



EXCEPTION TEA

Dossier de projet

Candidat Néo Masson
Chef de projet : Anass Benfares

Expert 1 : Suleyman Ceran
Expert 2 : Claude Albert Muller Theurillat

Table des matières

1	Analyse préliminaire.....	4
1.1	Introduction.....	4
1.2	Objectifs	4
1.2.1	Respect du modèle MVC avec Laravel.....	4
1.2.2	Qualité et lisibilité du code	4
1.2.3	Processus d'inscription et d'authentification complet.....	4
1.2.4	Sécurité des accès et des données.....	4
1.2.5	Validation serveur et retours utilisateurs clairs	4
1.2.6	Système de gestion des thés	4
1.2.7	Listes et exportation PDF.....	5
1.3	Planification	5
1.3.1	Planification initiale	6
1.3.2	Méthodologie de travail.....	7
2	Analyse / Conception.....	8
2.1	Maquettes.....	8
2.2	Modélisation de la base de données.....	13
2.2.1	MCD	13
2.2.2	MLD	13
2.2.3	MPD.....	14
2.3	Stratégie de test	14
2.3.1	Tests fonctionnels.....	14
2.3.2	Tests unitaires	15
2.3.3	Données de test	16
2.4	Risques techniques.....	16
2.4.1	Apprentissage sur bibliothèques externes.....	16
2.4.2	Expérience limitée en JavaScript.....	16
2.4.3	Familiarité restreinte avec Laravel.....	16
2.5	Dossier de conception	16
2.5.1	Environnement.....	16
2.5.2	Outils.....	16
3	Réalisation.....	17
3.1	Problèmes rencontrés	17
3.1.1.	Problèmes de	17
3.2	Fonctionnalités importantes.....	17
3.2.1	Barre de Recherche, filtres et tri	17
3.2.2	Connexion/authentification.....	23
3.2.3	Listes déroulantes de la page d'ajout.....	23
3.2.4	Exportation en PDF.....	27
1.2.1.	Description physique du projet.....	31
1.2.1.1.	Répertoires où le logiciel est installé :	31
1.2.1.2.	Liste des fichiers et description :	31
3.1.	Description des tests effectués	31
1.2.2.	Test 1.....	31
1.3.	Erreur restante.....	32

1.4.	Liste des documents fournis	32
2.	Conclusions	32
2.2.	Objectifs et Résultats	32
2.3.	Suites possibles pour le projet (évolutions & améliorations)	32
2.3.1.	Suite 1.....	32
3.	Annexes.....	33
3.2.	Aides externes	33
3.3.	Résumé du rapport du TPI / version succincte de la documentation	33
3.4.	Sources – Bibliographie.....	33
3.5.	Glossaire	34
3.6.	Table des illustrations	35
1.	Journaux de travail.....	35
2.	Manuel d'Installation.....	35
5.4.1.	Manuel 1.....	35
5.4.1.1.	Introduction.....	36
5.4.1.2.	Prérequis	36
5.4.1.3.	Explication des commandes	36
5.4.2.	Résolution des problèmes.....	36
5.4.2.1.	Erreur de dépendances manquantes	36
5.4.2.2.	Vérification de l'installation.....	36
5.4.3.	Conclusion	36
3.	Manuel d'Utilisation.....	36

1 Analyse préliminaire

Dans cette partie du rapport nous verrons le cadre ainsi que le but du projet.

1.1 Introduction

Ce projet s'inscrit dans le cadre du travail pratique individuel (**TPI**) réalisé en fin de formation d'informaticien à l'ETML. Il répond à la demande d'ExceptionTea SA, une entreprise lausannoise spécialisée dans les thés rares, qui souhaite disposer d'une solution personnalisée pour référencer, organiser et gérer sa collection.

L'application web sera développée avec le framework PHP Laravel, en suivant une architecture MVC rigoureuse. Elle proposera une interface moderne, responsive et intuitive, conçue avec HTML5, CSS3 et Tailwind CSS.

1.2 Objectifs

1.2.1 **Respect du modèle MVC avec Laravel**

Mettre en œuvre rigoureusement l'architecture Modèle-Vue-Contrôleur à chaque étape du développement, en séparant clairement la logique métier (modèles), la gestion des requêtes et règles (contrôleurs) et la présentation (vues Blade).

1.2.2 **Qualité et lisibilité du code**

Produire un code clair et bien structuré, en respectant les conventions officielles Laravel, ainsi que les standards HTML5 et Tailwind CSS pour le front-end. Les noms de classes, méthodes et fichiers doivent être explicites et cohérents.

1.2.3 **Processus d'inscription et d'authentification complet**

Implémenter l'ensemble des fonctionnalités utilisateurs : création de compte, connexion, réinitialisation de mot de passe. L'ensemble doit être utilisable sans bug, avec des redirections et messages d'état appropriés.

1.2.4 **Sécurité des accès et des données**

Appliquer les normes de sécurité Laravel : hachage des mots de passe, protection CSRF, gestion des autorisations via middleware et policies, et chiffrement des sessions. Les rôles et permissions doivent garantir que seuls les utilisateurs authentifiés accèdent aux pages protégées.

1.2.5 **Validation serveur et retours utilisateurs clairs**

Mettre en place des règles de validation sur tous les formulaires côté serveur. En cas d'erreur, l'utilisateur reçoit un message précis indiquant le champ concerné et la nature du problème.

1.2.6 **Système de gestion des thés**

Une interface complète de gestion des produits (CRUD) sera intégrée à l'application. Les utilisateurs pourront créer, modifier, supprimer ou consulter les informations liées aux thés (nom, variété, type, provenance, prix, etc.).

1.2.7 Listes et exportation PDF

Les utilisateurs auront la possibilité de créer des listes de thés, en fonction de leurs préférences. Une fonctionnalité d'export au format PDF permettra de générer et sauvegarder ces listes de manière pratique.

1.3 Planification

Ce projet a débuté le mercredi 30 avril 2025 à 8h00 et se terminera le lundi 26 mai 2025 à 11h25. Pour planifier ses différentes étapes, j'ai utilisé un modèle Excel fourni par l'ETML et conçu par Monsieur Lymberis.

Ce fichier, tout comme mon journal de travail, se présente sous la forme d'un tableau. Dans la première colonne, on peut sélectionner la catégorie de la tâche à réaliser, puis indiquer dans la suivante le nombre de périodes de 5 minutes que la tâche a nécessité. La colonne "Explications" sert à décrire plus précisément ce qu'il est prévu de faire, et enfin une dernière colonne permet d'ajouter des liens de référence utiles.

Voici un extrait illustrant une séquence planifiée :

Séquence	7			Date: lundi, 5 mai 2025	Après - midi
Tâche	Tranche [5min]	Explications: qu'est-ce qui se fait et comment ?			Liens, références, ...
Conception - MLD MCD	13	1h 5 min	Conception des modèles de base de données MLD et MCD		
Conception - Tests	13	1h 5 min	Conception des tests qui seront effectués à la fin du projet pour valider les fonctionnalités		
Doc - Journal de travail	2	10 min	Rédaction des tâches effectuées durant la journée.		
Total tranche	28	2h 20 min			

Figure 1 - Exemple - Séquence de planification

Une fois toutes les tâches planifiées (comme ci-dessus), le fichier permet de générer un diagramme de Gantt qui illustre la planification et le suivi de l'avancement. Chaque barre verte représente la période prévue pour une tâche : on voit ainsi d'un coup d'œil ce qui était programmé et à quel moment du projet. Les barres bleues, quant à elles, proviennent du journal de travail et indiquent le temps réellement investi : elles permettent de comparer précisément la planification initiale à la réalité du terrain.

Voici le début de ce diagramme :

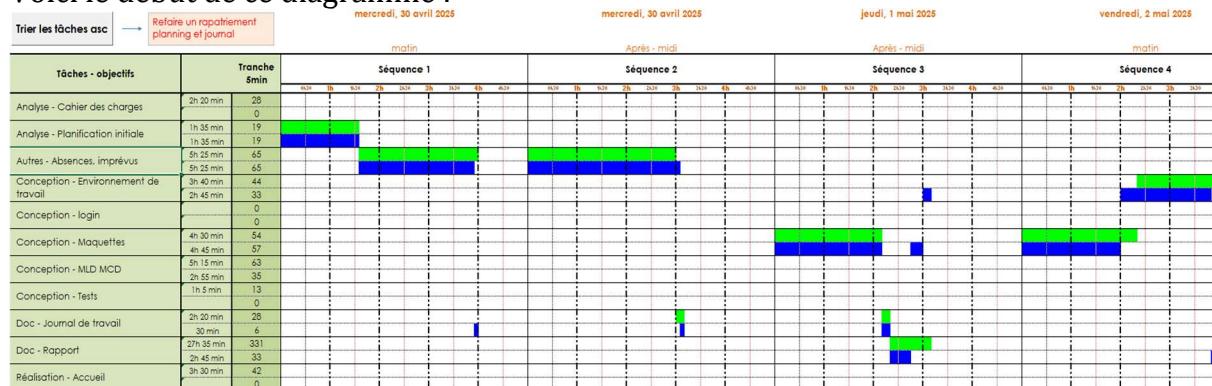


Figure 2 Exemple - Diagramme Gantt :

Révision de la planification initiale du projet :

- *planning indiquant les dates de début et de fin du projet ainsi que le découpage connu des diverses phases.*
- *partage des tâches en cas de travail à plusieurs.*

Il s'agit en principe de la planification définitive du projet. Elle peut ensuite être affinée (découpage des tâches). Si les délais doivent être ensuite modifiés, le responsable de projet doit être avisé, et les raisons doivent être expliquées dans l'historique.

