hover::after{
transform: translate(-5px, -5px);
}
```

Puis on fait l'animation 'retour' : on définit le comportement des objets lorsque la souris quitte le cadre :

```
.inscription:hover::before{
transform: translate(5px,5px);
}
```

3 Page d'inscription

3.1 HTML

Rien de particulier au niveau du HTML ; il s'agit d'un formulaire classique. Quelques "div" sont ajoutées pour un effet CSS ultérieur (par exemple afin de créer le 'header')

3.2 CSS

3.2.1 Gestion du Header

Les principales lignes pour la mise en forme du header sont les suivantes :

z-index : +10 ;
position : fixed ;
top : 0px ;

Ainsi la première ligne agit sur le "z-index", c'est-à-dire la couche sur laquelle se situe l'élément ; en le fixant à +10 on est ainsi sur qu'aucun élément ne viendra s'y supperposer.
La seconde ligne permet de régler la position a "fixed". Ainsi l'élément ne vient pas se placer relativement aux autres éléments mais uniquement par rapport à l'écran.
Enfin, la dernière ligne top : 0px ; indique à l'élément de se placer tout en haut de la page ; Combiné aux balises précédentes, cela nous permet donc bien de générer un header

3.2.2 Exemple d'un élément de formulaire de type "label"

Divers éléments de décorations ont été ajoutés aux champ de saisie de texte (input field - label). Nous présenterons donc ici le code ayant servi de base à tout les éléments.

La principale décoration consiste à modifier la bordure. Ainsi, en changeant la bordure du bas lors du survol (hover), on obtient l'effet suivant (le code a été allegé pour en faciliter la compréhension) :

```
.label{
<éléments classiques: position, taille, couleur...>
}

.label:hover{
border-style: solid;
border-color: #ABC664 #ABC664 green #ABC664;
}
```

3.3 Python - Flask

Afin d'éxecuter du code sur l'application web nous avons, conformément aux instructions de l'équipe pédagogique, mis en place un environnement Flask. Voici une explication du code lié à la page d'inscription :

Analysons ce code ligne à ligne :

```
@app.route("/inscription", methods=['GET', 'POST'])
```

La première ligne nous informe que ce qui suit (soit ici la fonction 'insc') doit être éxécutée lorsque l'on tente d'accéder à <adresse>/inscription. methods=['GET','POST'] nous permet d'accéder aux éléments du formulaire.

```
def insc():
    msg = ''
    if request.method == 'POST' and request.form.get('first_name') and request.form.get('name') an
```

La deuxième ligne déclare la fonction insc.La troisième nous permet de modifier un message, pour l'instant vide, sur la page d'accueil. On pourra par la suite modifier ce message afin d'indiquer à l'utilisateur la raison pour laquelle la connexion a échouée. La quatrième teste si les champs first_name, name, et password ont bien été remplis par l'utilisateur avant que celui-ci ne tente d'envoyer le formulaire. On vérifie également que le premier et le second mot de passe correspondent.
Si tel est le cas on entre alors dans le if :

— On commence par ouvrir la connexion à la base de données. On récupère ensuite le maximum atteint par le champ 'id', afin de pouvoir donner à notre utilisateur un id correspondant à cet indice augmenté de 1 :

```
rid = get db().cursor()
rid.execute('SELECT MAX(id) FROM utilisateurs')
new_id = rid.fetchall()[0][0]+1
```

— On récupère ensuite les valeurs entrées dans les champs 'first_name', 'name', 'pseudo', 'email', 'phone number' et 'password' dans des variables (respectivement nom, prenom, pseudo, adresse_mail, num_de_tel et mdp) :

```
nom = request.form.get('first_name')
prenom = request.form.get('name')
pseudo = prenom = request.form.get('pseudo')
adresse_mail = request.form.get('email')
num_de_tel = request.form.get('phone_number')
mdp = bytes(request.form.get('password'), 'utf-8')
```

— On hash ensuite le mot de passe grâce à la fonction SHA-256. Cela nous permettra, grâce aux propriétés de cette fonction, de garantir la sécurité des mots de passe :

```
mass = hashlib.sha256(mdp).hexdigest()
```

— Enfin, on ajoute l'ensemble des données dans la base de données grâce au SQL. On différencie le cas où l'utilisateur est un producteur de celui ou il est un client :

