



Projet MY Self Training

27.03.23

—

Sandy RAZAFITRIMO

etsik

www.etsik.com

Vue d'ensemble

Présentation technique

MY Self Training est une application de mobile learning et de micro learning développée en PHP avec le framework Laravel. La base de données de l'application est en MySQL, ce qui assure une gestion efficace des données.

L'application est développée en mode mobile first avec une architecture progressive webapp, ce qui permet aux utilisateurs de profiter de l'expérience d'apprentissage sur toutes les plateformes, y compris sur les smartphones. Le JS est écrit en vanilla JS, ce qui rend l'application légère et rapide.

MY Self Training dispose d'un backend qui permet aux gestionnaires d'ajouter facilement des cours et des quizz à l'application via un ordinateur de bureau. Le backend est conçu de manière conviviale pour garantir une expérience utilisateur agréable. Un autre backend client permet aux utilisateurs d'accéder à leurs statistiques, ce qui leur permet de mesurer leur progression et leur performance.

Grâce à la combinaison de PHP, Laravel, MySQL, JS et Progressive Webapp, MY Self Training est une application de mobile learning de haute qualité qui offre une expérience d'apprentissage exceptionnelle aux utilisateurs. Elle permet aux apprenants de tous niveaux d'apprendre rapidement et efficacement grâce à des cours courts et interactifs, accompagnés de quizz formatifs et sommatifs pour valider leur compréhension.


L'application est facile à utiliser et à administrer, ce qui en fait un choix idéal pour les gestionnaires de formation et les apprenants.

Présentation marketing

MY Self Training est une application de mobile learning révolutionnaire qui permet aux utilisateurs d'apprendre facilement et rapidement grâce à des cours courts et interactifs. Les cours sont conçus pour être accessibles à tout moment et n'importe où, ce qui les rend idéaux pour les personnes occupées qui ont peu de temps à consacrer à l'apprentissage.

Chaque cours est accompagné d'un quizz formatif et d'un quizz sommatif, qui aident les utilisateurs à valider leur compréhension et à mesurer leur progression. Les quizz formatifs sont conçus pour être courts et amusants, tandis que les quizz sommatifs permettent aux utilisateurs de mettre en pratique les connaissances qu'ils ont acquises.

MY Self Training est parfait pour les personnes qui cherchent à se perfectionner et à acquérir de nouvelles compétences en peu de temps. Il peut être utilisé pour apprendre une variété de sujets, des langues étrangères aux compétences professionnelles.



L'application est facile à utiliser et offre une expérience d'apprentissage immersive et stimulante pour les utilisateurs de tous niveaux.

Modèle concurrent : <https://www.sololearn.com>

Objectifs

Mise en place d'une version bêta sous 2 Semaines :

- Mettre en place un suivi de cours qui permet aux utilisateurs de visualiser leur progression dans l'apprentissage, avec la possibilité de marquer les cours comme terminés ou en cours.
- Ajouter des activités de base telles que des cours et des QCM, et permettre aux utilisateurs de voir leur score après chaque activité.
- Mettre en place une gestion des utilisateurs pour permettre aux utilisateurs de sauvegarder leur progression et leur score.
- Autoriser l'accès aux utilisateurs à tous les cours.
- Pour la gestion des utilisateurs, il n'est pas nécessaire de mettre en place une gestion entreprise pour le moment, cela sera ajouté ultérieurement..

Fonctionnalités attendues

En bleues, les fonctionnalités prioritaires pour la beta.

Pour l'entreprise :

- Création d'un profil société
- Création d'une URL société personnalisée
- Personnalisation de la société (couleurs, logo, baseline)
- Ajout d'utilisateurs avec des dates d'activité spécifiées et accès à certains cours
- Suivi de connexion et statistiques des utilisateurs (présence, complétude, note, validation, etc.)

Pour l'employé de l'entreprise :

- Suivi de cours personnalisé avec indication de la complétude (pourcentage de cours suivis)

- Progression dans les apprentissages
- Bilan global de l'apprentissage avec indication du pourcentage de cours suivis et des notes obtenues.
- Intégration d'une IA pour répondre aux diverses questions

Pour le gestionnaire de l'application :

- Ajout d'une entreprise
- Gestion des droits des entreprises (accès à certains cours)
- Création de cours et d'activités (ordre des cours, des quizz)
- Création des quizz sommatifs.
- Liste non exhaustive de données statistiques attendues :
 - Nombre de participants actif
 - taux d'abandon
 - Nombre de connexion / nb interaction par user / jour
 - Parcours des apprenants
 - Note
 - Connexion par module par user
 - Somme des connexions par module
 - NB - Connexion dans le temps / par jour
 - Nombre de personne connecté par j

Technique

<https://github.com/littleblackman/mysefltraining>

A la racine il y a le schéma de la BDD, ainsi qu'un dumb de la BDD actuelle.