1.3.1 Planification initiale

Premièrement voici à quoi ressemble ma planification initiale sous forme de graphique :

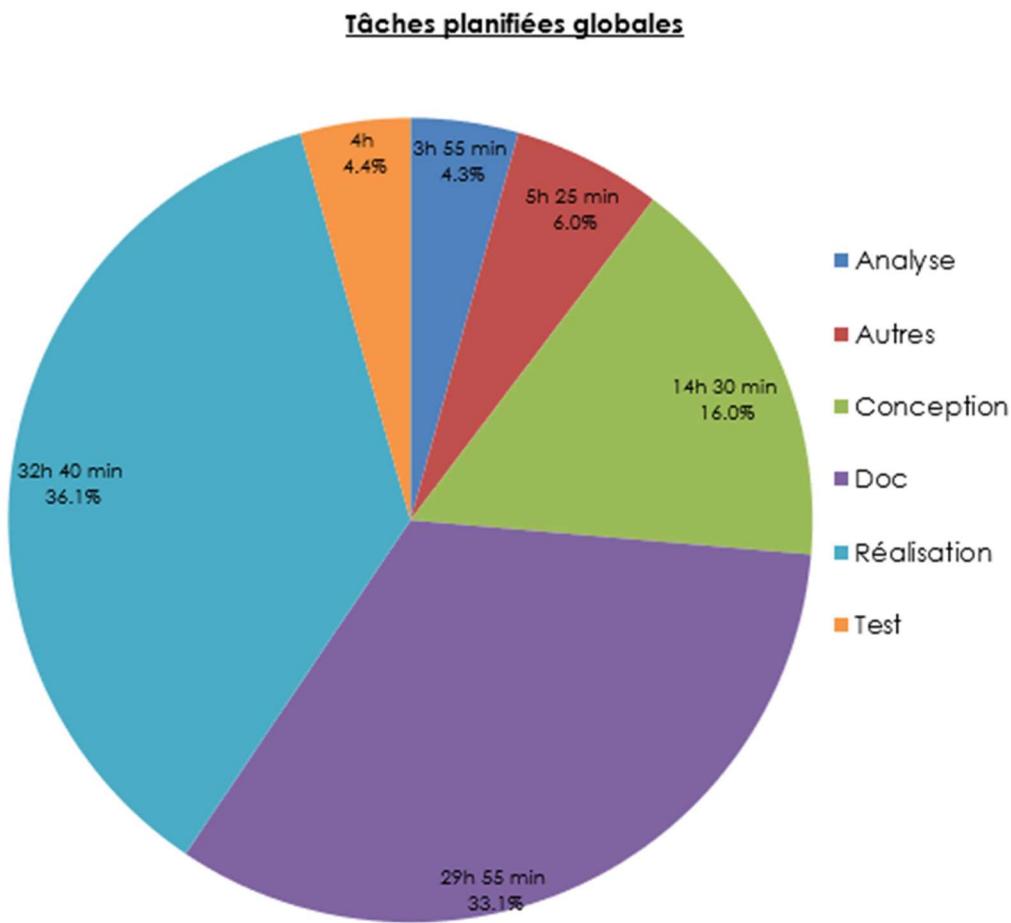


Figure 3 - Graphique des tâches planifiées

On observe tout d'abord qu'avec près de 33 % du temps consacré à la documentation, celle-ci occupe une place majeure dans le projet, soulignant l'importance accordée à la traçabilité et à la qualité du livrable écrit.

L'analyse (4,3 %) et la conception (16 %) totalisent environ 20 %, exactement ce qui était préconisé dans le cahier des charges pour la phase d'étude.

La réalisation atteint 36 %, très proche des 35 % réservés à l'implémentation, ce qui confirme un bon respect de la répartition initiale.

Dans la catégorie "autres" nous pourrons retrouver notamment les absences et imprévus potentiels, ces deux choses n'étant pas planifiables ces 6 % représente une légère marge de sécurité, du temps prévu en plus dans le cas où une fonctionnalité ne seraient pas terminé ou que la documentation devrait être finalisée.

Enfin, les tests représentent 4,4 % du planning, ce temps dédié à la validation des fonctionnalités garantit la fiabilité du logiciel.

1.3.2 Méthodologie de travail

Pour ce projet, mon choix de méthodologie s'est tourné vers la méthode des six pas, celle-ci répond particulièrement bien aux exigences d'un TPI sur le développement web. Elle permet aussi une approche structurée avec une phase d'analyse approfondie qui permet de bien cerner les points importants et les objectifs du travail. La phase suivante se concrétise par la planification claire des tâches, suivie d'une conception méthodique des solutions. Le développement lui se fait de manière itérative, ce qui donne l'avantage de la flexibilité du projet en cours de route. Enfin, des phases de contrôle qualité rigoureuses assurent la fiabilité du produit final. Contrairement au modèle en cascade, cette méthode permet une progression plus souple, chaque étape pouvant évoluer indépendamment. Les dernières phases garantissent une évaluation complète avant la livraison, identifiant les points d'amélioration pour les projets futurs.

1.3.2.1 Décomposition en tâches selon les six pas

1. S'informer

- Lecture et compréhension du cahier des charges pour cadrer l'application ExceptionTea.
- Entretiens avec l'expert ou l'enseignant pour valider les objectifs.
- Lecture d'informations sur Laravel, MySQL et Tailwind CSS.

2. Planifier

- Découpage du projet en jalons (maquettes, prototype, développement, tests, documentation).
- Estimation des durées pour chaque phase et chaque tâche.
- Mise en place du dépôt GitHub et configuration du suivi de version.

3. Décider

- Validation des wireframes basse fidélité pour chaque page clé.
- Choix des librairies front-end (Tailwind CSS, plugin Vite) et des outils de génération de PDF (Dompdf ou Snappy).
- Élaboration du MCD, du MLD et du MPD pour modéliser la structure de la base de données.

- Conception des maquettes des différentes pages.

4. Réaliser

- Développement back-end : création des modèles, migrations, contrôleurs et routes.
- Développement front-end : intégration des maquettes en Blade, application des styles Tailwind et interactivité (pop-ups, filtres).
- Configuration de la base de données MySQL et exécution des migrations Laravel.
- Rédaction de la documentation technique et du journal de bord.

5. Contrôler

- Exécution des tests unitaires et fonctionnels (vérification des flux CRUD, sécurité, validation des formulaires).
- Revue de code et corrections éventuelles (refactoring au normes, ajustement des performances).
- Préparation de la démonstration finale et génération des livrables.

6. Evaluer

- Recueil des retours venant de l'évaluation du chef de projet et des experts.
- Analyse des écarts entre le cahier des charges et le résultat livré.
- Identification des améliorations et des bonnes pratiques à retenir pour un futur projet.

2 Analyse / Conception

2.1 Maquettes

J'ai choisi de réaliser des maquettes plus haute-fidélités au lieu de simples wireframes, afin de faciliter la phase frontend et d'éviter toute hésitation lors de l'intégration.

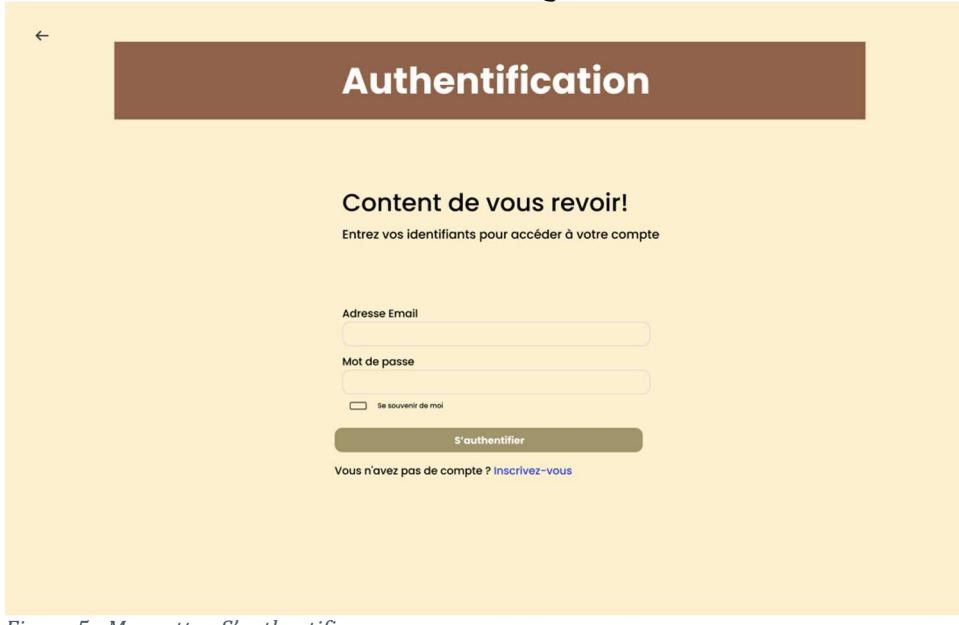
Les voici :



La maquette de la page d'enregistrement est une interface web avec un fond beige. En haut à gauche, il y a un bouton de retour (**<**). Le titre principal est "S'enregistrer" dans un grand bouton marron. En dessous, le sous-titre "Commencez dès maintenant" est affiché en noir. Il y a trois champs textes pour les informations personnelles : "Nom", "Adresse Email" et "Mot de passe". En bas de la page, il y a un bouton "S'enregistrer" et un lien bleu "Vous avez déjà un compte ? S'authentifier".