```
if request.form.get('prod'):
    cur.execute('INSERT INTO utilisateurs VALUES(?,?,?,?,?,?,?,?)', (new_id,
                prenom, nom, adresse_mail, num_de_tel, mass, 2, pseudo))
else:
    cur.execute('INSERT INTO utilisateurs VALUES(?,?,?,?,?,?,?,?)', (new_id,
                prenom, nom, adresse_mail, num_de_tel, mass, 1, pseudo))
```

— On ferme alors la connexion à la base de données. L'idée d'utiliser session pour identifier les utilisateurs ayant été retenues, on règle donc session["id_utilisateur"] à l'id de la personne. Enfin, on renvoie la page :

```
conn.commit()
conn.close()
session["id_utilisateur"] = new_id
return redirect(url_for("renvoyer_dashboard_util"))
```

— Enfin, s'il y a eu un problème dans le remplissage du formulaire, on modifie le message de la page :

```
elif request.method == 'POST':
    msg = "Les mots de passe ne correspondent pas"
return render_template('inscription.html', message=msg)
```

4 Page de connexion

4.1 HTML

Rien de particulier au niveau du HTML ; il s'agit d'un formulaire classique. Quelques "div" sont ajoutées pour un effet CSS ultérieur (par exemple afin de créer le 'header')

4.2 CSS

4.2.1 Gestion du Header

Pareil que pour la page d'inscription.

4.2.2 Exemple d'un élément de formulaire de type "label"

Pareil que pour la page d'inscription.

4.3 Python - Flask

Afin d'éxecuter du code sur l'application web nous avons, conformément aux instructions de l'équipe pédagogique, mis en place un environnement Flask. Voici une explication du code lié à la page de connexion :

Analysons ce code ligne à ligne :

```
@app.route("/connexion", methods=['GET', 'POST'])
def connexion_page():
```

```
if request.method == 'POST' and request.form.get('name') and request.form.get('password'):
    r = get_db().cursor()
```

On commence par indiquer que la fonction connexion_page doit s'éxecuter lors de l'accès à 'connexion'. Dans cette fonction, on récupère les champs de saisie (méthode 'POST'), et on vérifie avant toute chose que les champs nom et mot de passe ont bien été saisis.

```
nom = request.form.get('name')
mdp = bytes(request.form.get('password'), 'utf-8')
mass = hashlib.sha256(mdp).hexdigest()
r.execute(''SELECT motdepasse FROM utilisateurs WHERE pseudo=?'', (nom,))
tuple = r.fetchall()
```

On récupère tout d'abord les champs 'name' et 'password', que l'on stocke respectivement dans 'nom' et 'mdp'. On calcule alors le hash du mot de passe entré (grâce à la bibliothèque hashlib) et on récupère via le curseur r le hash du mot de passe associé dans la base de donnée . Cela nous permet, dans la prochaine ligne, de comparer les deux :

```
if tuple[0][0] == mass:
    r = get db().cursor()
    r.execute(''SELECT id FROM utilisateurs WHERE pseudo=?'', (nom,))
    tuple = r.fetchall()[0][0]
    session[''id utilisateur''] = tuple
```

Si tel est le cas, on récupère l'id utilisateur dans la base de donnée, et on le stocke grâce à session (afin d'éviter à l'utilisateur d'avoir à se reconnecter à chaque rafraîchissement de la page).

```
rsession = get db().cursor()
rsession.execute(''SELECT statut FROM utilisateurs WHERE id=- ''.format(
session[''id utilisateur'']))
session type = rsession.fetchall()[0][0]
if session type == 1:
    return redirect(url for(''renvoyer dashboard util''))
elif session type == 2:
    return redirect(url for(''renvoyer prod''))
```

On récupère ensuite le type de compte associé (utilisateur ou producteur) et on renvoie la page associée.

5 Interface principale - côté client

5.1 Carte

Nous avons décidé de centrer notre application web autour de la carte, censée indiquer les producteurs les plus proches. Malgré l'absence d'API nous permettant de mener à bien notre idée initiale (du moins, d'API gratuite), nous avons pris le parti de laisser la carte afin d'avoir une idée de son emplacement et de sa dimension ; en revanche, aucune modification / aucun ajout n'est possible par l'utilisateur.

5.2 Commande

5.2.1 HTML

Une grande partie du code est enveloppée dans du block content afin de pouvoir interagir avec lui.
On commence ensuite par régler le message par défaut, au cas où il n'y aurait pas de commande en cours :

```
{% if not liste_commandes %}
    <h1 class="commande">Vous n'avez rien commandé</h1>
{% endif %}
```

Puis, on affiche les éléments commandés s'il y a une commande :

```
{% for element in liste_commandes %}
        <img src={{photo[liste_commandes.index(element)]}} class="commande-photo">
        <h1 class="commande">{{ element[4] }}€</h1>
{% endfor %}
```

5.2.2 Python

On commence par vérifier si l'utilisateur est connecté avant d'éxecuter :

```
    @app.route("/utilisateur/dashboard")
def renvoyer_dashboard_util():
    if "id_utilisateur" in session:
        id = session["id_utilisateur"]
```

On récupère ensuite les commandes de l'utilisateur.

