AP 2 - Linguist

Documentation Technique

Cette documentation vise à présenter les axes majeurs de la méthode de conception de l'application « Linguist », site web permettant à des utilisateurs d'apprendre des Langues au travers de divers leçons.

Sommaire:

Présentation du projet Linguist

Contexte

Linguist est une application web vous permettant d'apprendre un total de cinq langues. L'apprentissage de langues n'est possible que par des utilisateurs possédant un compte, il sera donc nécessaire, de la part de ces utilisateurs, de mettre en place leur compte personnel.

Le visiteur doit, après une connexion via identifiant et mot de passe, avoir accès à de multiples fonctionnalités, à savoir l'accès à des leçons, un choix de langues, la gestion complète de son profil, ainsi que la gestion du tarif auquel il souhaite souscrire.

Fonctionnalités détaillées

Une page d'inscription et de connexion est mise en place, afin de permettre aux utilisateurs de mettre sur pied leur compte personnel.

A cette fonctionnalité s'ajoute la gestion du profil. Une page dédié permet à l'utilisateur de visualiser l'ensemble des informations le concernant (hormis mot de passe). L'utilisateur devra pouvoir modifier ses informations, et également supprimer son compte. La page permettra d'afficher les langues que l'utilisateur a décidé d'apprendre.

Un total de cinq langues sont disponibles : L'anglais, Le Français, L'italien, L'albanais et le Russe.

Des pages respectives, correspondant à chacune des langues proposées, incluront une brève description de la langue, accompagné d'une illustration associée, et d'un bouton permettant à l'utilisateur de valider son choix.

Architecture et méthodes à mettre en œuvre

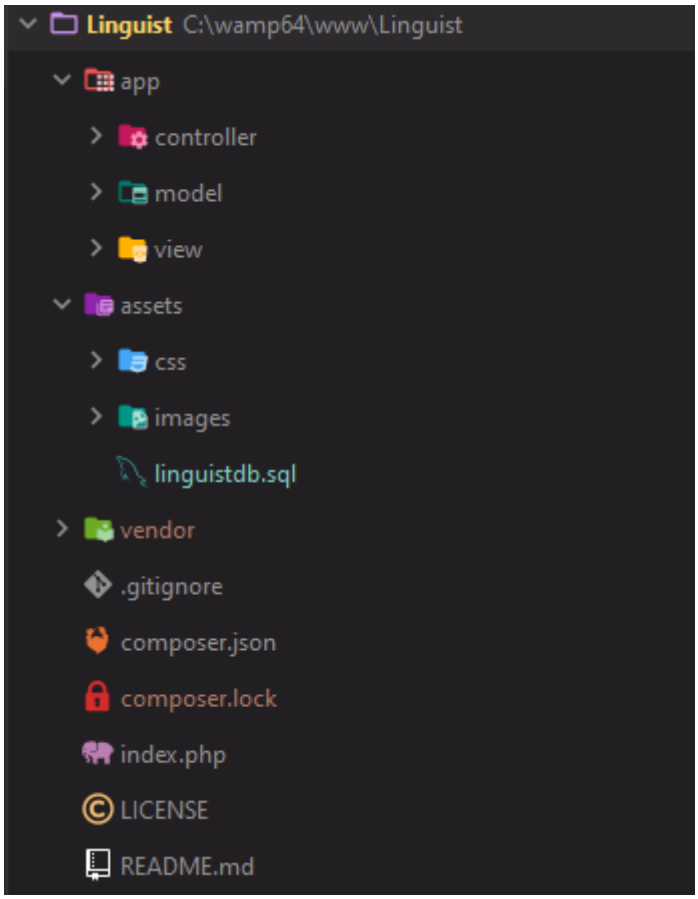
Une architecture dite « Modèle-Vue-Contrôleur » (MVC) sera à appliquer, et devra décomposer le site en parties bien distinctes.

Le langage de programmation orientée objet « PHP » devra être employé. Des méthodes et concepts se rapprochant de la Programmation orientée Objet devront construire l'application.

Les données seront stockées au sein d'un Système de Gestion de Bases de Données Relationnelles (SGBDR). Le choix fût porté sur MySQL en dernière instance.

Développement de l'application

Arborescence



Le fichier « app » contient le cœur du code formant l'application. Comme indiqué précédemment, il suit le modèle MVC et à ce titre, comporte 3 fichiers du même nom au sein de ce répertoire.

Un second répertoire « assets » permet de regrouper l'ensemble des éléments statiques allant être intégrés au site web, à savoir les fiches de style (CSS), les images, ou encore le dump de la base de données, servant à mettre en place l'architecture de cette dernière.

L'application passe par « composer », un gestionnaire de dépendances adapté au langage PHP. Le fichier « vendor » contiendra donc l'ensemble des librairies nécessaires afin de faire fonctionner l'application convenablement.

Accès à la base de données

Afin de se connecter à l'instance MySQL, des « data sources » seront fournies à une classe d'accès aux données (DAO).

```php
class DAO
{
    protected static $dbh;
    private static $host = "localhost";
    private static $db = "linguist";
    private static $user = "root";
    private static $passwrd = "root";

    public static function PDOconnect(): PDO
    {
        try {
            self::$dbh = new PDO( dsn: "mysql:host=" . self::$host . ";dbname=" . self::$db, self::$user, self::$passwrd,
                array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8')
            );
            self::$dbh->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
            return self::$dbh;
        } catch (PDOException $PDOerror) {
            print "An error has occurred while trying to access PDO: " . $PDOerror->getMessage();
            die();
        }
    }
}
```

On fournit donc l'ensemble des informations nécessaires à l'application afin que cette dernière puisse se connecter au serveur, puis à la base de données, et ce via l'utilisateur souhaité. Tout cela afin que l'application puisse effectuer l'ensemble des opérations souhaitées au sein de la base de données.

