

Logiciel de gestion de projet

Tapia Josu

Mercader Lenny

I) Introduction et présentation du logiciel.....	4
1) Vue global du projet.....	4
2) Champ d'action du projet.....	4
3) Description et compétence de l'utilisateur.....	4
II) Description globale.....	5
1) Environnement du système.....	5
2) Exigence fonctionnelles.....	5
a) Authentification et autorisation des utilisateurs.....	5
b) Tableau de bord.....	6
c) Gestion de projets et de tâches.....	7
d) Collaboration d'équipe.....	9
3) Exigence non fonctionnelles.....	10
a) Performance.....	10
b) Fiabilité et Disponibilité.....	10
c) Sécurité.....	10
d) Utilisabilité et Accessibilité.....	10
e) Compatibilité.....	10
f) Maintenabilité et Extensibilité.....	11
g) Portabilité.....	11
h) Testabilité.....	11
III) Caractéristique technique.....	12
1) Maquette UI/UX.....	12
a) Compte.....	13
b) Tableau de bord.....	14
c) Projet.....	15
d) Tâche.....	15
2) Arborescence du logiciel.....	17
3) Fonctionnement global du logiciel.....	18
a) Composant externes.....	18
i) Laravel Breeze.....	18
ii) sqlite.....	18
b) Gestion des contrôleurs.....	18
c) Middleware.....	20
d) Requests.....	20
e) Kernel.....	21
f) Gestions des models.....	21
g) Notifications.....	22
h) View/Components.....	22
i) bootstrap.....	23
j) config.....	23
k) database.....	24
i) factories.....	24

ii) migrations.....	24
iii) seeders.....	26
l) public.....	26
m) resources.....	27
i) Gestion des vues.....	27
1) admin/users.....	27
2) auth.....	27
3) components.....	28
4) layouts.....	28
5) profile.....	28
1) partials.....	28
6) projects.....	29
7) tasks.....	29
n) routes.....	30
o) storage.....	31
p) tests.....	31
4) Description du lancement.....	32
l) Docker.....	32
II) Manuel.....	32

I) Introduction et présentation du logiciel

1) Vue global du projet

Cet outil permettra aux utilisateurs de planifier, de suivre et de gérer des projets. Il intégrera des fonctionnalités telles que la gestion des tâches, la collaboration en équipe, le suivi du temps et la création de rapports. Le projet nécessitera l'application de divers principes et pratiques de génie logiciel abordés tout au long du cours, notamment l'ingénierie des exigences, la conception, la mise en œuvre, les tests et le déploiement de logiciels.

Nous verrons dans ce premier chapitre le périmètre du projet ainsi que la description de l'utilisateur-type.

Nous verrons ensuite dans un second chapitre toutes les fonctionnalités présentes dans l'application ainsi que son environnement.

Pour finir dans le troisième chapitre nous définirons la documentation et tous les aspects technique de l'application

2) Champ d'action du projet

L'application est un logiciel de gestion de projet d'une entreprise. Elle est conçue pour être fonctionnelle peu importe l'entreprise et est donc personnalisable dans le type de tâches et les délais. Elle a pour objectif de simplifier la gestion des tâches et la planification du projet. Plus précisément, elle permet au manager et membre des équipes de gérer, consulter et conclure des tâches. A noter que les membres de l'équipe peuvent seulement consulter le projet, seuls les managers peuvent le créer, l'éditer ou le clore. Ils ont également un système de chat, de gestion de temps et de notification.

L'administrateur doit pouvoir assigner et modifier les rôles des membres, ainsi qu'avoir un accès débogage pour modifier, ajouter ou supprimer des tâches/projets. Il aura aussi accès à l'historique des tâches et projets.

3) Description et compétence de l'utilisateur

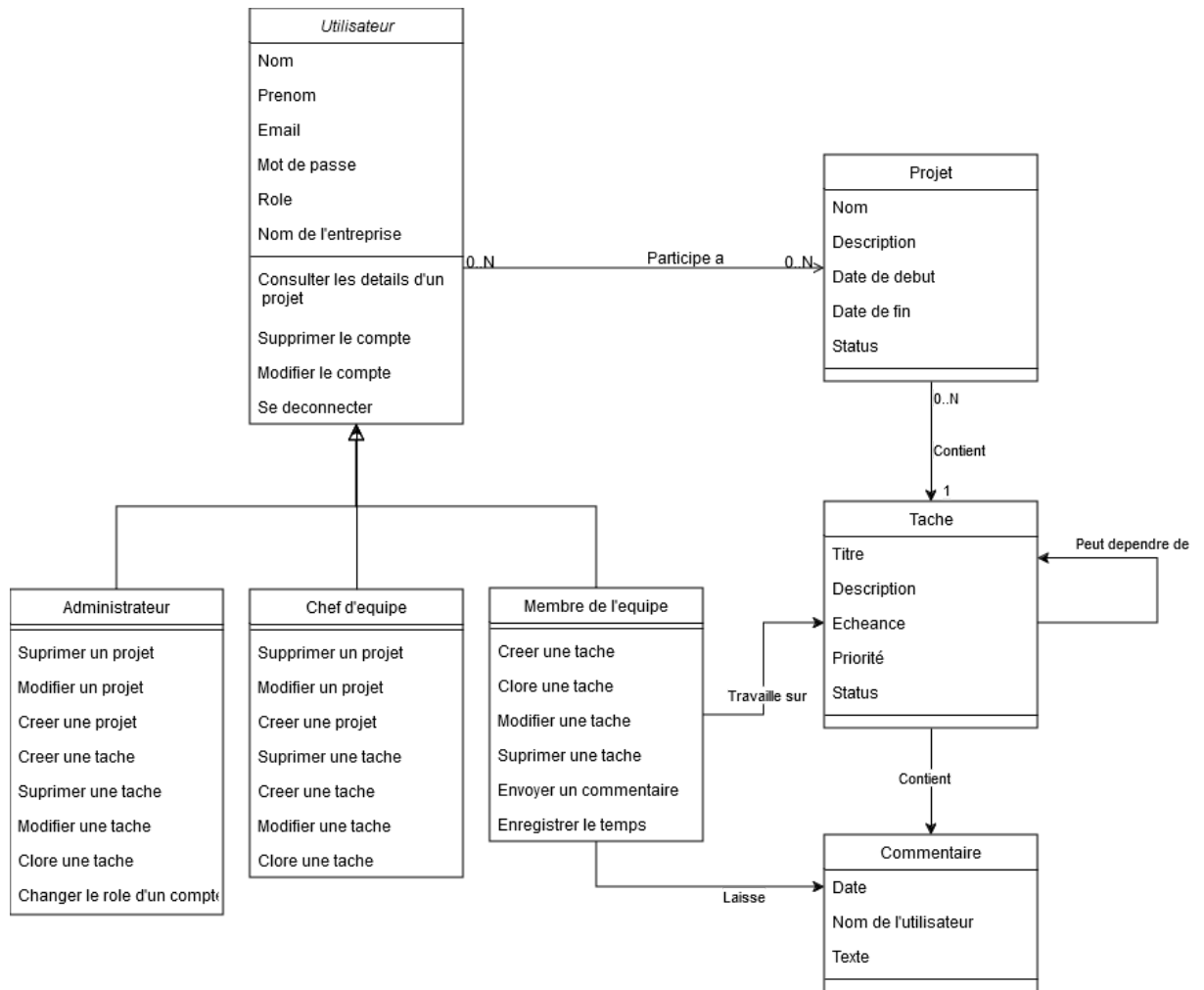
L'utilisateur est un employé de l'entreprise, il est considéré comme connaissant les différentes parties prenant intervenant sur son projet.