```
    r = get_db().cursor()
        r.execute("SELECT * FROM commande WHERE id_util={}".format(id))
        liste_commandes_tuple = r.fetchall()
        lst_photo = []
```

On récupère ici un tuple (requête SQL). Il faut donc, afin de pouvoir travailler dessus, les ranger dans des listes :

```
        rphoto = get_db().cursor()
        photos = []
        for e in lst_photo:
            rphoto.execute("SELECT photo from produits where id={}".format(e))
            val = rphoto.fetchall()
            photos.append(val[0][0])
```

Enfin, on voudrait calculer la valeur totale de la commande, afin que l'utilisateur sache combien il a a payer. Pour cela, on parcourt la liste de commande, et on ajoute à chaque fois la quatrième coordonnée du tuple (voir la structure de la base de données) :

```
    for element in liste_commandes_tuple:
        lst_photo.append(element[1])
        a_payer += element[4]
```

On récupère de la même façon les photos :

```
    r2 = get_db().cursor()
    r2.execute("SELECT photo FROM produits WHERE id={}".format(id))
    rnom = get_db().cursor()
```

Et enfin le prénom de l'utilisateur, afin de pouvoir personnaliser la page :

```
    rnom.execute("SELECT Prenom FROM utilisateurs WHERE id={}".format(id))
    nom = rnom.fetchall()[0][0]
```

Il ne nous reste alors plus qu'à afficher la page, et à gérer l'exception du cas où l'utilisateur n'est pas connecté :