Figure 4 - Maquette - S'enregistrer

Cette page permet de s'enregistrer la première fois, lorsqu'on n'a pas encore de compte. Pour ce faire, il faut entrer son nom, son adresse e-mail et un mot de passe. Puis la validation se fait avec le bouton "S'enregistrer".



La maquette de la page d'authentification est une interface web avec un fond beige. En haut à gauche, il y a un bouton de retour (**<**). Le titre principal est "Authentification" dans un grand bouton marron. En dessous, le message "Content de vous revoir!" est affiché en noir, suivi du sous-titre "Entrez vos identifiants pour accéder à votre compte". Il y a deux champs textes pour les identifiants : "Adresse Email" et "Mot de passe". Il y a également une case à cocher "Se souvenir de moi". En bas de la page, il y a un bouton "S'authentifier" et un lien bleu "Vous n'avez pas de compte ? Inscrivez-vous".

Figure 5 - Maquette - S'authentifier

Ici nous avons la page qui permet de se connecter si on a déjà un compte, il suffit de rentrer son adresse mail et son mot de passe dans les champs prévus à cet effet. Puis la validation se fait avec le bouton "S'authentifier".

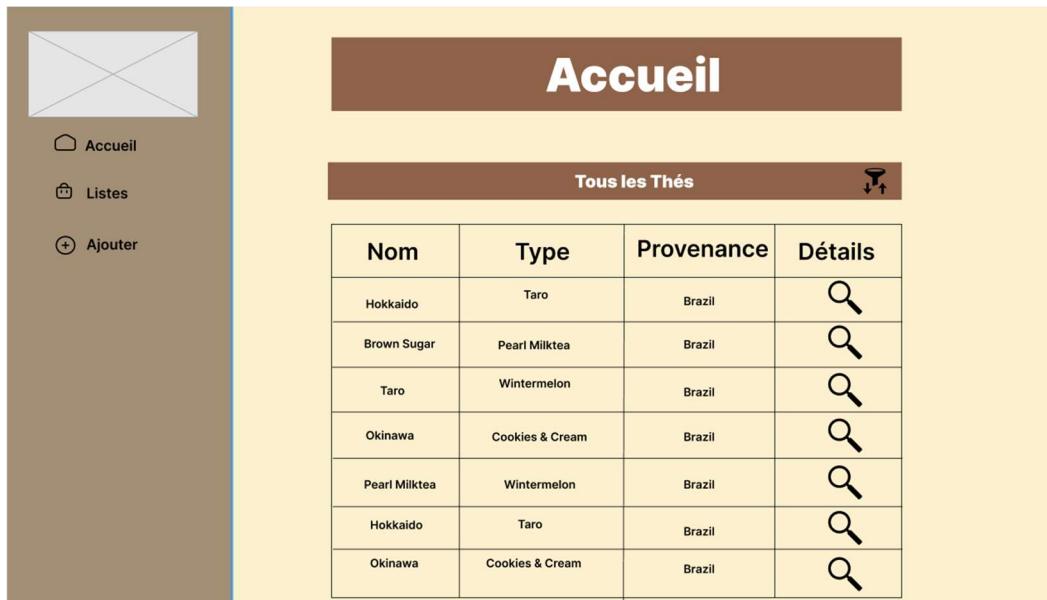


Figure 6 - Maquette - Accueil

Voici la page d'accueil. On y trouve une liste de thés disponibles dans la boutique, avec des options de tri et de filtrage. Il est possible d'accéder à la page de détails d'un thé en cliquant sur l'icône en forme de loupe située à droite de chaque élément. À gauche, des boutons de navigation permettent de parcourir les différentes pages du site.

Nom du Thé

Description
Lorem Ipsum....

Préparation
Lorem Ipsum....

Quantité : XX

Type : XXX

Variété : XXX

Provenance : XXX

Supprimer

Modifier

Figure 7 - Maquette - Détails du Thé

Ici, la page qui permet d'obtenir plus d'informations sur le thé sélectionné. On peut y consulter des éléments tels que la description, les instructions de préparation, la quantité en stock, la variété et la provenance du thé. Deux boutons situés dans le pied de page permettent de modifier ou de supprimer les informations liées à ce thé.

La maquette de formulaire pour ajouter un thé est divisée en plusieurs sections :

- Section de base :** Nom (champ avec placeholder "e.g. Apple, Banana, etc."), Préparation (champ).
- Section détaillée :** Description (champ), Quantité (champ).
- Section financière :** Prix (champ).
- Section temporelle :** Date (champ avec icône de calendrier).
- Section variétés :** Variété (liste déroulante).
- Section provenance :** Provenance (liste déroulante).
- Section types :** Type (liste déroulante avec options Type A, Type B et Type C). Chaque option a une icône de crayon et une croix à droite.

En bas du formulaire, il y a deux boutons : "Effacer" et "Ajouter".

Figure 8 - Maquette - Ajouter un thé

Sur cette page, nous avons un formulaire permettant d'ajouter un nouveau thé dans la base de données. Nous devons renseigner toutes ces informations importantes dans des champs prévus à cet effet. Les cinq premiers champs sont essentiellement des champs texte. Le champ de la date aura une icône de calendrier à droite et sera de type "date", ce qui facilite la sélection. Les trois dernières informations à renseigner sont la variété, la provenance et le type. Ces trois champs sont sous forme de listes déroulantes, il suffit de sélectionner l'information correcte dans la liste pour l'assigner au nouveau thé en question. Ces listes déroulantes permettent également de supprimer des options grâce à la croix à droite de l'élément, de modifier des éléments avec l'icône de crayon et d'ajouter un élément avec le bouton "Ajouter un élément" en fin de liste.

←

Listes

Nom de la liste	Date	Nombres	Actions
Kkopi Original	03/04/2023	32	
Caramel Machiatto	03/04/2023	32	
Vanilla	03/04/2023	32	
Chocolate	03/04/2023	32	
Kkopi Original	03/04/2023	32	
Caramel Machiatto	03/04/2023	32	
Vanilla	03/04/2023	32	
Chocolate	03/04/2023	32	
Kkopi Original	03/04/2023	32	
Caramel Machiatto	03/04/2023	32	
Vanilla	03/04/2023	32	
Chocolate	03/04/2023	32	

Ajouter une liste

Figure 9 - Maquette – Listes

Ci-dessus, la page contenant toutes les listes de thés disponibles, répertoriées par leur nom, la date de création et le nombre de thés présents dans chaque liste. Dans la colonne "Actions", nous avons trois boutons :

1. La loupe qui affiche les détails de la liste,
2. Le stylo qui modifie les éléments de la liste,
3. La poubelle qui supprime entièrement la liste.

←

Nom de la liste	Date
-----------------	------

XX Thés

Cookies & Cream	Kkopi Barako
Pearl Milktea	Hazelnut Latte
Wintermelon	Caramel Latte
Taro	Vanilla Latte
Original	Kkopi Latte
Vanilla	Kkopi Barako
Caramel	Hazelnut Latte
Hazelnut	Caramel Latte
Kkopi Barako	Vanilla Latte
Hazelnut Latte	Kkopi Latte
Caramel Latte	
Vanilla Latte	
Kkopi Latte	

Exporter en PDF

Figure 10 - Maquette - Détails d'une liste

Voici la page de détails d'une liste, elle permet à l'utilisateur de voir le contenu de toute la liste ainsi que de l'exporter en PDF avec le bouton vert.