Il est considéré comme ayant connaissance des fonctionnalités de gestion de compte telles que la connexion à un compte, sa création etc.

Il est également considéré comme étant rigoureux et sérieux, et donc qu'il est attentif à ses notifications et remplit toutes les cases et caractéristiques lorsqu'il soumet un formulaire (par exemple quand il crée un projet).

II) Description globale

1) Environnement du système



2) Exigence fonctionnelles

a) Authentification et autorisation des utilisateurs

Use case : Création de compte

Acteur pouvant l'utiliser : Tous

Description : L'utilisateur peut créer un compte, il doit entrer un username et son mot de passe.

Description type des étapes :

- L'utilisateur clique sur "s'inscrire"
- Il entre son mot de passe et son username
- L'application vérifie si les données sont valables (mot de passe correspond et les identifiants n'existent pas déjà)
- L'utilisateur doit entrer des informations supplémentaires tels que son nom, prénom, etc.
- Une fois confirmé, le compte est créé.

Use case : Connexion au compte

Acteur pouvant l'utiliser : Tous

Description : l'utilisateur entre ses coordonnées pour se connecter

Description type des étapes :

- L'utilisateur clique sur "se connecter"
- Il entre ses identifiants
- L'application vérifie qu'ils existe dans la BD
- S' ils n'existe pas alors il propose de créer un compte ou de réessayer, sinon l'utilisateur est connecté

Use case : Déconnexion au compte

Acteur pouvant l'utiliser : Tous

Description : Déconnecte l'utilisateur

Description type des étapes :

- L'utilisateur clique sur se déconnecter
- L'utilisateur est déconnecté

Use case : Gérer le compte

Acteur pouvant l'utiliser : Tous

Description : L'utilisateur peut modifier ses information personnel, son rôle (avec l'accord de l'admin), et ses identifiant etc

Description type des étapes :

- L'utilisateur clique sur "modifier le compte"
- L'utilisateur fait les modification
- L'utilisateur clique sur "sauvegarder les modification
- L'application vérifie que tout est correct puis les appliques

b) Tableau de bord

Use case : Voir la liste des projet

Acteur pouvant l'utiliser : Tous

Description : Affiche la liste de tous les projets en cours

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances et leur progression

Use case : Voir la liste des tâches

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe et Membre de l'équipe

Description : Affiche la liste de toutes les tâches pour un projet.

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Afficher les tâches"
- La liste de toutes les tâches auxquelles il est assigné s'affichent

Use case : Visionner l'avancement d'un projet

Acteur pouvant l'utiliser : Membres assignés au projet

Description : Fournit un récapitulatif de toutes les informations relatives à un projet (tâche en cours, employé assigné, échéance etc)

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- Le résumé de l'avancement d'un projet s'affiche

c) Gestion de projets et de tâches

Use case : Créer un projet

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe

Description : Créer des projets et renseigner des détails tels que le nom du projet, la description, les dates de début et de fin et les membres de l'équipe assignée.

Description type des étapes :

- L'utilisateur clique sur "Créer un projet"
- L'utilisateur entre toutes les informations nécessaire
- L'application vérifie les informations
- Si les informations sont correcte alors le projet est créé

Use case : Modifier un projet

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe

Description : L'utilisateur peut modifier toutes les informations relatives au projet

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Modifier le projet"
- L'utilisateur entre toutes les informations nécessaire
- L'application vérifie les informations
- Si les informations sont correcte alors le projet est modifié

Use case : Clore un projet

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe

Description : L'utilisateur peut mettre fin au projet

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Clore le projet"
- Un message de confirmation apparaît
- Le projet est clôt

Use case : Créer une tâche

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe, Membre d'équipe

Description : L'utilisateur peut créer une tâche, il doit alors renseigner l'affectation, l'échéance, un titre, un descriptif, les dépendance et la priorité

Description type des étapes :

- L'utilisateur clique sur "Afficher la liste des projets"
- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Afficher les tâches"
- L'utilisateur clique sur "Créer une tâche"
- L'utilisateur entre toutes les informations nécessaire
- L'application vérifie les informations
- Si les informations sont correcte alors la tâche est créé

Use case : Modifier une tâche

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe, Membre d'équipe

Description : L'utilisateur peut modifier toutes les informations relatives à une tâche

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Afficher les tâches"
- L'utilisateur choisit une tâche
- L'utilisateur clique sur "Modifier la tâche"
- L'utilisateur modifie toutes les informations nécessaire
- L'application vérifie les informations
- Si les informations sont correcte alors la tâche est modifié

Use case : Clore une tâche

Acteur pouvant l'utiliser : Administrateur, Chef d'équipe, Membre d'équipe

Description : L'utilisateur peut clore une tâches

Description type des étapes :

- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Afficher les tâches"
- L'utilisateur choisit une tâche
- L'utilisateur clique sur "Finir la tâche"
- Un message de confirmation apparaît demandant le temps assigné à la tâche (l'utilisateur peut la laisser vide)
- La tâche est terminé

d) Collaboration d'équipe

Use case : Laisser un commentaire

Acteur pouvant l'utiliser : Membres assignés à la tâche

Description : L'utilisateur peut laisser un commentaire sur un projet (pour indiquer si il faut ajouter ou modifier quelque chose)

Description type des étapes :

- L'utilisateur clique sur "Afficher la liste des projets"
- La liste de tous les projet s'affichent avec indiqué leurs échéances
- L'utilisateur choisit un projet
- L'utilisateur clique sur "Afficher les tâches"
- L'utilisateur choisit une tâche
- L'utilisateur clique sur "Ajouter un commentaire"
- L'utilisateur entre le contenu du commentaire
- Le commentaire s'envoie

3) Exigence non fonctionnelles

a) Performance

L'outil de gestion de projet doit offrir des performances élevées pour garantir une expérience utilisateur fluide et réactive. Les spécifications en termes de performance incluent :

- **Temps de réponse** : Les pages doivent se charger en moins de 2 secondes dans 95 % des cas.
- **Capacité de traitement** : Le système doit être capable de prendre en charge plusieurs utilisateurs simultanés sans dégradation des performances.
- **Temps de traitement des tâches** : Les opérations de création, de mise à jour ou de suppression des tâches et des projets doivent s'exécuter en moins de 2 secondes.

b) Fiabilité et Disponibilité

Le système doit assurer un haut niveau de fiabilité et de disponibilité pour minimiser les interruptions de service.

- **Disponibilité** : Le système doit être accessible 99.9 % du temps par mois.
- **Tolérance aux pannes** : En cas de défaillance, on doit pouvoir rétablir le service en moins de 5 minutes.
- **Récupération des données** : En cas de panne, les données ne doivent pas être perdues. Le système doit disposer de mécanismes de sauvegarde quotidienne automatique.

c) Sécurité

La sécurité est un aspect critique du projet, surtout à cause de la gestion des données confidentielles des utilisateurs et des projets.

- **Authentification et autorisation** : Un contrôle d'accès basé sur les rôles (administrateur, chef de projet, membre de l'équipe) sera mis en place.

d) Utilisabilité et Accessibilité

L'outil doit être facile à utiliser et accessible à un large public d'utilisateurs.

- **Interface intuitive** : L'interface utilisateur doit être intuitive et conforme aux normes de conception UX.
- **Temps d'apprentissage** : Les nouveaux utilisateurs doivent être capables de naviguer dans le système et de réaliser des tâches de base en moins de 15 minutes sans formation.

e) Compatibilité

Le système doit être compatible avec les navigateurs, appareils et environnements courants.

- **Navigateurs supportés** : Le système doit être pleinement compatible avec Google Chrome, Mozilla Firefox, Microsoft Edge et Safari.
- **Compatibilité multiplateforme** : L'application doit fonctionner sur les systèmes d'exploitation Windows, macOS et Linux.

- **Conception réactive** : L'application doit être utilisable sur les ordinateurs de bureau, les tablettes et les smartphones.

f) Maintenabilité et Extensibilité

Le système doit être simple à maintenir et à faire évoluer.

- **Facilité de mise à jour** : Les mises à jour de l'application ne doivent pas interrompre le service plus de 10 minutes.

- **Extensibilité** : Le système doit permettre l'ajout de nouvelles fonctionnalités (ex. : nouveaux modules) sans modification de l'architecture principale.

- **Documentation** : Le code doit être bien commenté, et une documentation détaillée doit être fournie pour faciliter la maintenance future.

g) Portabilité

L'application doit être portable et adaptable à différents environnements d'exécution.

- **Conteneurisation** : L'application doit être conteneurisée en utilisant Docker afin d'assurer une compatibilité multi-plateforme.

- **Migration** : Le système doit être migrable entre plusieurs plateformes cloud (par ex., AWS, Azure, GCP) avec des efforts minimaux.

h) Testabilité

Le système doit être conçu pour permettre des tests efficaces.

- **Automatisation des tests** : Les tests doivent être automatisés avec un cadre (framework) de tests continu (CI/CD).

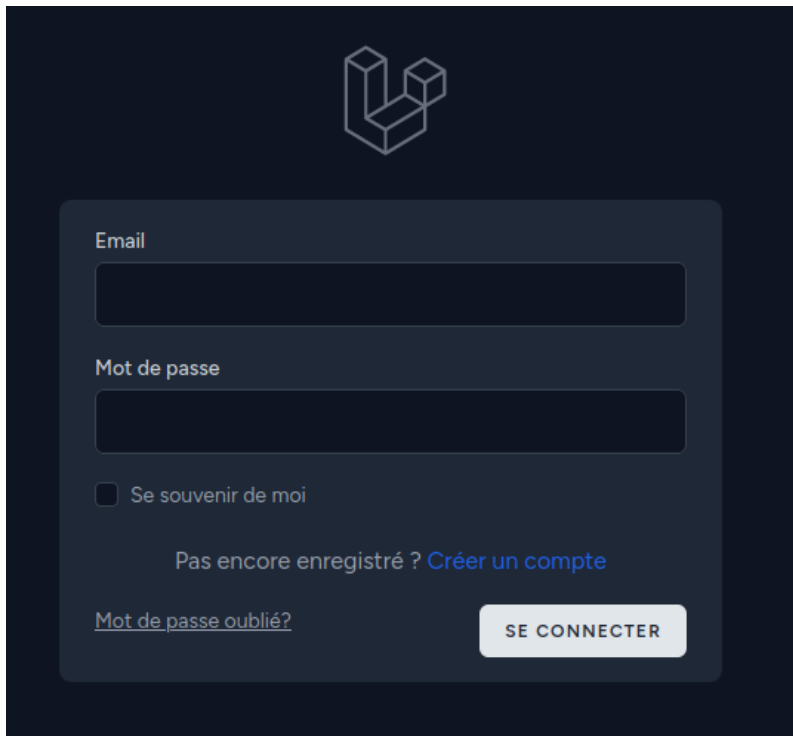
- **Couverture des tests** : Le taux de couverture des tests unitaires doit atteindre au moins 80 %.