```
    return render_template("dashboard_util.html", nom=nom, liste_commandes=liste_commandes_tuple,
    else:
        return render_template("connexion.html")
```

6 Dashboard producteur

Le dashboard producteur est sensiblement construit sur le même modèle que le dashboard utilisateur

Base de données

La base de données a été construite autour de la table “utilisateurs”. Nous allons détailler la construction des différentes tables de cette base de données, ainsi que les éventuelles relations entre elles, puis nous verrons la normalisation de ces relations.

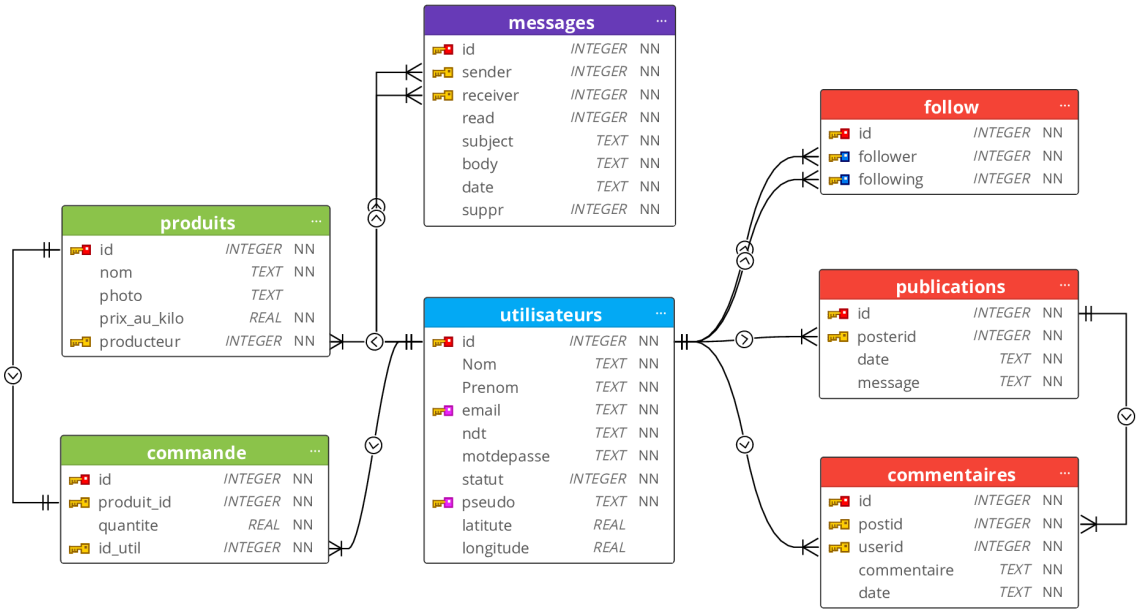


FIGURE 2 – Diagramme de la base de données

7 Construction de la base de données

Dans cette partie, nous allons expliquer la construction de la base de données : signification des différents éléments des tables, et relations éventuelles entre ces tables. La couleur du titre de chaque sous-partie est associée à la couleur du groupe de tables (fig. 2) concerné par cette sous-partie.

7.1 Utilisateurs

Pour chaque inscription, nous demandons un nom, un prénom, un pseudonyme, une adresse e-mail, un numéro de téléphone, un mot de passe, le choix entre un compte producteur ou acheteur, et une adresse (latitude, longitude). Ces informations étant susceptibles de changer (changement ou erreur dans le nom par exemple), nous avons choisi d’identifier les comptes utilisateurs par un identifiant unique (un entier naturel) attribué automatiquement à l’inscription, et constituant donc notre clé primaire. Toutes ces données sont stockées dans la table “utilisateurs”. De cette manière, toutes les autres tables qui ont un rapport direct avec les utilisateurs peuvent les identifier tout en étant à l’abri d’éventuels changements dans les données des utilisateurs.

7.2 Vente de produits

Les produits qui sont à vendre ou qui ont été vendus par les producteurs sont stockés dans la table “produits”. Un identifiant unique (entier naturel) est affecté à chaque produit, ainsi qu’un nom, un lien vers une photo, son prix au kilo (0 dans le cas d’une vente à l’unité) et l’identifiant du producteur qui vend ce produit.

Les commandes sont stockées dans la table “commande”. Un identifiant unique (entier naturel) est affecté à chacune de ces commandes, qui comportent également l’identifiant du produit acheté, sa quantité (un nombre entier si la vente se fait à l’unité, une masse si la vente se fait au kilo), et l’identifiant de l’utilisateur qui l’a acheté.

7.3 Communication

7.3.1 Fil d'actualité

Les publications des utilisateurs sont stockées dans la table "publications". A chaque publication est affecté un identifiant unique (entier naturel), ainsi que l'identifiant de l'utilisateur qui publie, la date de publication, et le message publié.

Chaque commentaire est stocké dans la table "commentaires", qui comporte un identifiant unique (entier naturel) affecté au commentaire, l'identifiant de la publication commentée, l'identifiant de l'utilisateur qui a commenté, le texte du commentaire et la date d'envoi de ce commentaire.

L'utilisateur choisit quel producteur il "suit", c'est-à-dire de quels producteurs il veut avoir les publications affichées dans son fil d'actualité. Pour cela, la table "follow" recense, pour chaque abonnement, un identifiant unique de cette relation (entier naturel), l'identifiant de l'utilisateur qui suit et celui de l'utilisateur suivi.

7.3.2 Messagerie

