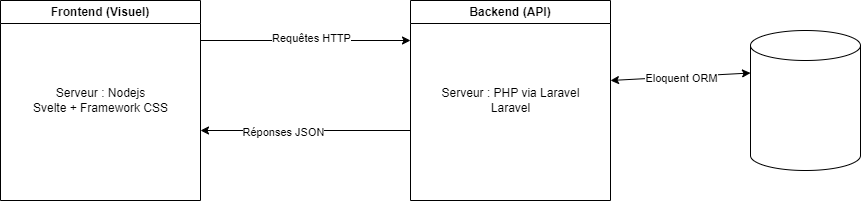
Projet web

# Introduction

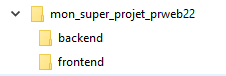
Pour mettre en pratique et améliorer vos compétences dans le développement web, vous aller créer un projet de A à Z sur le thème de votre choix. Le projet sera basé sur les technologies et langages suivants : HTML, CSS, PHP, JS. Vous allez créer le backend sous forme d’API avec le framework Laravel, et l’espace admin sera mis en place avec Sanctum (voir doc Laravel). Le frontend qui se connectera à l’API sera développé avec Svelte.



# Consignes

Le projet portera sur une application web ou un site web. Vous êtes libres de choisir le thème et le sujet du projet. Voici quelques règles :

* !! Vous **devez** utiliser les outils fournis par Laravel. Par exemple, pour la base de données, vous devez utiliser les migrations et les seeders. Utilisez des contrôleurs, des ressources, des requests.
* Vous devez respecter les best practices de Svelte et Laravel.
* Votre projet doit comporter 2 dossiers, un pour le frontend, et un autre pour le backend. Voici un exemple de structure :



*Laravel*

*Svelte*

* !! Le backend ne contiendra **pas de vues, ni de visuels**. De ce fait, vous devez intégrer le framework CSS sur frontend.
* Les images sont stockées sur le backend. Vous devez donc stocker leur UID (unique ID) dans votre base de données. C’est une colonne de type VARCHAR, unique.

Il devra respecter les conditions suivantes :

* Une partie publique qui affichera les données
* Un espace administrateur pour permettre les CRUD
* Minimum 1 envoi de fichiers (file upload)
* Minimum 5 tables (tables relationnelles non comprises)
* Minimum 1 table relationnelle (relation N-N)
* Pour chacune des 5 tables : un modèle et une classe
* Pour 3 tables au minimum : CRUD complet (frontend + backend)
* Les erreurs doivent être bien gérées et affichées clairement (ex : email déjà saisi en bdd)
* Les actions réussies doivent être bien gérées et affichées clairement (ex : Notification « La donnée a bien été enregistrée)
* Des composants réutilisables (navigation, footer, entête)
* Appliquer du CSS
* Utiliser un framework CSS (hors bootstrap)
* Un design agréable à utiliser
* Responsive
* Bugless
* Un code propre, commenté, bien organisé
* Une structure de dossier facile à comprendre et à utiliser
* Programmation orientée objets avec des classes
* 2 dossiers, un pour le frontend et un pour le backend
* Une documentation (collection) Postman pour votre API

Bonus :

* Tout autre élément développé qui n’est pas demandé.

# Organisation

Le projet se déroulera de la manière suivante :

1. Choix du sujet
2. Choix du Framework CSS
3. Modéliser la base de données
4. Développer le CRUD pour 1 table dans le backend (API)
5. Développer le CRUD pour 1 table dans le frontend (Connexion à l’API)
6. Rendu intermédiaire noté
7. Développer le CRUD pour 1 table dans le backend (API)
8. Développer le CRUD pour 1 table dans le frontend (Connexion à l’API)
9. Présentation notée
10. Développer le CRUD pour 1 table dans le backend (API)
11. Développer le CRUD pour 1 table dans le frontend (Connexion à l’API)
12. Ajouter la connexion et déconnexion avec Sanctum
13. Améliorer l’application en appliquant du CSS personnel
14. Développer d’autres fonctionnalités, selon vos envies (points bonus)

# Astuces / recommandations / informations

* Trouvez une librairie (package) qui permette de générer facilement et rapidement les collections postman depuis Laravel.
* Utilisez un outil tel que ShareX pour générer des captures d’écran propres et rapides pour les documentations demandées.
* Testez vos routes API au fur et à mesure du développement avec Postman.
* Profitez du temps à disposition pendant les cours pour éviter le travail à la maison, et profitez de l’aide des enseignants pour ne pas rester bloqué(e).
* Utilisez les documentations officielles de Laravel et Svelte, elles sont très bien documentées.
* Les routes doivent être définies dans le frontend et dans le backend. Elles ne seront pas similaires.