- **Environnement de test** : Le système doit inclure un environnement de pré-production pour les tests.

III) Caractéristique technique

1) Maquette UI/UX

a) Compte



The login form is centered on a dark blue background. At the top is a logo consisting of three white 3D cubes arranged in an 'L' shape. Below the logo is a light blue rounded rectangle containing the login fields. The fields are labeled 'Email' and 'Mot de passe'. There is a checkbox labeled 'Se souvenir de moi'. Below the fields is a link 'Pas encore enregistré ? Créer un compte'. At the bottom left is a link 'Mot de passe oublié?' and at the bottom right is a button labeled 'SE CONNECTER'.

Email

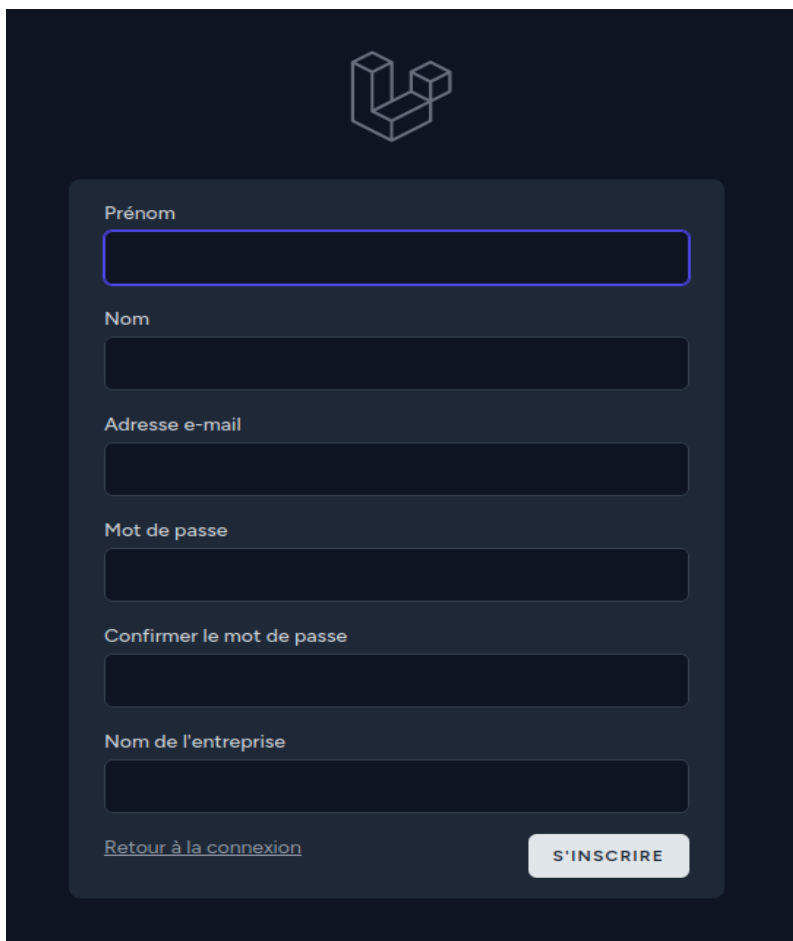
Mot de passe

☐ Se souvenir de moi

Pas encore enregistré ? [Créer un compte](#)

[Mot de passe oublié?](#)

SE CONNECTER



The registration form is centered on a dark blue background. At the top is the same logo of three white 3D cubes in an 'L' shape. Below the logo is a light blue rounded rectangle containing the registration fields. The fields are labeled 'Prénom', 'Nom', 'Adresse e-mail', 'Mot de passe', 'Confirmer le mot de passe', and 'Nom de l'entreprise'. At the bottom left is a link 'Retour à la connexion' and at the bottom right is a button labeled 'S'INSCRIRE'.

Prénom

Nom

Adresse e-mail

Mot de passe

Confirmer le mot de passe

Nom de l'entreprise

[Retour à la connexion](#)

S'INSCRIRE

Modifier le Profil

Informations du profil

Mettez à jour les informations de profil et l'adresse e-mail de votre compte.

Nom

Email

admin@example.com

ENREGISTRER

Mettre à jour le mot de passe

Assurez-vous que votre compte utilise un mot de passe long et aléatoire pour rester sécurisé.

Mot de passe actuel

Nouveau mot de passe

Confirmer le mot de passe

ENREGISTRER

Supprimer le compte

Une fois votre compte supprimé, toutes ses ressources et données seront définitivement supprimées. Avant de supprimer votre compte, veuillez télécharger toutes les données ou informations que vous souhaitez conserver.

SUPPRIMER LE COMPTE

b) Tableau de bord

Tableau de bord

Gérer les utilisateurs

Tableau de bord

Ajouter un projet

Notifications

Vous avez été assigné à la tâche: Tache 1

[Voir la tâche](#)

sfgqrgh

s<regqehr

Status: in_progress

Dates: 2024-12-12 - 2024-12-13

Consulter les détails

rgqrg

qergqeh

Status: in_progress

Dates: 2024-12-12 - 2024-12-26

Consulter les détails

Test 5

zsergqerg

Status: in_progress

Dates: 2024-12-12 - 2024-12-13

Consulter les détails

c) Projet

Créer un projet

Nom du projet

Description

Date de début

mm / dd / yyyy

Date de fin

mm / dd / yyyy

Utilisateurs assignés

Créer le projet

Tableau de bordGérer les utilisateurs

Détails du projetListe des tâchesRecapitulatif du temps

Test 5

zsergqerg

Statut in progress

Dates : 2024-12-12 - 2024-12-13

Utilisateurs assignés :
Admin
Manager
Member
a

Modifier le projetSupprimer le projet

Dashboard

Modifier le projet : Projet de GL

Nom du projet

Projet de GL

Description

test

Date de début

12 / 11 / 2024

Date de fin

12 / 18 / 2024

Utilisateurs assignés

Mettre à jour

d) Tâche

Tableau de bordGérer les utilisateurs

Détails du projetListe des tâchesRecapitulatif du temps

Ajouter une tâche

Titre	Priorité	Date d'échéance	Statut	Actions
Tache 1	Medium	2024-12-16	Open	Consulter les détails