Les messages sont stockés dans la table "messages". Chacun de ces messages a un identifiant unique (entier naturel), l'identifiant de l'utilisateur qui envoie le message, l'identifiant de celui qui le reçoit, un entier servant de booléen (car d'après mes recherches, il n'y a pas de type booléen à proprement parler dans sqlite, il faut donc utiliser un entier naturel) qui sert à savoir si l'utilisateur a lu le message ou non (par défaut il est donc sur 0, soit l'état "non lu"), le sujet et le corps du message, la date d'envoi et un entier naturel servant de booléen qui est sur 1 si le receveur du message a décidé de masquer le message (lorsqu'il l'a lu par exemple). Il est donc sur 0 par défaut.

8 Formes normales

Notre base de données respecte les conditions requises pour être en 3NF. Nous allons détailler dans cette partie les différentes raisons pour lesquelles toutes les conditions des 1NF, 2NF et 3NF sont remplies.

8.1 Première forme normale (1NF)

Pour être en première forme normale, une relation doit posséder au moins une clé primaire et tous ses attributs doivent être atomiques.

Comme nous pouvons le voir sur le diagramme (fig. 2), chaque table contient un attribut "id" qui constitue la clé primaire de cette table. Ces attributs "id" sont en auto-incrément. De plus, chaque attribut est atomique : c'est pour cela que la table "commande" a été créée (pour ne pas directement inscrire la liste des commandes dans la table "produits"), ainsi que la table "commentaires" (pour ne pas avoir une liste de commentaires pour chaque publication dans la table "publications"). Il en est de même pour la table "follow", qui sert à ne pas stocker une liste d'utilisateurs suivis pour chaque client dans la table "utilisateurs".

Ainsi, les conditions sont remplies, la base de données respecte la première forme normale.

8.2 Deuxième forme normale (2NF)

Pour être en deuxième forme normale, une relation doit être en première forme normale et tout attribut n'appartenant pas à la clé ne doit pas dépendre d'une partie seulement de la clé. Toutes les relations sont en 1NF d'après la sous-partie précédente. Les clés primaires de toutes les tables sont composées d'un attribut uniquement, donc le respect de la deuxième forme normale est assuré.

8.3 Troisième forme normale (3NF)

Pour être en troisième forme normale, une relation doit être en deuxième forme normale et tout attribut n'appartenant pas à la clé ne doit pas dépendre fonctionnellement d'un attribut qui n'appartient pas à la clé. Toutes les relations sont en 2NF d'après la sous-partie précédente. D'après le diagramme de la base de données et l'explication de la construction de celle-ci dans la partie précédente, dans une table donnée, aucun attribut ne dépend d'un autre, à part de la clé primaire. Le respect de la troisième forme normale est donc assuré.

Algorithmie

Après avoir entré une adresse (voir volet Web, /jardins), les producteurs les plus proches sont récupérés (valeur fixée à 50 km à la ronde dans le code). Ensuite, ils sont classés par distance : on a donc une liste de liste d'informations des producteurs trouvés (comprenant leur distance à l'utilisateur). J'ai donc modifié l'algorithme classique de tri fusion, pour qu'il soit capable de trier une liste de listes selon la case de la liste correspondant à la distance à l'utilisateur. On a donc une complexité en $O(nlog(n))$.

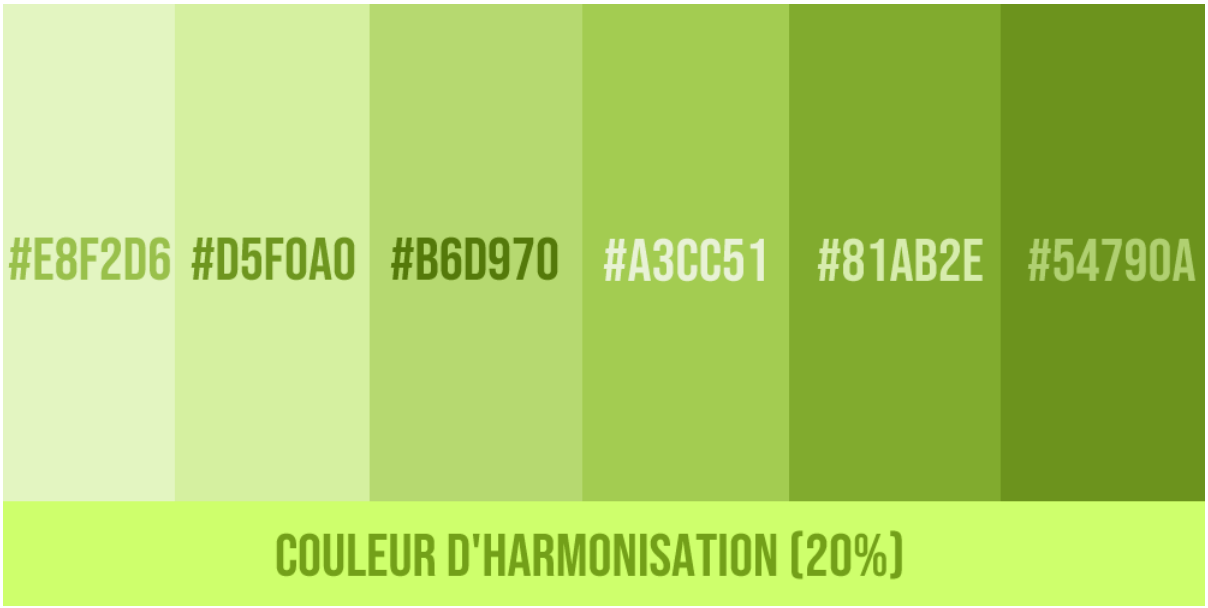
Graphisme

Charte Graphique

Palette de couleur

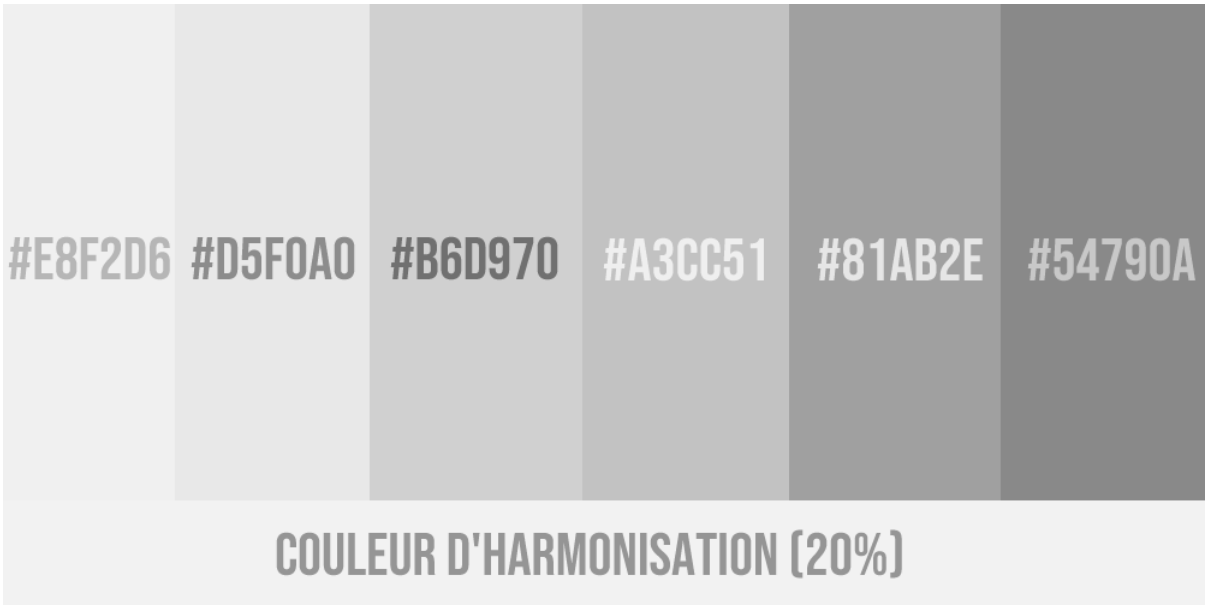