Choix de développement

Stack technique : PHP, vanilla JS, SASS, Node

Framework : Laravel

Contraintes

- Développement orienté Service : thin controllers and fat models (en service en priorité)
- Un dossier Service peut être créé dans le dossier "app"
- Respect des principes SOLID
- Utilisation du Git Flow (serveur sous Plesk) (la branche dev s'appelle 'pre-prod')
- L'anglais est la langue utilisée partout : code, fonctions, commit, etc. Seuls les commentaires en pull request sont en français.
- Il faut changer le nom de l'application c'est dorénavant "MY Self Training"
- Il faut prévoir l'utilisation de la langue anglaise.

Au niveau du CSS et du visuel, ainsi que de la librairie CSS :

Vous pouvez la modifier au besoin, il faut faire au mieux pour vous. De préférence pas de bootstrap ni de JS trop encombrant.

Model

Le système de migration de Laravel n'a pas été utilisé, il faudra donc mettre à jour cette partie (non prioritaire). Un dump de la BDD est dispo sur github.

Il faut également renommer les tables qui sont au pluriel en les mettant au singulier. (Ex: pour "courses" qui devient "course"). C'est il me semble le fonctionnement par défaut de Laravel mais je préfère au singulier et garder le pluriel pour les tableaux d'objets.

De même je ne fais de syntaxe, dans l'exemple suivant on parle de 'activity' . Si on a un tableau de plusieurs objets, on ne nomme 'activities' mais 'activitys'. Cela me paraît plus simple.

Dans la BDD, tous les champs n'ont pas été ajouté, il faut donc les ajouter:

- created_at, updated_at, created_by, updated_by pour toutes les tables
- il faut renommer "quizzes" en "quiz"
- chaque table a bien sûr un "id" unique en auto-increment (sauf éventuellement les jointures). Ex: la table course a "id" pour identifiant unique (et non pas "course_id", cela ne sera que pour les clés étrangères).
- dans le modèle joint est une juste modélisation visuel, les types de champs peuvent et doivent être modifiés (ex: certains "is_active" sont en varchar(45) au lieu d'être des booléens)

Le principe est que :

- 1 cours porte sur un sujet complet (ex: "Créer un cours en elearning")
- 1 cours est composé de plusieurs chapitres (ex: "choisir ses outils, concevoir un cours, etc.")
- 1 chapitre est composé de plusieurs leçons (ex pour le chapitre "choisir ses outils" on plusieurs leçons : "choisir son LMS", "choisir son outils auteurs", "créer des vidéos")
- 1 chapitre est composé de plusieurs Quiz qui peuvent s'intercaler entre les différentes leçons.
- Chaque chapitre est composé de leçons et de quizz ordonnés les uns par rapport aux autres.
- L'email est le login de l'utilisateur. Il est donc unique.
- Chaque user appartient à une compagnie (pour les users sans company on peut créer une company.)

Gestion des Users et des droits

Ma préférence est la suivante :

On crée des rôles et des credentials dans l'application.

Il y a différents rôle : “direction, manager, teacher, learner, guest” et bien sûr les joker “admin” voir “superadmin” pour gérer les CRON, les extracts ou autre.

Chaque “feature” a sa série de credentials. Ex: “course::create”, “course::update”

Chaque rôle a des credentials par défaut.

Chaque user peut avoir plusieurs rôle et donc cumuler les credentials.

A l’authentification on remonte tous les credentials d’un user.

On pourra à terme ajouter 1 par 1 des credentials par user.

Le user est donc lié à la table “role” par une jointure et à la table “credentials” par une autre jointure.

Et sur chaque partie de template ou de controller on peut appeler vérifier le controller ainsi:

```
<?php
    if (!hasCredential('menu::admin') && !hasCredential('menu::manager')) {
        // do the job
    }
```

La fonction hasCredential ayant pour but de vérifier dans la session la présence du credential.

Cette méthode n’est pas obligatoire, mais je pense qu’il faut à terme avoir cette souplesse côté client. Donc si vous en avez une autre plus efficace, plus rapide, bref meilleure, n’hésitez pas. Ce n’est qu’une proposition.

CourseService

Il n’y a pas de table de jointures pour lier les différentes activités entre eux. C’était une option possible. On aurait eu chapter_element qui listait toutes les activités. Il aurait alors fallu faire une boucle sur chapter_element est appelé toutes les éléments 1 par 1. Faire des requêtes dans un foreach. J’ai préféré l’option suivante.