Dashboard

Créer une tâche pour le projet "Projet de GL"

Titre de la tâche

Description

Priorité


Faible

Date d'échéance

mm / dd / yyyy

Affecter des utilisateurs

Créer la tâche

Tableau de bordGérer les utilisateurs

Détails du projetListe des tâchesRecapitulatif du temps

Détails de la tâche

Tache 1

ceci est une tâche

Priorité

Medium

Date d'échéance

2024-12-16

Statut

Open

Modifier

Supprimer

Clore la tâche

Assigner du temps

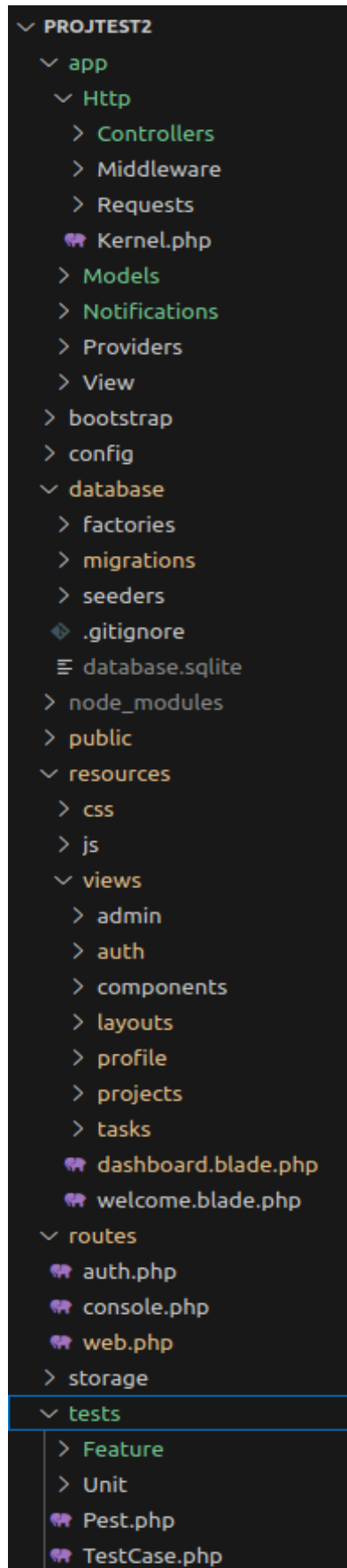
Commentaires

Aucun commentaire pour l'instant.

Ajoutez un commentaire...

Envoyer

2) Arborescence du logiciel



3) Fonctionnement global du logiciel

a) Composant externes

i) Laravel Breeze

Breeze met en place plusieurs fonctionnalités prêtes à l'emploi :

- 1) Authentification (Login, Logout, Register)
- 2) Réinitialisation de mot de passe (Mot de passe oublié)
- 3) Middleware de protection des routes (ex : auth middleware)
- 4) Vues Blade pour les interfaces utilisateur de base

ii) sqlite

SQLite est une base de données relationnelle légère et embarquée. Contrairement à MySQL ou PostgreSQL, elle ne nécessite pas de serveur. Les données sont stockées dans un simple fichier .sqlite ou .db sur le disque.

Caractéristiques principales

- 1) Légère et rapide : Idéale pour les applications mobiles, embarquées et les tests.
- 2) Sans serveur : Fonctionne localement sans service de base de données en arrière-plan.
- 3) Autonome : Le fichier de base de données contient tout (données, tables, index).
- 4) Langage SQL : Supporte la majorité des commandes SQL standards (SELECT, INSERT, UPDATE, etc.).

b) Gestion des contrôleurs

Le dossier Controllers contient les fichiers qui gèrent la logique de l'application.

- **AdminController.php** : Responsable de la gestion des fonctionnalités administratives de l'application.
- **CommentController.php** : Gère les opérations liées aux commentaires des utilisateurs.
- **Controller.php** : Le contrôleur de base de Laravel, étendu par les autres contrôleurs.

- **DashboardController.php** : Gère l'affichage du tableau de bord de l'utilisateur.
- **NotificationController.php** : S'occupe de la gestion des notifications pour les utilisateurs.
- **ProfileController.php** : Gère les informations et les mises à jour du profil utilisateur.
- **ProjectController.php** : Responsable des opérations liées aux projets dans l'application.
- **TaskController.php** : Gère les tâches associées à chaque projet et les actions liées.

Le dossier Auth dans Controllers contient des contrôleurs responsables des processus d'authentification et de gestion des utilisateurs dans l'application.

- **AuthenticatedSessionController.php** : Gère la création et la gestion des sessions après une connexion réussie.
- **ConfirmablePasswordController.php** : Responsable de la gestion du mot de passe pour confirmer l'utilisateur lors de certaines actions.
- **EmailVerificationNotificationController.php** : Gère l'envoi des notifications de vérification par e-mail.
- **EmailVerificationPromptController.php** : Affiche la demande de vérification d'email après l'inscription.
- **LoginController.php** : Gère la logique de connexion des utilisateurs.
- **NewPasswordController.php** : Gère la demande de réinitialisation du mot de passe.
- **PasswordController.php** : Gère l'édition et la modification des informations de mot de passe de l'utilisateur.
- **PasswordResetLinkController.php** : Responsable de l'envoi du lien de réinitialisation du mot de passe.
- **RegisteredUserController.php** : Gère l'enregistrement de nouveaux utilisateurs.
- **VerifyEmailController.php** : Gère la vérification de l'adresse e-mail après inscription.

Ces contrôleurs sont essentiels pour les processus d'authentification de l'application. Ils traitent les actions liées à l'inscription, la connexion, la réinitialisation du mot de passe ou encore la gestion des sessions.

Chaque contrôleur est conçu pour gérer une partie spécifique de l'application, en suivant les principes du modèle **Model View Controller**.