2.2 Modélisation de la base de données

2.2.1 MCD

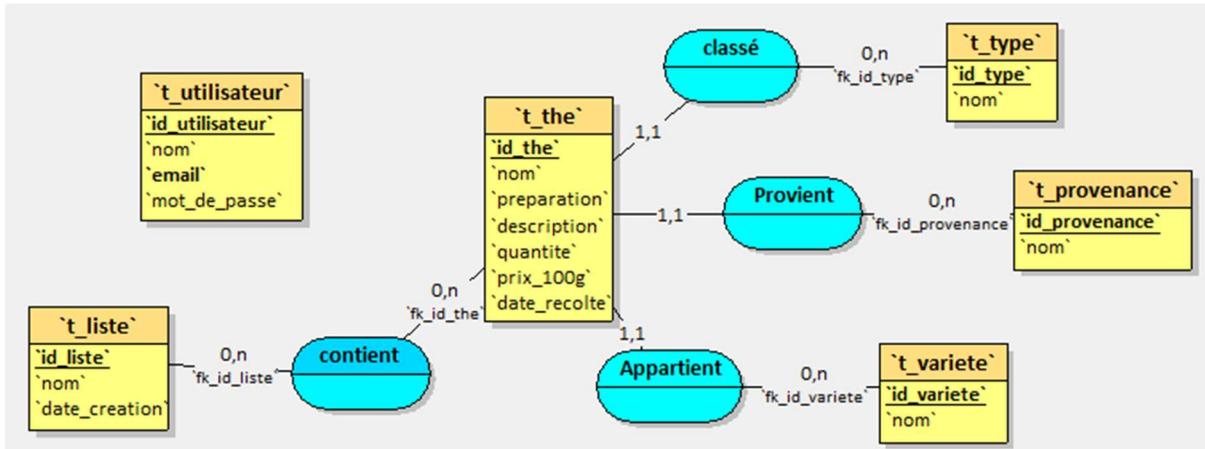


Figure 11 - BDD - MCD

Le MCD présente, de manière visuelle et abstraite, les entités clés du domaine, ici Thé, Provenance, Type, Variété, Utilisateur et Liste en plus de leurs relations. Chaque entité est représentée par un rectangle listant ses principales propriétés (nom, description, etc.), et chaque association par un ovale verbal (par exemple "contient" pour relier Liste et Thé, "Provient" pour relier Thé et Provenance). Ce schéma garantit que tous les éléments métier et leurs interactions sont correctement identifiés avant toute implémentation technique.

2.2.2 MLD

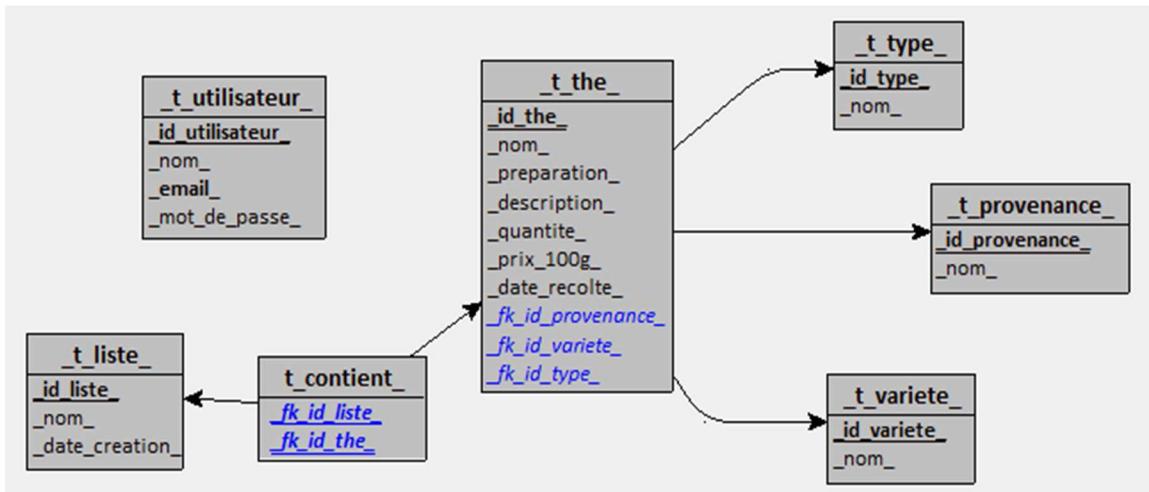


Figure 12 - BDD - MLD

Le MLD correspond à une version technique du MCD, mieux adaptée aux contraintes d'un système de gestion de base de données relationnelle comme MySQL. Il formalise chaque entité du MCD en une table concrète avec des noms et une clé primaire. Les associations deviennent des liens explicites via des clés étrangères (`fk_...`), intégrées aux tables concernées. La relation plusieurs(0,n)-à-plusieurs(0,n) entre les listes et les thés est gérée par une table d'association appelée **t_contient**, dont la **clé primaire composite** (`fk_id_liste`, `fk_id_the`) permet d'éviter les doublons dans les liens.

2.2.3 MPD

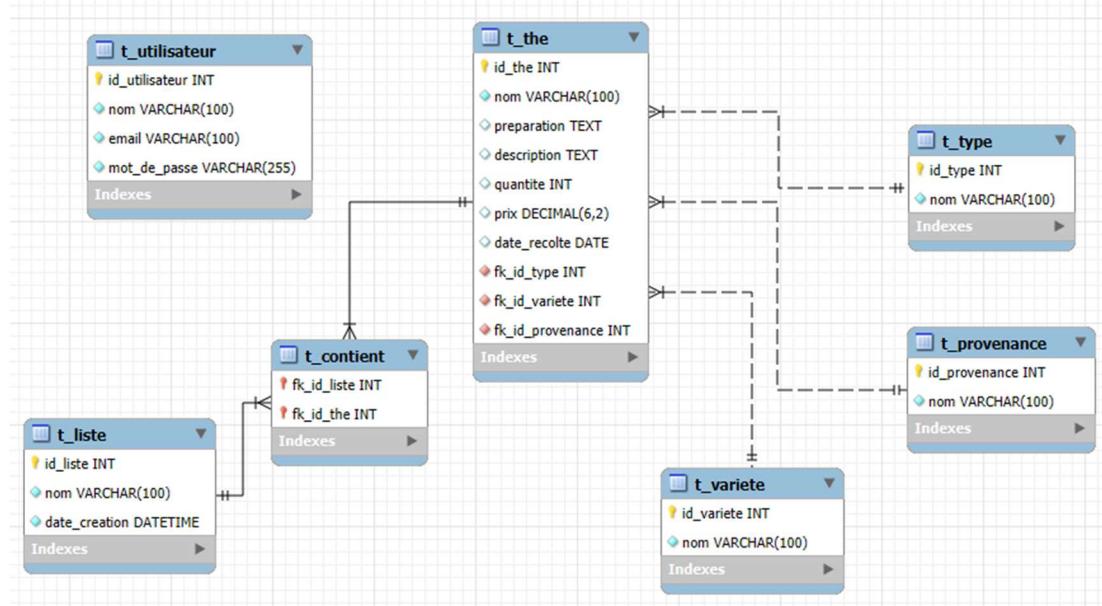


Figure 13 - BDD – MPD

Le MPD correspond à la représentation concrète de la base de données directement issue de l'outil Workbench de MySQL. Il affiche chaque table avec ses colonnes, types de données, clés primaires et clés étrangères, ainsi que les connexions schématisées entre elles. Ce schéma reflète exactement ce que le SGBD met en œuvre, prêt à être en production.

2.3 Stratégie de test

La stratégie de tests vise à garantir que l'application répond précisément aux exigences fonctionnelles et non fonctionnelles définies dans le cahier des charges, tout en assurant la robustesse et la maintenabilité du code. Elle se déploie selon deux axes :

1. Tests fonctionnels (Feature Tests)
2. Tests unitaires

2.3.1 Tests fonctionnels

Objectif : vérifier que chaque fonctionnalité clé du cahier des charges fonctionne dans son ensemble.

Exemples de scénarios :

1. Inscription / connexion : un nouvel utilisateur peut créer un compte et se connecter.
2. CRUD sur les thés : ajout, modification, consultation et suppression d'un thé.
3. Listes de thés : création d'une liste, ajout et retrait de thés, export PDF.

Exécution : le but est de rédiger d'abord une fiche simple par scénario (étapes et résultat attendu).

Une fois la fonctionnalité implémentée, on reprend cette fiche pour valider les tests, à condition qu'ils respectent entièrement le scénario décrit.

Voici un tableau des tests fonctionnels :

N° du test	Fonctionnalité	Action	Résultat attendu
1	Inscription / authentification	Tenter de s'inscrire avec un nouvel e-mail valide puis se déconnecter et se reconnecter	L'inscription crée un compte, la connexion fonctionne et l'utilisateur est redirigé vers le dashboard
2	Sécurité utilisateur (mot de passe, autorisations)	Tenter d'accéder à /the/create sans être connecté	L'accès est refusé
3	Validation serveur des formulaires	Soumettre le formulaire d'ajout de thé avec un champ « prix » vide et un nom trop court	Le formulaire est rejeté, et des messages d'erreur clairs s'affichent sous chaque champ concerné
4	Gestion du stock de produits (CRUD)	Créer un nouveau thé, puis modifier sa quantité, le consulter et enfin le supprimer	Après chaque action, les données sont correctement enregistrées / affichées / supprimées dans la base MySQL
5	Listes personnalisées + export PDF	Créer une liste de thés, y ajouter plusieurs thés, puis cliquer sur « Exporter en PDF »	Un fichier PDF est téléchargé, contenant la liste des thés avec leur nom.

Le respect du modèle MVC ainsi que la qualité du code (conventions Laravel, Tailwind CSS, HTML5) ne faisant pas partie des tests fonctionnels, ils ne figurent donc pas dans le tableau ci-dessus.