# Évaluation

Font parties intégrantes de l’évaluation :

* Documentation utilisateur
  + Respecter le canevas de l’école
  + Structuration claire
  + Agréable à lire
  + Propre
* Rendu intermédiaire
* Présentation
  + Présenter la base de données (2’)
  + Présenter la structure du code (2’)
  + Présenter les fonctionnalités (4’)
  + Pour la présentation du XXXX, au minimum :  
    1 crud complet backend + frontend, la doc utilisateur doit présenter les fonctionnalités développées
* Collection Postman
* Dossier « technique » à rendre
  + Modélisation de la bdd
  + Script SQL pour régénérer la bdd
  + Code développé

## Rendu intermédiaire

**Frontend – 42 points**

|  |  |  |
| --- | --- | --- |
| Critère | Remarque |  |
| Facilité de démarrer le projet | Inclut un readme ? Facile de faire démarrer sur la machine locale | 3 |
| Bug ? | Pas de bug : tous les points, bug : aucun point | 5 |
| Organisation des dossiers (pages + composants) | La structure est claire et facilement compréhensible. Les dossiers sont bien nommés et organisés. Les fichiers sont bien nommés et organisés. | 5 |
| Create | Utilisation, propreté, réutilisabilité, gestion et affichage des erreurs, mise à jour des éléments HTML | 5 |
| Read | Utilisation, propreté, réutilisabilité, gestion et affichage des erreurs, mise à jour des éléments HTML | 5 |
| Update | Utilisation, propreté, réutilisabilité, gestion et affichage des erreurs, mise à jour des éléments HTML | 5 |
| Delete | Utilisation, propreté, réutilisabilité, gestion et affichage des erreurs, mise à jour des éléments HTML | 5 |
| Commentaires | Utilisation judicieuse, commentaires utiles | 3 |
| Await / fetch / async / export / const / let | Utilisation correcte des mots clés | 3 |
| Routing |  | 3 |

**Backend – 45 points**

|  |  |  |
| --- | --- | --- |
| Critère | Remarque |  |
| Facilité de démarrer le projet | Inclut un readme ? Facile de faire démarrer sur la machine locale | 3 |
| Bug ? | Pas de bug : tous les points, bug : aucun point | 5 |
| Organisation des dossiers (pages + composants) | La structure est claire et facilement compréhensible. Les dossiers sont bien nommés et organisés. Les fichiers sont bien nommés et organisés. | 5 |
| Create | Utilisation, propreté, réutilisabilité, gestion des erreurs, réponses en format json, paramètres des routes | 2 |
| Read | Utilisation, propreté, réutilisabilité, gestion des erreurs, réponses en format json, paramètres des routes | 2 |
| Update | Utilisation, propreté, réutilisabilité, gestion des erreurs, réponses en format json, paramètres des routes | 2 |
| Delete | Utilisation, propreté, réutilisabilité, gestion des erreurs, réponses en format json, paramètres des routes | 2 |
| Commentaires | Utilisation judicieuse, commentaires utiles | 3 |
| Migrations | <https://laravel.com/docs/10.x/migrations#main-content> | 3 |
| Seeders | <https://laravel.com/docs/10.x/seeding#writing-seeders> | 3 |
| API Resources | <https://laravel.com/docs/10.x/eloquent-resources#main-content> | 3 |
| Validation | <https://laravel.com/docs/10.x/validation#main-content> | 3 |
| Routing | <https://laravel.com/docs/10.x/routing> | 3 |
| Models | <https://laravel.com/docs/10.x/eloquent> | 3 |
| Relations | <https://laravel.com/docs/10.x/eloquent-relationships> | 3 |