Les contrôleurs se présentent sous la forme suivante :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class Controller
{
    //
}
```

- **use** : permettent d'importer les différents fichiers (models associé par exemple) que nous avons besoins
- **class** : un controller est une classe représentant un objet qui représente les actions que le model peut faire. On lui met simplement des méthodes selon l'usage qu'on souhaite. Pas d'attribut cependant, ils sont réservés uniquement aux models.

c) Middleware

Le dossier Middleware contient des classes qui interceptent les requêtes HTTP pour les filtrer ou ajouter des logiques spécifiques avant qu'elles n'atteignent les contrôleurs, comme la vérification de l'authentification de l'utilisateur.

d) Requests

Le dossier Requests contient des classes qui gèrent la validation des données envoyées par l'utilisateur dans les formulaires avant qu'elles ne soient traitées par les contrôleurs, comme la validation des informations de connexion (**LoginRequest.php**) ou de mise à jour du profil (**ProfileRequest.php**).

e) Kernel

Le fichier **Kernel.php** contient la logique centrale du cycle de vie des requêtes et des tâches à exécuter dans l'application, telles que l'enregistrement des middlewares, des commandes artisan et la gestion des requêtes HTTP.

f) Gestions des models

Le dossier Models contient des fichiers représentant les entités principales de l'application, qui interagissent avec la base de données.

- **Comment.php** : Modèle représentant les commentaires associés aux tâches. Il interagit avec la table des commentaires dans la base de données.
- **Notification.php** : Modèle représentant les notifications dans l'application. Il gère la communication d'événements (comme l'assignation de tâches) et l'enregistrement des notifications.
- **Project.php** : Modèle représentant les projets. Il contient les informations principales liées aux projets (nom, description, dates, etc.) et interagit avec la table des projets.
- **Task.php** : Modèle représentant les tâches dans l'application. Il gère les tâches liées aux projets, y compris leurs titres, descriptions et échéances.
- **User.php** : Modèle représentant les utilisateurs de l'application. Il contient les informations des utilisateurs, gère l'authentification et la gestion des rôles.

Ces modèles permettent de manipuler facilement les données de la base, en définissant la logique métier et les relations entre les différentes entités.

Les modèles sont sous la forme suivante :

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Project extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'description',
        'start_date',
        'end_date',
        'status'
    ];

    public function users()
    {
        return $this->belongsToMany(User::class, 'project_user');
    }

    public function tasks()
    {
        return $this->hasMany(Task::class);
    }
}

```

- **use** : permettent d'importer les différents fichiers que nous avons besoins
- Ensuite on écrit simplement une classe d'un objet en créant les attributs et méthodes

Pour créer un modèle, on entre la commande suivante :

php artisan make:model NomDuModel

g) Notifications

Le dossier Notifications contient les classes qui gèrent les notifications envoyées aux utilisateurs, telles que **TaskAssigned.php** et **TaskDeadlineNotification.php**, permettant d'envoyer des notifications liées aux tâches et leurs échéances.

h) View/Components

Le dossier View/Components contient des composants réutilisables dans les vues de l'application. **AppLayout.php** définit la structure de la page principale de l'application, tandis que **GuestLayout.php** gère celle de la page destinée aux invités, permettant ainsi une navigation fluide et cohérente selon le type d'utilisateur.

i) bootstrap

Le dossier bootstrap contient les fichiers nécessaires pour initialiser l'application. **app.php** configure l'application en définissant les services et paramètres essentiels. **providers.php** enregistre les fournisseurs de services qui gèrent les composants clés de l'application.

j) config

Le dossier config contient des fichiers qui définissent la configuration de l'application Laravel. Voici un résumé du rôle de chaque fichier :

- **app.php** : Contient les configurations principales de l'application, comme le nom de l'application, l'environnement, le fournisseur de services, et d'autres paramètres globaux.
- **auth.php** : Définit les configurations liées à l'authentification, comme les guards (méthodes de connexion), les providers et les configurations des mots de passe.
- **cache.php** : Configure les paramètres du cache, notamment le type de cache utilisé (fichiers, base de données, Redis, etc.).
- **database.php** : Contient la configuration de la base de données, incluant les connexions aux différentes bases (MySQL, SQLite, etc.) et les paramètres liés à l'ORM Eloquent.
- **filesystems.php** : Configure les systèmes de fichiers de l'application, permettant de gérer le stockage local, les disques distants (S3, FTP, etc.) et l'accès aux fichiers.
- **logging.php** : Définit les configurations pour la journalisation des erreurs et des événements dans l'application (logs), y compris le type de logging (fichiers, Slack, etc.).
- **mail.php** : Configure les paramètres liés à l'envoi d'e-mails, comme les drivers (SMTP, Mailgun, etc.) et les paramètres d'authentification.
- **queue.php** : Contient les configurations relatives aux queues de travail, comme les drivers de queue (base de données, Redis, etc.) et les paramètres de traitement des jobs.
- **services.php** : Définit les services externes (API, services tiers) utilisés par l'application, comme les clés d'API, les secrets, et autres configurations externes.
- **session.php** : Configure les paramètres de gestion des sessions, y compris le driver (fichiers, base de données, etc.), le temps d'expiration, et la gestion des cookies.

Ces fichiers permettent de centraliser et gérer facilement tous les paramètres de configuration de l'application.

k) database

Le dossier **database** contient les fichiers liés à la gestion de la base de données dans l'application. Cela inclut les migrations, les usines (factories) pour générer des données factices, les migrations, ainsi que les seeders pour remplir la base de données avec des données initiales.

i) factories

Le dossier factories dans le répertoire database est utilisé pour générer des données factices (ou "dummy data") pour les tests ou les démonstrations. Cela est particulièrement utile lors du développement pour tester des fonctionnalités de l'application sans avoir à entrer manuellement des données.

- **UserFactory.php** : Ce fichier contient la définition d'une "usine" pour créer des utilisateurs. Laravel utilise ce modèle pour générer des utilisateurs fictifs avec des données réalistes, comme des noms, des e-mails, etc., et vous pouvez facilement en générer un grand nombre pour les tests ou la base de données de développement.

ii) migrations

Le dossier **migrations** contient des fichiers définissant la structure de la base de données.