2.3.2 Tests unitaires

Objectif : vérifier que les fonctions essentielles du code donnent les bons résultats, sans dépendre de l'interface ou de la base de données.

Exemples de cibles :

1. Methods de validation des modèles (par exemple, règles de quantité et de prix).
2. Calculs éventuels (par exemple, conversion de dates ou formats).

Outil : Unit Tests de Laravel (php artisan make:test ... --unit).

Exécution : peut se faire avec la commande : php artisan test --filter Unit

2.3.3 **Données de test**

Les tests ont été effectués avec des données simulées, pas de données réelles pour des raisons de sécurité.

2.4 **Risques techniques**

Plusieurs points de vigilance ont été identifiés :

2.4.1 **Apprentissage sur bibliothèques externes**

L'intégration de Dompdf par exemple ou d'un outil équivalent pour l'export en PDF peut demander un peu de temps de prise en main supplémentaire. Pour réduire cet impact, je consulterai rapidement une documentation officielle ou des vidéos sur le sujet afin de maîtriser les fonctionnalités essentielles sans retarder mon planning.

2.4.2 **Expérience limitée en JavaScript**

Le projet requiert des interactions dynamiques (pop-ups, filtres, ...). Comme JavaScript n'a pas été vu en cours, il est possible que certaines mises en œuvre prennent plus de temps que prévu. Pour y faire face, j'envisage de réutiliser des exemples de scripts simples et de privilégier **des snippets éprouvés**, plutôt que d'apprendre un framework au complet.

2.4.3 **Familiarité restreinte avec Laravel**

Bien que Laravel soit réputé pour son intuitivité, je ne l'ai utilisé qu'au cours d'un projet pré-TPI. Certaines conventions ou fonctionnalités avancées pourraient nécessiter une phase de repérage. Pour limiter ce risque, je m'appuierai sur la documentation officielle et les exemples issus de mon projet de pré-TPI.

2.5 **Dossier de conception**

Le dossier de conception décrit l'environnement de travail mis en place pour le projet, ainsi que les outils principaux utilisés. Il sert à poser le cadre technique avant de coder réellement.

2.5.1 **Environnement**

Ce projet se réalise intégralement en local, sur un poste de travail standard (Windows 10 ou équivalent) disposant d'une configuration matérielle suffisante pour exécuter un **IDE** et un navigateur moderne sans latence. Aucune infrastructure serveur externe n'est requise durant la phase de développement : tout tourne directement en local.

2.5.2 **Outils**

Voici les outils utilisés durant ce projet ainsi que leur version :

<u>Outils</u>	<u>Versions</u>
PHP	8.3.4
Laravel	12.12.0
TailwindCSS	4.1.5
MySQL	8.4.5
MySQL Workbench	8.0.42
Looping	4.1
Composer	2.8.8
NodeJS	22.15.0
NPM	10.9.2
Visual Studio Code	1.99.3
GitHub Desktop	3.4.19
Vite	6.3.4
Breeze	2.3.6
Blade	12.12.0
Pest	3.8.2

3 Réalisation

3.1 Problèmes rencontrés

3.1.1. Problèmes de ...

3.1.1.1. Problème

3.1.1.2. Cause

3.1.1.3. Solution

3.2 Fonctionnalités importantes

3.2.1 Barre de Recherche, filtres et tri

3.2.1.1. Visuel

3.2.1.1.1. Barre de recherche

Sur la page d'accueil (listing de tous les thés), une barre de recherche textuelle a été ajoutée pour faciliter la navigation lorsque la liste devient longue. Elle se comporte en filtrant en temps réel la liste des thés dont le nom contient la chaîne tapée (recherche insensible à la casse). Voici son visuel :

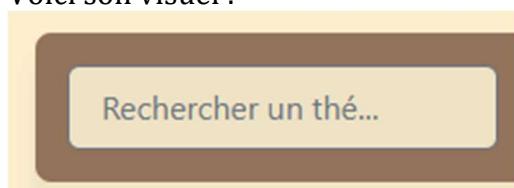


Figure 14 - Barre de recherche

Par exemple si on cherche "ear" il nous filtre la liste et n'affiche que le thé dont le nom commence par ces trois lettres :

NOM ↗	TYPE ↗	PROVENANCE ↗
Earl Grey	Thé noir	Inde

Figure 15 - affichage - recherche

Si nous avions d'autres thé commençant par les mêmes caractères ils apparaîtraient aussi.

3.2.1.1.2 Filtrage

Il y a deux filtres disponibles, le filtrage par type de thé et celui par leur provenance. Voici à quoi ressemble les deux listes.

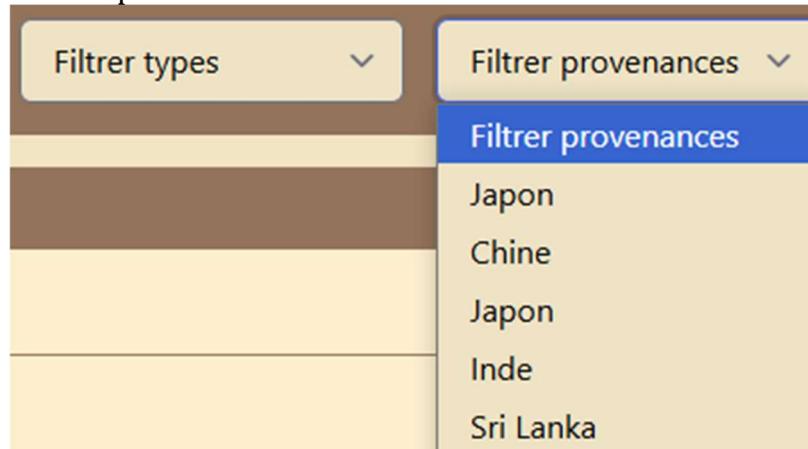


Figure 16 - visuel - filtrage

Pour sélectionner le type ou la provenance que l'on souhaite nous devons cliquer sur l'icône de flèche à droite et une liste déroulante s'ouvre (comme sur le filtre de droite).

Nous pouvons choisir de cumuler deux filtres puis seuls les éléments qui contiennent les deux critères apparaissent. Comme dans cet exemple :

NOM ↗	TYPE ↗	PROVENANCE ↗
Gyokuro Uji	Thé vert	Chine
Assam	Thé vert	Chine

Figure 17 - exemple - filtrage

3.2.1.1.3 Tri

Chaque en-tête de colonne (Nom, Type, Provenance) affiche une double flèche « ↑ »

Un premier clic trie la colonne par ordre alphabétique.

NOM ↑
Assam
Ceylan Orange Pekoe
Darjeeling First Flush
Earl Grey
Gyokuro Uji
Pu-erh Vintage
Silver Needle
Tieguanyin

Figure 18 - exemple - tri

Un second clic sur cette double flèche inverse l'ordre.

3.2.1.2 Fonctionnement

À présent nous allons voir comment se traduit ces fonctionnalités dans le code :

3.2.1.2.1 Tri

La partie code du tri s'effectue de la ligne 90 à la 123 avec la fonction "sortTable".

```

90  /**
91   * Fonction de tri du tableau
92   * @param {string} column - La colonne sur laquelle effectuer le tri
93   */
94  function sortTable(column) {
95      const index = {
96          'nom': 0,
97          'type': 1,
98          'provenance': 2
99      }[column];
100
101     sortDirection[column] = !sortDirection[column];
102
103     const tbody = table.querySelector('tbody');
104     const sortedRows = rows.sort((a, b) => {
105         const aValue = a.children[index].textContent.trim().toLowerCase();
106         const bValue = b.children[index].textContent.trim().toLowerCase();
107         return sortDirection[column] ?
108             aValue.localeCompare(bValue) :
109             bValue.localeCompare(aValue);
110     });
111
112     // Mise à jour du tableau avec les lignes triées
113     tbody.innerHTML = '';
114     sortedRows.forEach(row => tbody.appendChild(row));
115 }
116
117 // Ajout des écouteurs d'événements pour le tri
118 table.querySelectorAll('th[data-sort]').forEach(th => {
119     th.addEventListener('click', () => {
120         const column = th.dataset.sort;
121         sortTable(column);
122     });
123 });

```

Figure 19 - code - fonction – sortTable

Lignes 95 à 99 on détermine quel index de cellule trier selon la colonne.

Ligne 101 : on bascule le sens du tri (asc/desc).

Lignes 103–110 : on trie le tableau de lignes en mémoire, puis on réinjecte l'ordre dans le <tbody>.

Lignes 118–121 : on associe la fonction aux clics sur les en-têtes marqués "data-sort".