Afin de créer un site dont le visuel était à la fois plaisant et intuitif, il était important de choisir des couleurs qui permettaient d'avoir un contraste élevé, même en niveau de gris, mais aussi agréables, tout en rappelant l'utilité du site.
Pour cela, plusieurs chartes graphiques ont été proposées, et votées au sein de l'équipe.

La palette de couleur choisie a été renommée "Vert pomme", et elle est donc constituée des couleurs qui seront principalement retrouvées sur le site.



Palette Pomme

Comme dit précédemment, cette palette a été également conçue afin d'être assez contrastée dans le cas d'un daltonisme : ainsi, bien que cette palette soit composée de couleurs dans des tons verts, le contraste entre les nuances soit suffisant pour distinguer les écritures du fond.



Palette Pomme en niveau de gris

Logo du site

Une fois la palette de couleur établie, nous nous sommes penchés sur la question du nom, et du logo pour notre site. Après plusieurs minutes de réflexions, nous avons fini par choisir le nom de TELEPOMME d'ici. Un rappel phonique à cette école qui est à la naissance du projet, tout en étant un écho à notre palette nouvellement choisie. Puis, finalement, le logo se présenta presque comme une évidence.

Logo de TELEPOMME d'ici

Quelques tests avec un visuel du site fait sur un logiciel de dessin a permis d'assurer une harmonisation du logiciel avec le reste de la charte graphique.

Exemple d'une partie du site

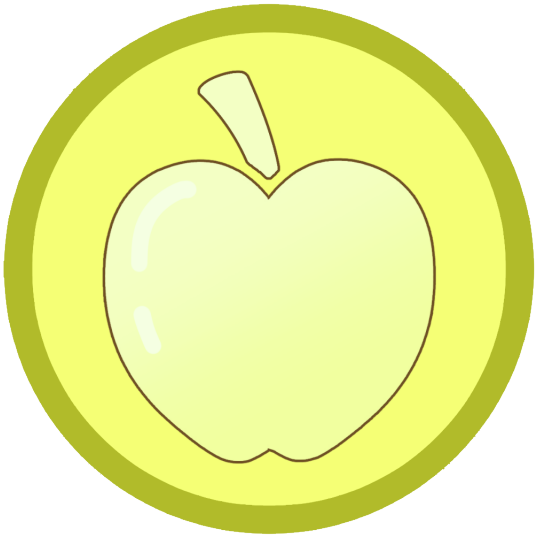
Icones de fruits et légumes

Lors de la réunion du 10/12/2022, alors que nous parlions de l'égalité entre les différents vendeurs de la plateforme, il a été décidé que les images de fruits, légumes et autres produits, seraient remplacés par des icones.
Ainsi, plusieurs propositions d'icones ont été posées, différant par leur style et leur couleurs, jusqu'à ce que soit choisie la forme finale.

Icone de la pomme

Icone de la tomate

Enfin, il a été décidé que comme il n'était pas impossible qu'une icone soit manquante, des icones par défaut seraient créées.



Icone de fruit par défaut

Compte-rendus de réunion

Projet Pluridisciplinaire d'Informatique Intégrative 1
PPII 1
1A TELECOM Nancy - 2022-2023

Compte-rendu de réunion

Lundi 10 Octobre 2022

---

1 Caractéristiques de la réunion

1.1 Informations générales

- Type : Réunion de chantier
- Lieu : Discord, en ligne
- Début : 20 h 00
- Durée : 01 h 00

1.2 Membres présents

- Léo VESSE *(Chef de Projet)*
- Olivia AING *(Secrétaire)*
- Cosimo UNGARO
- Thomas LERUEZ

1.3 Ordre du jour

- REPRISE DES ÉLÉMENTS À TRAITER : renseignements, noms et CGU
- Lecture du brouillon de l'État de l'Art
- Consultation de l'enchaînement des pages web

1.4 Backlog

Léo VESSE	Contacter l'AMAP qui est en lien avec Marché	Blocage : N'a pas pu trouver la présidente de Marché
Olivia AING	Faire la liste des contacts des AMAP / producteurs	Blocage : N'a pas eu le temps de le faire depuis la veille
	Faire la page de couverture de l'État de l'Art	Blocage : Ne sait pas dans quelle direction aller
Cosimo UNGARO	Se renseigner sur ce qui existe : applications, sites, associations	Terminé (contenu dans le brouillon de l'État de l'Art)
	Appeler les contacts pour diverses informations	Blocage : Week-end
Thomas LERUEZ	Se renseigner sur ce qui existe : applications, sites, associations	En cours : S'informe sur des associations listées sur un site gouvernemental

2 Échanges au cours de la réunion