- **0001_01_01_000000_create_users_table.php** : Crée la table des utilisateurs, qui contient des informations comme le nom, l'email et les mots de passe des utilisateurs.
- **0001_01_01_000001_create_cache_table.php** : Crée une table pour gérer le cache de l'application, ce qui peut améliorer les performances en stockant des données fréquemment utilisées.
- **0001_01_01_000002_create_jobs_table.php** : Crée une table pour gérer les jobs en file d'attente, permettant à l'application de traiter des tâches de manière asynchrone.
- **2024_11_28_151643_create_projects_table.php** : Crée la table des projets, stockant des informations comme le nom du projet, sa description, ses dates de début et de fin.
- **2024_12_04_175020_create_project_user_table.php** : Crée une table pivot pour gérer la relation entre les projets et les utilisateurs, permettant à un projet d'avoir plusieurs utilisateurs.

- **2024_12_04_190937_create_task_tables.php** : Crée la table des tâches, contenant des informations sur les tâches, telles que les titres, descriptions et dates limites.
- **2024_12_06_090959_add_role_to_users_table.php** : Ajoute une colonne pour gérer le rôle des utilisateurs (par exemple : admin, membre de l'équipe, etc.).
- **2024_12_12_091936_create_comments_table.php** : Crée la table des commentaires, permettant aux utilisateurs d'ajouter des remarques ou des retours sur les tâches.
- **2024_12_12_114138_create_notifications_table.php** : Crée la table des notifications, stockant les alertes ou messages que les utilisateurs recevront concernant les tâches ou projets.

Ces fichiers permettent de versionner les modifications de la base de données et de gérer facilement la création et les mises à jour des tables via les migrations.

```

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('project_user', function (Blueprint $table) {
            $table->id();
            $table->foreignId('project_id')->constrained()->onDelete('cascade');
            $table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('project_user');
    }
};

```

- **use** : permettent d'importer les différents fichiers (models associé par exemple) que nous avons besoins
- **class** : on crée une classe représentant un objet. On lui met simplement des méthodes selon l'usage qu'on souhaite.

Pour créer une migration, on entre la commande suivante sur le terminale :

php artisan make:migration nom_migration

Et après avoir effectué toutes les modifications souhaité :

php artisan migrate

iii) seeders

Le dossier seeders contient des fichiers qui permettent de remplir la base de données avec des données initiales ou de test. Voici un résumé de leur contenu :

- **AdminSeeder.php** : Ce fichier est utilisé pour insérer un ou plusieurs utilisateurs et administrateurs dans la base de données, lors de l'installation ou pour la configuration initiale de l'application.

l) public

Le dossier public contient les fichiers accessibles publiquement via le serveur web, tels que les fichiers CSS, JavaScript et images.

- **css** : Ce dossier contient les fichiers CSS utilisés pour le style visuel de l'application.
- **js** : Ce dossier contient les fichiers JavaScript nécessaires pour l'interactivité et les fonctionnalités côté client de l'application.
- **.htaccess** : Un fichier de configuration pour le serveur Apache, souvent utilisé pour les redirections, la gestion des URL et la sécurité.
- **favicon.ico** : L'icône associée à l'application, affichée dans l'onglet du navigateur.
- **index.php** : Le point d'entrée de l'application web Laravel. Ce fichier est exécuté à chaque requête HTTP pour gérer l'application.
- **mix-manifest.json** : Un fichier généré par Laravel Mix, qui contient des informations sur les versions et les chemins des fichiers compilés (CSS, JavaScript, etc.).
- **robots.txt** : Un fichier utilisé pour indiquer aux moteurs de recherche quelles pages doivent être explorées ou ignorées.

Il sert à rendre disponible au public tous les fichiers statiques nécessaires au bon fonctionnement de l'application.

m)resources

Le dossier **resources** contient les fichiers qui sont utilisés lors de la construction et du rendu des pages web de l'application. **css** pour le design et le style visuel, **js** qui gère les comportements interactifs et les fonctionnalités côté client ainsi que **views** qui contient les fichiers **Blade**, qui sont des vues HTML utilisées pour afficher le contenu dynamique à l'utilisateur. Ces fichiers sont rendus côté serveur par Laravel.

i) Gestion des vues

Les views sont des fichiers qui contiennent le code HTML et PHP pour afficher le contenu aux utilisateurs. Ces fichiers sont responsables de la présentation des données envoyées par les controllers.

- **dashboard.blade.php** : Ce fichier est responsable de l'affichage du tableau de bord pour les utilisateurs connectés, où les informations importantes telles que les tâches, projets, ou notifications sont présentées. Ce fichier sert également de base pour afficher des informations dynamiques via les contrôleurs, qui les passent comme variables à cette vue.

1) admin/users

Centralise les vues relatives à la gestion des utilisateurs par les administrateurs.

- **index.blade.php** : Ce fichier est la vue de la page dédiée à la gestion des utilisateurs dans le tableau de bord administrateur.

2) auth

Le dossier auth dans views contient des vues Blade utilisées pour la gestion de l'authentification des utilisateurs.

- **confirm-password.blade.php** : Affiche le formulaire pour confirmer le mot de passe lorsque l'utilisateur tente d'accéder à des pages sensibles.
- **forgot-password.blade.php** : Affiche le formulaire permettant aux utilisateurs de demander la réinitialisation de leur mot de passe.
- **login.blade.php** : Contient le formulaire pour que les utilisateurs se connectent à leur compte.
- **register.blade.php** : Affiche le formulaire d'inscription pour permettre aux utilisateurs de créer un compte.
- **reset-password.blade.php** : Permet aux utilisateurs de réinitialiser leur mot de passe après avoir demandé une réinitialisation.
- **verify-email.blade.php** : Affiche la page de vérification de l'email après l'inscription de l'utilisateur.

Ce dossier centralise toutes les vues liées à l'authentification des utilisateurs dans l'application.

3) components

Le dossier components dans views contient des composants Blade réutilisables pour différentes parties de l'application. Ils sont générés automatiquement.

4) layouts

Le dossier layouts dans views contient les fichiers de mise en page (layouts) principaux utilisés pour l'ensemble de l'application.

- **app.blade.php** : Fichier de mise en page principal utilisé pour la plupart des pages de l'application. Il définit la structure globale de la page, comme l'en-tête, le pied de page et la navigation.
- **guest.blade.php** : Fichier de mise en page utilisé spécifiquement pour les pages destinées aux invités (par exemple, la page de connexion ou d'inscription).