3.2.1.2.2 Filtrage et recherche

Dans notre fichier de vue dashboard.php nous avons d'abord le code html qui permet de

faire apparaître ces options de filtrage des thés :

```

9   <!-- Section des filtres de recherche -->
10  <div class="bg-[#967259] p-4 rounded-lg">
11    <div class="flex flex-wrap gap-4">
12
13      <!-- Barre de recherche textuelle -->
14      <input type="text" id="searchInput" placeholder="Rechercher un thé..." 
15          class="px-4 py-2 rounded-md focus:outline-none focus:ring-2 focus:ring-[#7d6049] bg
16
17      <!-- Filtre par type de thé -->
18      <select id="typeFilter" class="w-full sm:w-48 px-4 py-2 rounded-md focus:outline-none f
19          <option value="">Filtrer types</option>
20          @foreach($types as $type)
21              <option value="{{ $type->nom }}>{{ $type->nom }}</option>
22          @endforeach
23      </select>
24
25      <!-- Filtre par provenance -->
26      <select id="provenanceFilter" class="w-full sm:w-48 px-4 py-2 rounded-md focus:outline-
27          <option value="">Filtrer provenances</option>
28          @foreach($provenances as $provenance)
29              <option value="{{ $provenance->nom }}>{{ $provenance->nom }}</option>
30          @endforeach
31      </select>
32
33  </div>
34
35 </div>

```

Figure 20 - code – section filtrage de recherche - HTML

Nous pouvons voir que la barre recherche est un simple input de type texte avec un Id de la ligne 13 à la 15.

Les lignes 17 à 23 sont celles pour la liste déroulante servant à choisir le type de thé à filtrer, avec une boucle foreach qui affiche tous les types disponibles.

Pour la provenance au lignes 25 à 31 c'est sensiblement la même chose que le filtre de type.

Voilà pour la partie HTML, maintenant regardons comment ces trois filtrages sont codés.

En suivant l'id "searchInput" dans le code, nous arrivons à la partie technique où le script et les méthodes sont déclarés. Ce script qui gère toute la partie tri et filtre :

```

77  @push('scripts')
78  <script>
79  /* Script pour la gestion du tableau interactif */
80  document.addEventListener('DOMContentLoaded', function() {
81      // Initialisation des éléments DOM
82      const table = document.getElementById('teasTable');
83      const searchInput = document.getElementById('searchInput');
84      const rows = Array.from(table.querySelectorAll('tbody tr'));

```

Figure 21 - code - début du script

Ici, la ligne 80 s'assure que le script ne s'exécute qu'après le chargement complet de l'HTML.

La ligne 82 récupère le tableau "teasTable" qui contient la liste de tous les thés disponibles, cet élément sera ensuite filtré.

La ligne 83 sélectionne le champ de recherche textuelle ; c'est lui qui capte la saisie de

l'utilisateur pour filtrer le tableau.

Ligne 84 transforme la liste des <tr> du <tbody> en tableau JavaScript, pour pouvoir itérer facilement sur chaque ligne.

Viens notre première fonction :

```

125  /**
126   * Fonction de filtrage du tableau
127   * Applique les filtres de recherche, type et provenance
128   */
129  function filterTable() {
130      const searchTerm = searchInput.value.toLowerCase();
131      const selectedType = typeFilter.value.toLowerCase();
132      const selectedProvenance = provenanceFilter.value.toLowerCase();
133
134      rows.forEach(row => {
135          const name = row.children[0].textContent.toLowerCase();
136          const type = row.children[1].textContent.toLowerCase();
137          const provenance = row.children[2].textContent.toLowerCase();
138
139          // Application des différents filtres
140          const matchesSearch = name.includes(searchTerm);
141          const matchesType = !selectedType || type === selectedType;
142          const matchesProvenance = !selectedProvenance || provenance === selectedProvenance;
143
144          // Affichage/masquage des lignes selon les filtres
145          row.style.display = (matchesSearch && matchesType && matchesProvenance) ? '' : 'none';
146      });
147 }

```

Figure 22 - code - fonction - filterTable

Ici nous avons donc la fonction filterTable qui applique tous les filtres en même temps à notre liste de thés.

La ligne 130 récupère le texte entré dans la barre de recherche et le converti en minuscule pour le rendre insensible à la casse.

Puis les deux lignes en dessous récupèrent la valeur du filtre type et du filtre provenance.

Le forEach de la ligne 134 à 146 parcours, dans un premier temps (135-137), chaque ligne en extrayant le texte de chaque cellules (en LowerCase pour que la recherche soit insensible à la casse). Puis il teste les trois conditions sur les valeurs extraites aux lignes 140 à 142.

Finalement la ligne 145 affiche la ligne dans le tableau si celle-ci respecte bien tous les critères.

Il nous reste à attacher les filtres à un Eventlistener (écoute les évènements) pour qu'à chaque modification de texte dans la barre de recherche ou à chaque modification d'un filtre la fonction "filterTable" soit réappelée. Comme ceci :

```

// Ajout des écouteurs d'événements pour le filtrage
searchInput.addEventListener('input', filterTable);
typeFilter.addEventListener('change', filterTable);
provenanceFilter.addEventListener('change', filterTable);

```

Figure 23 - code - EventListener

3.2.2 Connexion/authentification

Pour l'inscription et la connexion, pas besoin de tout coder à la main : c'est Laravel Breeze qui s'en charge.

Ce plugin officiel contient par défaut :

- Les routes (/login, /register, /forgot-password, etc.)
- Les contrôleurs pour gérer l'authentification et la réinitialisation de mot de passe
- Les vues Blade, déjà stylées avec Tailwind CSS
- Le middleware auth et guest pour protéger les pages

Il suffit de lancer php artisan breeze:install, puis npm install && npm run dev pour avoir un système d'authentification complet, sécurisé et prêt à l'emploi.

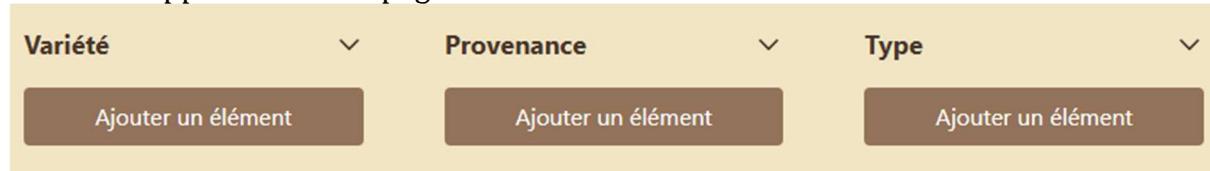
3.2.3 Listes déroulantes de la page d'ajout

3.2.3.1 Visuel

Pour faciliter la catégorisation des thés, la page d'ajout propose trois listes déroulantes distinctes : Variété, Provenance et Type.

Ces sélecteurs sont stylisés pour s'intégrer harmonieusement au design global de l'application.

Voici leur apparence sur la page :



Chaque liste présente les caractéristiques suivantes :

Un en-tête cliquable avec une icône de flèche indiquant qu'il s'agit d'un menu déroulant

Une zone de défilement lorsque la liste contient de nombreux éléments

Des boutons d'édition et de suppression à côté de chaque élément

Un bouton "Ajouter un élément" en bas pour créer de nouvelles entrées

Lorsqu'un utilisateur clique sur l'en-tête, la liste se déploie et affiche tous les éléments disponibles :



Figure 25 - visuel - listes déroulantes ouvertes

L'utilisateur peut sélectionner un élément en cliquant sur le bouton radio correspondant. Il est également possible de modifier ou supprimer un élément existant grâce aux icônes situées à droite.

Pour ajouter un nouvel élément, l'utilisateur clique sur le bouton "Ajouter un élément", ce qui ouvre une fenêtre de dialogue :

[Insérer capture d'écran de la fenêtre popup d'ajout]

3.2.3.2 Fonctionnement

Le fonctionnement technique de ces listes déroulantes repose sur plusieurs mécanismes JavaScript qui interagissent avec le backend via des appels API.

3.2.3.2.1 Structure HTML

Chaque liste est structurée de manière similaire, comme on peut le voir dans ces extraits du code pour la liste des Provenances :

En premier le bouton pour ouvrir la liste :

```
<!-- Sélection de la provenance -->


<button type="button" onclick="toggleList('provenanceList')" class="w-full flex justify-between items
        <span>Provenance</span>
        <svg id="provenanceIcon" class="w-5 h-5 transform transition-transform duration-200" fill="none"
            <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 91-7 7-7-7" />
    </svg>
</button>


```

Figure 26 - code - bouton - listes déroulantes

Puis ici un foreach qui nous affiche chaque provenance disponible avec deux boutons à sa droite, modifier (qui ouvre une pop-up) et supprimer.

```
<div id="provenanceList" class="hidden space-y-2 mt-4 max-h-48 overflow-y-auto scrollbar-thin">
    @foreach($provenances as $provenance)
        <div class="flex justify-between items-center bg-[#FFEFCD] rounded p-2">
            <label class="flex items-center space-x-2 flex-grow">
                <input type="radio" name="provenance_id" value="{{ $provenance->id_provenance }}" class="text-[#967259]">
                <span>{{ $provenance->nom }}</span>
            </label>
            <div class="flex space-x-2">
                <button type="button" onclick="openEditpopup('provenance', {{ $provenance->id_provenance }}, '{{ $provenance->nom }}'
                    class="text-[#967259] hover:text-[#7d6049]">
                    <svg class="w-5 h-5" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                        <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M15.232 5.232l3.536 3.536m-2.036-5
                    </svg>
                </button>
                <button type="button" onclick="deleteItem('provenance', {{ $provenance->id_provenance }})"
                    class="text-red-500 hover:text-red-700">
                    <svg class="w-5 h-5" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                        <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6L12 12" />
                    </svg>
                </button>
            </div>
        </div>
    @endforeach
</div>
```

Figure 27 - code - liste

Les trois sections (Variété, Provenance, Type) utilisent le même modèle avec simplement des données différentes.