Une fois tout le monde présent et les problèmes de son (plus ou moins) réglés, la réunion débute. Cosimo nous fait passer le premier brouillon de l'état de l'art.

Se renseigner sur les plateformes existantes

Cosimo et Thomas se mettent d'accord pour chercher des informations sur le plus de sites de circuits courts sur la liste donnée par Cosimo.

Cosimo a aussi fouillé le site officiel des AMAP, et s'est penché sur les paiements : il n'y a pas de site à proprement parler, les producteurs doivent se débrouiller avec des e-mails et des brochures. Pour les paiements, les chèques sont conseillés pour laisser une trace.

Cosimo loue Leaf. Léo ajoute que ce serait bien d'en plus de récupérer les fonctionnalités, faire des recherches un peu plus en profondeur.

Cosimo propose de faire la plateforme de paiement plus tard. Olivia propose de passer par PayPal, soutenue par Thomas.

Nom de l'équipe et du site

Nous nous mettons d'accord sur le fait que ce n'est pas nécessaire dans l'immédiat. Cela aura lieu à un brainstorming ultérieurement.

Écriture des Conditions Générales d'Utilisation

Ce n'est pas nécessaire dans l'immédiat non plus. Cosimo propose de faire cela plus tard, et Olivia à la fin. Cosimo propose d'utiliser un générateur (semblable à celui pour le dropshipping), et Léo propose de le développer nous-même. Olivia propose de demander de l'aide à Mme Heurtel lorsque le temps viendra. Elle rajoute qu'il faudra noter au fur et à mesure du projet les points qu'il faudra aborder dans les CGU.

Thomas rajoute qu'il faudra faire les RGPD (tout ce qui concerne les droits d'exploitation et de conservation des données personnelles). Léo propose de mettre tout ça en place sous la forme d'un arbre ou d'une mind map.

Lecture du document brouillon de l'état de l'art

Le document a été validé, la construction et le contenu convient à l'ensemble de l'équipe (Merci Cosimo !)

Consultation de l'enchaînement des pages web

La première version est disponible sur l'état de l'art.

- Page d'accueil (Cosimo)
  Olivia propose de mettre le total de CO2 et le total d'argent qui a été reversé aux producteurs sur la page d'accueil, pour montrer aux utilisateurs le résultat de leurs efforts, en chiffres.

  Pour la carte, celle de Google Maps est payante, donc il faudra en trouver une autre.
- Dashboard client (Cosimo)
  Thomas propose de mettre une bande avec les grammes de CO2 économisé.

  Cosimo ajoute qu'il faudrait mettre l'argent économisé en ne passant pas par des distributeurs.

  Léo complète avec le montant que l'utilisateur a reversé aux producteurs.
- Dashboard producteur (Cosimo)
  Rien ajouté.

- Page d'informations (Léo)
  Léo propose de rajouter une page d'information pour pouvoir nous contacter en cas de souci. On créerait alors également un support ou un feedback à travers les coordonnées que l'on mettra pour le site.

  Thomas ajoute qu'il faudrait faire un ticket côté client et côté producteur pour reporter les éventuels problèmes.
- Sécurité
  On impose une sécurisation par double authentification (via numéro de téléphone).

  Ainsi, en cas de souci, on peut blacklister le numéro de téléphone / l'adresse IP / l'adresse MAC...

  Thomas dit que ça ne sera probablement pas possible, mais Olivia répond qu'au moins, on en aura parlé.
- Autre
  Design

  Olivia est chargée de faire les icones de fruits et légumes

  Logistique

  Le financement du projet se ferait par dons, ainsi, tout l'argent récolté peut être restitué aux producteurs. Il faudra créer une adresse mail officielle pour recevoir les feedbacks.