Ces fichiers servent de base pour les autres vues de l'application et permettent de centraliser la gestion de l'interface.

5) profile

Le dossier profile dans views contient des vues liées à la gestion du profil de l'utilisateur.

- **edit.blade.php** : Vue principale pour l'édition du profil, qui inclut les différentes sections et formulaires relatifs à la gestion du profil utilisateur.

1) *partials*

Contient des sous-vues réutilisables pour les différentes sections du profil utilisateur, permettant de mettre à jour ou supprimer des informations liées au profil.

- **delete-user-form.blade.php** : Formulaire permettant à l'utilisateur de supprimer son compte.
- **update-password-form.blade.php** : Formulaire pour la mise à jour du mot de passe de l'utilisateur.
- **update-profile-information-form.blade.php** : Formulaire pour la mise à jour des informations personnelles du profil de l'utilisateur.

6) projects

Le dossier projects dans views contient des vues liées à la gestion des projets.

- **create.blade.php** : Vue utilisée pour la création de nouveaux projets. Elle contient un formulaire permettant à l'utilisateur de saisir les informations nécessaires pour ajouter un projet.
- **edit.blade.php** : Vue permettant de modifier les informations d'un projet existant. Elle contient un formulaire pré-rempli avec les données du projet que l'utilisateur peut modifier.
- **show.blade.php** : Vue qui affiche les détails d'un projet spécifique, utilisée pour afficher des informations complètes sur un projet après sa création.

Ces fichiers permettent de gérer l'affichage et les actions associées à la création, la modification et l'affichage des projets dans l'application.

7) tasks

Le dossier tasks dans views contient des vues relatives à la gestion des tâches.

- **create.blade.php** : Vue qui permet à l'utilisateur de créer une nouvelle tâche. Elle contient un formulaire permettant de saisir les informations nécessaires, telles que le titre, la description, les dates et la priorité de la tâche.
- **edit.blade.php** : Vue permettant à l'utilisateur de modifier une tâche existante. Elle affiche un formulaire avec les informations pré-remplies de la tâche à modifier.
- **index.blade.php** : Vue qui affiche la liste de toutes les tâches. Elle sert à afficher les tâches de manière générale.
- **show.blade.php** : Vue qui affiche les détails d'une tâche spécifique. Elle permet à l'utilisateur de consulter toutes les informations concernant une tâche donnée, comme sa description, son statut et ses dates de début et de fin.

Ces fichiers sont responsables de l'affichage et de la gestion des différentes actions liées aux tâches dans l'application.

n) routes

Le dossier routes contient les fichiers qui définissent les différentes routes de l'application.

- **auth.php** : Contient les routes liées à l'authentification, telles que celles pour la connexion, l'enregistrement des utilisateurs, la réinitialisation des mots de passe et la vérification des e-mails.
- **console.php** : Définit les routes utilisées par les commandes Artisan de Laravel. Ces routes sont liées aux consoles et aux tâches planifiées.
- **web.php** : Contient les routes web principales de l'application. Elles gèrent les requêtes HTTP qui sont destinées à être traitées par un navigateur web, comme l'affichage des pages d'accueil, des profils d'utilisateur ou encore des pages de gestion des tâches.

Ces fichiers de route permettent de définir les chemins (URL) auxquels les utilisateurs peuvent accéder, ainsi que les contrôleurs et actions qui y sont associés.

```
<?php

use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;
use App\Http\Middleware\CheckRole;
use App\Http\Controllers\AdminController;
use App\Http\Controllers\ProjectController;
use App\Http\Controllers\DashboardController;
use App\Http\Controllers\TaskController;

Route::get('/', function () {
    return view('auth.login');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});
```

- **use** : permettent d'importer les différents fichiers (models associé par exemple) que nous avons besoins
- On écrit chaque route avec **Route::TypeDeLaRoute('/chemin', function(){ fonction a faire});**
- Les routes permettent de définir comment les requêtes HTTP (comme GET, POST, etc.) sont associées aux contrôleurs ou aux actions spécifiques dans l'application.

o) storage

Le dossier storage est utilisé pour stocker des fichiers générés par l'application.

p) tests

Le dossier tests contient les fichiers utilisés pour effectuer les tests unitaires et fonctionnels de l'application. Il se divise en deux sous-dossiers principaux :

- **Feature** : Contient des tests fonctionnels qui simulent l'utilisation de l'application, comme les interactions entre différentes parties du système.
- **Unit** : Contient des tests unitaires qui vérifient le bon fonctionnement de petites unités de code, telles que des fonctions ou des méthodes.

Les fichiers Pest.php et TestCase.php sont des fichiers de base pour structurer les tests, Pest étant un framework de test qui simplifie l'écriture des tests dans Laravel.

4) Description du lancement

I) Docker

- a) On se place dans le root

```
C:\Users\rexfo>cd C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2
```

- b) On lance le docker avec
sudo docker-compose up --build -d

```
C:\Users\rexfo\Desktop\ProjetGL\projTest2\projTest2>sudo docker-compose up --build -d
```

- c) On accède sur <http://localhost:8000/>
d) Pour reseed dans le cas ou les comptes pré-conçus ne sont pas pris en compte
php artisan migrate --seed

II) Manuel

- a) On se place dans le root

```
C:\Users\rexfo>cd C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2
```

- b) On installe les dependance composer

```
C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2>composer install
```

- c) On installe les modules de nodes

```
C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2>npm install
```

- d) npm run dev

```
C:\Users\rexfo\Desktop\ProjetGL\projTest2\projTest2>npm run dev
```

- e) On copie l'environnement

```
C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2>copy .env.example .env
```

- f) On fait les migrations

```
C:\Users\rexfo\Desktop\ProjetGL\projTest2\projTest2>php artisan migrate --seed
```

- g) On génère la clé

```
C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2>php artisan key:generate
```

- h) On lance l'application

```
C:\Users\rexfo\Desktop\ProjetGL\git\projTest2\projTest2>php artisan serve  
Working is not supported on this platform
```

- i) On va sur <http://127.0.0.1:8000> sur son navigateur