Pour la Gestion de l'ouverture/fermeture des listes déroulantes c'est la fonction "toggleList" qui s'en charge :

```
// Fonction pour basculer l'affichage des listes
function toggleList(listId) {
    const list = document.getElementById(listId);
    const icon = document.getElementById(listId.replace('List', 'Icon'));

    list.classList.toggle('hidden');
    if (list.classList.contains('hidden')) {
        icon.classList.remove('rotate-180');
    } else {
        icon.classList.add('rotate-180');
    }
}
```

Figure 28 -code - fonction - toggleList

Cette fonction alterne la visibilité de la liste et fait pivoter l'icône de flèche pour indiquer l'état actuel (ouvert ou fermé).

Lorsqu'un utilisateur souhaite ajouter ou modifier un élément, un popup apparaît grâce aux fonctions "openAddpopup" et "openEditpopup". Ces fonctions préparent le formulaire avec les bonnes valeurs et affichent la boîte de dialogue. Le bouton de sauvegarde du popup utilise ensuite des requêtes AJAX pour communiquer avec l'API :

```
popupSaveBtn.onclick = async (e) => {
  e.preventDefault();
  e.stopPropagation();
  const type = popupType.value;
  const action = popupAction.value;
  const id = popupItemId.value;
  const name = popupItemName.value;

  if (!name.trim()) {
    showNotification('Veuillez entrer un nom', 'error');
    return;
  }

  try {
    popupSaveBtn.disabled = true;
    const method = action === 'add' ? 'POST' : 'PUT';
    const url = action === 'add' ? `/api/${type}s` : `/api/${type}s/${id}`;

    const response = await fetch(url, {
      method,
      headers: {
        'Content-Type': 'application/json',
        'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-token"]').content
      },
      body: JSON.stringify({ nom: name })
    });

    const data = await response.json();
    if (response.ok) {
      await updateList(type);
      showNotification(data.message || `${type} ${action === 'add' ? 'ajouté' : 'modifié'} avec succès`);
      closepopup();
    } else {
      showNotification(data.message || 'Une erreur est survenue', 'error');
    }
  } catch (error) {
    console.error('Erreur:', error);
    showNotification('Une erreur est survenue lors de la communication avec le serveur', 'error');
  } finally {
    popupSaveBtn.disabled = false;
  }
};
```

3.2.3.2.4 Mise à jour dynamique des listes

Après chaque opération, la fonction `updateList` est appelée pour rafraîchir le contenu de la liste concernée sans avoir à recharger toute la page :

```
async function updateList(type) {
    try {
        const response = await fetch(`api/${type}s/list`);
        if (!response.ok) {
            throw new Error('Erreur lors de la récupération des données');
        }

        const items = await response.json();
        const listContainer = document.getElementById(`${type}List`);
        const selectedValue = document.querySelector(`input[name="${type}_id"]:checked`).value;

        let html = '';
        items.forEach(item => {
            const itemId = item[`id_${type}`];
            html += `
                <div class="flex justify-between items-center bg-[#FFEFCD] rounded p-2">
                    <label class="flex items-center space-x-2 flex-grow">
                        <input type="radio" name="${type}_id" value="${itemId}" ${selectedValue == itemId ? 'checked' : ''} class="text-[#967259]">
                        <span>${item.nom}</span>
                    </label>
                    <div class="flex space-x-2">
                        <button type="button" onclick="openEditpopup('${type}', ${itemId}, '${item.nom}')"
                            class="text-[#967259] hover:text-[#7d049]">
                            <svg class="w-5 h-5" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                                <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M15.232 5.232l3.536 3.536m-2.036-5.036a2.5 2.5 0 113.536 3.536L6.5 2.5" />
                            </svg>
                        </button>
                        <button type="button" onclick="deleteItem('${type}', ${itemId})"
                            class="text-red-500 hover:text-red-700">
                            <svg class="w-5 h-5" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                                <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6L12 12" />
                            </svg>
                        </button>
                    </div>
                </div>
            `;
        });
    };
}
```

Figure 29 - code - fonction - `updateList`

Cette approche permet une expérience utilisateur fluide et réactive, sans les temps de chargement qu'impliquerait un rechargeement complet de la page.

3.2.4 Exportation en PDF

3.2.4.1 Visuel

La fonction d'export PDF permet aux utilisateurs de générer un document PDF à partir de la page de détails d'une liste de thés, et ce, grâce à un bouton "Exporter en PDF" que voici :



Figure 30 - visuel - Exporter PDF

Ce document est ensuite téléchargé automatiquement puis peut être imprimé, partagé etc....

Le PDF généré présente un format clair avec :

- Un en-tête contenant le nom de la liste ainsi que sa date de création
- Un compteur indiquant le nombre total de thés dans la liste
- Une liste présentant chaque thé avec ses caractéristiques principales

Voici un exemple de PDF généré :

Petit-déjeuner

Créée le 08/05/2025

4 Thés

Darjeeling First Flush

Type: Thé noir

Provenance: Japon

Variété: Darjeeling

Assam

Type: Thé vert

Provenance: Chine

Variété: Sencha

Earl Grey

Type: Thé noir

Provenance: Inde

Variété: Earl Grey

Ceylan Orange Pekoe

Type: Thé noir

Provenance: Inde

Variété: Darjeeling

Figure 31 - visuel - PDF

Cette mise en page facilite la lecture et permet d'avoir une vue d'ensemble claire de la collection.

3.2.4.2 Fonctionnement

Le bouton décrit dans la partie précédente, envoie une requête à la route "listes.export" qui est traitée par la méthode export du "ListeController".

Ensuite ce "ListeController" utilise la bibliothèque de Laravel "DomPDF" pour transformer un template HTML en document PDF :

```
/*
 * Génère un PDF de la liste de thés
 *
 * Cette méthode crée un document PDF contenant :
 * - Les informations de base de la liste (nom, date)
 * - La liste détaillée des thés avec leurs caractéristiques
 * - Un en-tête et pied de page personnalisés
 *
 * @param Liste $liste La liste à exporter en PDF
 * @return \Illuminate\Http\Response Le fichier PDF à télécharger
 */
public function generatePDF(Liste $liste)
{
    $liste->load('thes.type', 'thes.provenance', 'thes.variete');

    $pdf = PDF::loadView('listes.pdf', compact('liste'));

    return $pdf->download($liste->nom . '.pdf');
}
```

Figure 32 - code - fonction - generatePDF

Cette méthode :

- Charge toutes les relations nécessaires pour afficher les détails complets des thés
- Utilise la façade PDF pour charger le template de vue "listes.pdf" provenant du fichier "pdf.blade.php"
- Transmet les données de la liste au template
- Retourne une réponse pour télécharger directement le fichier avec le nom de la liste

Le template "listes.pdf.blade.php" est un document HTML avec des styles CSS intégrés, optimisé pour le rendu PDF :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{{ $liste->nom }}</title>
    <!-- Styles CSS pour le PDF -->
    <style>...
    </style>
</head>
<body>
    <!-- En-tête avec le nom de la liste et sa date de création -->
    <div class="header">
        <h1>{{ $liste->nom }}</h1>
        <p>Créée le {{ $liste->date_creation->format('d/m/Y') }}</p>
    </div>

    <!-- Compteur du nombre de thés dans la liste -->
    <div class="tea-count">
        <h2>{{ $liste->thes->count() }} Thés</h2>
    </div>