  L'idée de départ, interprétée par Olivia, était de faire une plateforme de producteur à particulier dans un premier temps, pour garantir les transactions.

3 Prochaine réunion

Date prévue :

Dimanche 16 octobre 2022

Lieu :

Discord, en ligne

Ordre du jour (non exhaustif) :

- GESTION DE PROJET : Work breakdown, matrice RACI et diagramme de GANTT
- DESIGN : Choix et validation de la charte graphique
- INFORMATIONS : Plus de détails sur les différentes associations

TO-DO list :

Léo VESSE	Contacter l'AMAP de l'école (via Marché)
	Faire le breakdown des tâches
Olivia AING	Faire la page de couverture de l'État de l'art
	Commencer à faire des icones
	Faire la matrice RACI et le Gantt une fois le breakdown fait
Cosimo UNGARO	Vérifier le plus de sites possibles
Thomas LERUEZ	Vérifier le plus de sites possibles

FIN DE LA RÉUNION

Projet Pluridisciplinaire d'Informatique Intégrative 1
PPII 1
1A TELECOM Nancy - 2022-2023

Compte-rendu de réunion

Dimanche 09 Octobre 2022

---

1 Caractéristiques de la réunion

1.1 Informations générales

- Type : Réunion préliminaire
- Lieu : Salle à manger de la collocation
- Début : 20 h 00
- Durée : 00 h 30

1.2 Membres présents

- Léo VESSE *(Chef de Projet)*
- Olivia AING *(Secrétaire)*
- Cosimo UNGARO
- Thomas LERUEZ

1.3 Ordre du jour

Pas d'ordre du jour.

2 Échanges au cours de la réunion

Rappel des consignes de l'épreuve

Léo rappelle que tout le monde doit travailler sur tous les domaines (dev, recherche, bd, ...).
Cosimo ajoute que nous sommes notés sur la participation sur le GitLab.
Thomas rappelle (avec amertume) que nous n'avons pas le droit au php, et qu'il faudra démontrer la complexité de nos algorithmes.

Les fonctionnalités de notre site

Cosimo parle de la priorisation des lieux proposant plusieurs produits recherchés, mais un peu plus loin, par rapport ceux n'en proposant qu'un seul mais étant plus proche.
Léo et Thomas font le parallèle avec Dijkstra.

Thomas parle de système de rank, pour distinguer les particuliers des professionnels.

On souligne qu'il faudra que notre site soit compatible avec les portables.

En ouverture, on parle d'application mobile.

3 Prochaine réunion

Date prévue :

Lundi 10 Octobre 2022

Lieu :

Discord, en ligne

Ordre du jour (non exhaustif) :

- REPRISE DES ELEMENTS À TRAITER : voir l'avancement des tâches

TO-DO list :

Léo VESSE	Contacter l'AMAP qui est en lien avec Marché
Olivia AING	Faire la liste des contacts des AMAP / producteurs
	Faire la page de couverture de l'État de l'Art
Cosimo UNGARO	Se renseigner sur ce qui existe : applications, sites, associations
	Appeler les contacts pour savoir : • ce qu'ils utilisent • ce qui les intéresserait (comme fonctionnalités) • ce dont ils auraient besoin
Thomas LERUEZ	Se renseigner sur ce qui existe : applications, sites, associations

FIN DE LA RÉUNION

Conclusion

Notre équipe a subi des difficultés à cause de différents problèmes de santé. Cela a mené à l'arrêt complet des quatre membres, de manière synchronisée ou non, afin de leur laisser la possibilité de récupérer. À cause de cela, nous n'avons pas été en mesure d'accomplir les tâches que nous nous étions données aussi bien que nous l'aurions voulu. La matrice RACI n'a pas été suffisamment respectée. Les pages ne sont pas assez liées entre elles, et l'habillage de certaines pages n'est pas terminé. Cependant, la plupart des modules que nous avons fait sur le site sont fonctionnels.