Modèles

Un modèle est dédié à chaque table, afin que son contenu puisse être stocké dans un objet. Voici un exemple de la classe dédiée aux leçons :

```php
class Lessons
{
    private $lessonId;
    private $name;
    private $wording;
    private $content;
    private $illustrationPath;
    private $languageId;

    public function __construct($lessonId, $name, $wording, $content, $illustrationPath, $languageId)
    {
        $this->lessonId = $lessonId;
        $this->name = $name;
        $this->wording = $wording;
        $this->content = $content;
        $this->illustrationPath = $illustrationPath;
        $this->languageId = $languageId;
    }

    public function getLessonId()
    {
        return $this->lessonId;
    }

    public function getName()
    {
        return $this->name;
    }
```

Le constructeur (première fonction) permettra d'instancier l'objet souhaité à partir des données récupérées depuis la base. On passe ensuite les valeurs dans l'ordre souhaité afin que l'objet reçoive les valeurs attendues.

Une classe d'accès aux données dédiée à chaque Classe sera nécessaire, par soucis d'organisation.

Exemple de méthodes pour la DAO dédiée aux leçons :

```php
public function getLessonById($lessonId)
{
    try {
        DAO::PDOconnect();
        $query = self::$dbh->prepare("SELECT * FROM lessons WHERE lessonId = :lessonId");
        $query->bindValue(':lessonId', $lessonId);
        $query->execute();
        $lesson = $query->fetch(PDO::FETCH_ASSOC);
        $thisLesson = new Lessons(
            $lesson["lessonId"],
            $lesson["name"],
            $lesson["wording"],
            $lesson["content"],
            $lesson["illustrationPath"],
            $lesson["languageId"],
        );
    } catch (PDOException $error) {
        print "An error occured: " . $error->getMessage();
        die();
    }
    return $thisLesson;
}
```

La fonction illustrée dans cet exemple permet de récupérer une leçon via son identifiant unique. Cet identifiant, passé en paramètre de la fonction, sera ensuite précisée (bind) au sein de la fonction SQL, afin d'effectuer une requête classique nous donnant la leçon voulue. Cette leçon étant unique, un tableau d'objets n'est pas nécessaire.

Le terme « New » permet d'instancier l'objet, on passe ensuite l'ensemble des données collectées au constructeur, qui va attribuer les données dans l'ordre indiqué.

Contrôleurs

Les contrôleurs vont permettre de traiter les objets instanciés depuis les modèles. Prenons comme exemple un contrôleur associé à la partie du site web gérant l'affichage des leçons :

```
public static function thisLessonDisplay()
{
    if (!Authentication::isLoggedOn()) {
        header( header: "Location: ?action=login");
    }
    $thisLesson = LessonsDAO::getLessonById($_GET["lessId"]);
    IndexController::basicAssets( header: false,  pageTitle: $thisLesson->getName() . " - Linguist");
    include "app/view/viewthislesson.php";
}
```

La fonction prend en charge l'affichage de la leçon souhaitée.

Quand une leçon est consultée, son identifiant est placé dans l'URL du site. On appelle donc la fonction présentée dans la partie dédiée aux modèles en plaçant comme paramètre cet Identifiant.

La leçon, sous forme d'objet, sera alors stocké dans la variable « thisLesson ». On appelle ensuite la vue et les paramètres adéquats afin de gérer au mieux l'affichage de la leçon, en y incluant le nouvel objet contenant la totalité de la leçon.

Vues

La vue est la partie relativement statique du site, admettant une structure prédéfinie d'une page et prenant à sa charge l'esthétique et l'apparence d'un site. Il est cependant possible de passer à la vue (depuis le contrôleur) des objets et variables allant influencer son contenu. Voici un exemple pour la page permettant de lister l'ensemble des leçons :

```
<?php if (empty($lessons)) { ?>
    <p class="emptySearch">No result found.</p>
<?php } else {
    foreach ($lessons as $k => $v) { ?>
        <style>
            <?php echo "#cardimg".$v->getLessonId()?>
            {
                background-image: url("<?php echo $v->getIllustrationPath()?>")
            ;
            }
        </style>
        <div class="card">
            <?php echo "<div class='cardimg' id='cardimg" . $v->getLessonId() . "'></div>" ?>
            <div class="cardcontent">
                <div class="cardwrap">
                    <h1 class="cardtitle"><?= $v->getName(); ?></h1>
                    <div class="cardwording"><?= $v->getWording(); ?></div>
                </div>
                <div class="cardbutton">
                    <a href='./?action=openlesson&lessId=<?= $v->getLessonId() ?>'>Check lesson </a>
                </div>
            </div>
        </div>
    <?php } ?>
<?php } ?>
```

On remarque que, dans cet exemple, la structure et le style de la page est déjà défini de façon statique. Cependant, il inclut une boucle qui permettra d'agencer l'ensemble des objets contenus dans le tableau qui est passé à la vue.

Résultat :