    <!-- Grille des thés de la liste -->
    <div class="tea-grid">
        @foreach($liste->thes as $the)
            <!-- Carte individuelle pour chaque thé -->
            <div class="tea-card">
                <h3>{{ $the->nom }}</h3>
                <p><strong>Type:</strong> {{ $the->type->nom }}</p>
                <p><strong>Provenance:</strong> {{ $the->provenance->nom }}</p>
                <p><strong>Variété:</strong> {{ $the->variете->nom }}</p>
            </div>
        @endforeach
    </div>
</body>
</html>
```

Figure 33 - code - template PDF

Le template utilise une mise en page en grille CSS pour organiser les thés en liste, avec un style cohérent avec l'identité visuelle de base de l'application.

Cette méthode d'export PDF présente plusieurs avantages :

- Séparation des responsabilités : Le contrôleur gère la logique, tandis que le template gère la présentation
- Facilité de maintenance : Le template peut être modifié indépendamment du code PHP à tout moment

- Expérience utilisateur : Le téléchargement direct du fichier avec un nom approprié facilite grandement l'utilisation

En combinant un contrôleur bien structuré et un template HTML/CSS adapté, l'application offre une fonction d'export PDF élégante et professionnelle qui valorise les données souhaitées.

Décrire la réalisation "physique" de votre projet

- *les répertoires où le logiciel est installé*
- *la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
- *les versions des systèmes d'exploitation et des outils logiciels*
- *la description exacte du matériel*
- *le numéro de version de votre produit !*
- *programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.*

NOTE : Evitez d'inclure les listings des sources, à moins que vous ne désiriez en expliquer une partie vous paraissant importante. Dans ce cas n'incluez que cette partie...techniquement les problèmes rencontrés et comment les résoudre

J'ai un dossier "NeoMasson_TPI" qui est aussi la base de mon repos GitHub qui contient trois principaux sous-dossiers :

- 1 Code, qui contient le dossier principal du projet avec tous les composants Laravel et toutes les pages de codes.
- 2 Documentation, qui renferme tout ce qui n'est pas du code et qui pourrait servir au projet, par exemple les maquettes, des tutos d'installation ou mon rapport TPI.
- 3 Rendu, c'est ici que je conserve mes fichiers ZIP de rendu. Cela peut aider pour le versioning ou si un expert a besoin que je lui renvoie l'un des rendus.

1.2.1. Description physique du projet

1.2.1.1. Répertoires où le logiciel est installé :

1.2.1.2. Liste des fichiers et description :

3.1. Description des tests effectués

1.2.2. Test 1

1.3. Erreur restante

1.4. Liste des documents fournis

Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions

- *le rapport de projet*
- *le manuel d'Installation (en annexe)*
- *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
- *autres...*

2. Conclusions

2.2. Objectifs et Résultats

Objectif	Description	Statut
	.	✓ Atteint
		✓ Atteint
		✓ Atteint
		✓ Atteint

Points positifs / négatifs

- *Difficultés particulières*

2.3. Suites possibles pour le projet (évolutions & améliorations)

2.3.1. Suite 1...

3. Annexes

3.2. Aides externes

Durant ce projet, j'ai utilisé différents outils qui m'ont permis de simplifier certaines tâches :

Reverso.net et ChatGPT 4.0 m'ont permis de corriger mon orthographe en soulignant mes fautes et d'améliorer ma formulation en me donnant des conseils.

Looping.exe m'a facilité la création d'un script SQL à partir de mon MCD.

Claude 3.5 Sonnet pour quand je n'arrivais pas à résoudre une erreur dans mon code, que je ne comprenais pas un snippet d'internet ou pour implémenter une fonction complexe. Je lui décrivais l'erreur ou la fonction souhaitée puis il m'expliquait de manière claire avec des exemples comment je pouvais résoudre le problème ou implémenter la fonction.

3.3. Résumé du rapport du TPI / version succincte de la documentation

3.4. Sources – Bibliographie

1. <https://github.com>
2. <https://stackoverflow.com>
3. <https://www.reverso.net/orthographe/correcteur-francais/>
4. <https://dev.mysql.com/downloads/installer/>
5. <https://laravel.com/docs/12.x>
6. <https://tailwindcss.com/>
7. <https://tailwindcss.com/docs/installation/framework-guides/laravel/vite>
8. <https://nodejs.org/en/download>
9. <https://getcomposer.org/>
10. https://www.reddit.com/r/laravel/comments/155cw2e/does_anyone_here_prefer_phunit_to_pest/?rdt=35242
11. <https://tecfa.unige.ch/guides/tie/html/mysql-intro/mysql-intro-7.html>
12. <https://louisvandevelde.be/index.php?dos=my&fic=meris>
13. <https://www.figma.com/>
14. <https://manjuparkavi.medium.com/10-essential-javascript-snippets-every-developer-should-know-6c68cabaa082>
15. <https://www.solidpepper.com/blog/modele-physique-de-donnees-mdp-definition-enjeux-exemple>
16. <https://redketchup.io/color-picker>
17. <https://kinsta.com/fr/blog/laravel-eloquent/#:~:text=Laravel%20est%20un%20framework%20PHP,dans%20les%20bases%20de%20donn%C3%A9es.>
18. <https://johackim.com/tailwind-css>
19. <https://www.pairform.fr/comprendre-lutilisation-don-delete-cascade-en-sql.html>
20. <https://learn.microsoft.com/fr-fr/sql/t-sql/statements/alter-schema-transact-sql?view=sql-server-ver16>
21. <https://blog-gestion-de-projet.com/strategie-de-test/>

22. <https://openclassrooms.com/forum/sujet/enregistrer-une-liste-dans-une-bdd>
23. <https://frvecteezy.com/png-gratuit/>
24. <https://www.afci-ju.ch/fichiers/Planification-par-la-mthode-des-6-tapes-15.pdf>
25. <https://heroicons.com/>

3.5. Glossaire

1. TPI : est un projet d'examen éliminatoire de fin de CFC en informatique qui comprend une partie technique basée sur un cahier des charges et une autre partie documentation dont ce rapport fait partie.
2. Laravel : Framework PHP qui suit le modèle MVC (Modèle–Vue–Contrôleur) et offre tout ce qu'il faut pour créer rapidement des applications web : gestion des routes, accès aux bases de données avec Eloquent, authentification, etc.
3. Tailwind CSS : Bibliothèque de classes CSS prêtes à l'emploi pour construire une interface sans écrire de CSS pure : marges, couleurs, typographie... tout se définit directement dans l'HTML.
4. Vues Blade : Moteur de templates dans Laravel. Les fichiers ...blade.php permettent d'imbriquer de l'HTML et des directives simples (@if, @foreach, @extends...) pour générer les pages du côté du serveur.
5. Middleware : Petits programmes qui s'intercalent entre la requête HTTP et ton contrôleur pour effectuer des vérifications ou des transformations (authentification, filtrage, journalisation...).
6. CRUD : Acronyme pour **C**reate, **R**ead, **U**pdate, **D**elete : les quatre opérations de base pour gérer n'importe quelle ressource (par exemple : créer, afficher, modifier ou supprimer un enregistrement de thé).
7. Diagramme de Gantt : Vue graphique d'un planning de projet sous forme de barres horizontales qui montrent la durée et l'ordre des tâches dans le temps.
8. Dompdf : Librairie PHP qui convertit du HTML/CSS en PDF, idéale pour générer des exports imprimables (dans notre cas les listes de thés).
9. Wireframe : Maquette simple, sans style ni couleurs, qui sert à définir la structure et l'agencement des éléments d'une page avant de passer à un design plus détaillé.
10. Clé primaire composite : Clé primaire constituée de plusieurs colonnes dans une table relationnelle, elle garantit qu'aucune combinaison de ces champs ne se répète. Exemple : (fk_id_liste, fk_id_the) pour la table d'association entre listes et thés.
11. SGBD (Système de Gestion de Base de Données): Logiciel qui stocke, organise et sécurise les données, et permet d'exécuter des requêtes pour les lire ou les modifier. Il peut servir de visualisation graphique de notre base de données.
12. Snippets éprouvés : Petits morceaux de code déjà testés et fiables, que l'on peut copier-coller pour gagner du temps et éviter les erreurs sur des fonctionnalités courantes comme les pop-ups et les filtres JavaScript.

13. IDE (Environnement de Développement Intégré) : Application tout-en-un pour coder, tester, éditer, déboguer. Exemples populaires pour Laravel : Visual Studio Code, PhpStorm.

3.6. Table des illustrations

Figure 1 - Exemple - Séquence de planification	5
Figure 2 Exemple - Diagramme Gantt	5
Figure 3 - Graphique des tâches planifiées	6
Figure 4 - Maquette - S'enregistrer	9
Figure 5 - Maquette - S'authentifier	9
Figure 6 - Maquette - Accueil	10
Figure 7 - Maquette - Détails du Thé	10
Figure 8 - Maquette - Ajouter un thé	11
Figure 9 - Maquette - Listes	12
Figure 10 - Maquette - Détails d'une liste	12
Figure 11 - BDD - MCD	13
Figure 12 - BDD - MLD	13
Figure 13 - BDD - MPD	14
Figure 14 - Barre de recherche	18
Figure 15 - affichage - recherche	18
Figure 16 - visuel - filtrage	18
Figure 17 - exemple - filtrage	18
Figure 18 - exemple - tri	19
Figure 19 - code - fonction – sortTable	20
Figure 20 - code – section filtrage de recherche - HTML	21
Figure 21 - code - début du script	21
Figure 22 - code - fonction - filterTable	22
Figure 23 - code - EventListener	22
Figure 24 - visuel - listes déroulantes	23
Figure 25 - visuel - listes déroulantes ouvertes	23
Figure 26 - code - bouton - listes déroulantes	24
Figure 27 - code - liste	24
Figure 28 -code - fonction - toggleList	25
Figure 29 - visuel - Exporter PDF	27
Figure 30 - visuel - PDF	28
Figure 31 - code - fonction - generatePDF	29
Figure 32 - code - template PDF	30

Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur) ... Et de toutes les aides externes (noms)

1. Journaux de travail

2. Manuel d'Installation

5.4.1. Manuel 1...

5.4.1.1. Introduction

5.4.1.2. Prérequis

5.4.1.3. Explication des commandes

5.4.2. Résolution des problèmes

5.4.2.1. Erreur de dépendances manquantes

5.4.2.2. Vérification de l'installation

5.4.3. Conclusion

3. Manuel d'Utilisation