Il faut une table par activité, pour l’instant on a “lesson” et “quiz”. A terme on va ajouter de nouvelles activités.

A terme nous aurons plusieurs activités dans les chapitres, il faut donc prévoir dès maintenant l'évolution avec une méthode dans CourseService qui appellera toutes les activités du chapitre et les classera dans l'ordre.

```
<?php
use App\Models\Chapter;
use App\Models\Lesson;
use App\Models\Quiz;

class CourseService {

    public function getActivitysByChapter(Course $course) {
        $activitysByChapter = [];

        foreach ($course->chapters as $chapter) {

            // get all lesson by chapter
            $lessons = Lesson::where('chapter_id', $chapter->id)
                ->orderBy('activity_order')
                ->get();

            // get all quiz by chapter
            $quizzes = Quiz::where('chapter_id', $chapter->id)
                ->orderBy('activity_order')
                ->get();

            // add others type of activities

            // merge all
            $activitys = $lessons->merge($quizzes)
                ->sortBy('activity_order')
                ->values();

            $activitysByChapter[$chapter->chapter_number] = $activitys;
        }

        return $activitysByChapter;
    }
}
```

Toutes les activités doivent avoir une "type_key", c'est à dire un nom unique qui permettra d'afficher le template.

Quand on affichera toutes les activités dans les template blade on fera une boucle sur les activités on pourra faire un insert du template adéquate. Et à ce moment là seul le template saura comment utiliser les données de la table correspondante.

Donc à chaque nouvelle activité ajoutée, on crée:

- une table avec un type_key
- un template portant le nom du type_key
- un ajout et un merge dans CourseService

Je me suis servi de ce système dans un système de blog où l'article était composé de différents types d'éléments avec des visuels différents : des éléments textes, des textes et images, ou des images seules.

Ca a donné le code suivant :

```
<section id="article_content" class="w-full space-y-8">
    @foreach($elements as $element)
        @include('article.template.'. $element->type->template_name,
        ['element' => $element])
    @endforeach
</section>
```

Le template porte le nom du "type_key" et on lui injecte l'objet lié.

Class ExtendModel

J'ai créé sur un autre projet une classe d'extension du modèle. Je ne sais pas si c'est la bonne méthode dans Laravel (je n'ai démarré Laravel que cette année), mais voilà comment je l'ai codée pour ajouter les champs created_by, updated_by. Les champs created_at, updated_at

La fonction to_array est utilisée pour générer rapidement du json sur toutes les classes

```
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Auth;

class ExtendedModel extends Model {

    public static function boot() {
        parent::boot();
        static::creating(function($model)
        {
            $user = Auth::user();
            $model->created_by = $user->id;
            $model->updated_by = $user->id;
        });
        static::updating(function($model)
        {
            $user = Auth::user();
            $model->updated_by = $user->id;
        });
    }

    public function toArray() {
        $objectArray = get_object_vars($this);
        return $objectArray;
    }
}
```

L'activité des User sur les cours

Pour démarrer on ne va suivre que la progression du user sur chaque cours.

user_course

Pour inscrire un user a un cours on la table user_course

C'est une jointure avec pour démarrer uniquement le champs "is_active" qui autorise l'accès ou non au cours.

A terme on ajoutera en plus des dates de début et de fin.

user_activity

La table user_activity doit servir à mesurer la complétude chapitre par chapitre.

Une ligne est ajoutée à chaque activité démarrée et elle est modifiée à chaque fin d'activité.

Cette table se base sur le course_id, le type_key et l'activity_id.

Quand on affiche les stats d'un user on va chercher tous ses cours (user_course) et ensuite quand on affiche le détail d'un cours, on appelle le cours d'un côté (table course) et toutes son activité sur le cours avec le course_id dans la table user_activity. Il faut classer ces activités comme dans CourseService, mais cette fois au lieu de les classer selon activity_order on crée une nouvelle clé concaténée grâce à activity_id et type_key.

Au moment d'afficher le détail du cours, comme on parcourt le tableau course, on récupère juste les infos dans le tableau user_activity grâce à la nouvelle clé combinée activity_id et type_key.

2 champs sont utilisées pour voir l'état de l'activité :

- is_completed est mis à jour quand l'apprenant valide qu'il a fini le cours, par défaut il est à 0 ou null
- is_validated est mis à jour quand l'activité est une évaluation qui nécessite une validation (ex: une note), il est à 0 en cas d'échec, 1 en réussite et null par défaut quand il n'est pas utilisé.

