|  |
| --- |
| Documentation Développeur  Du Projet LPRS |

|  |
| --- |
| Par Amine NAKHIL, Loïc Guo et Quentin Lignani |



1. Inscription/Connexion
2. Page « Présentation du Lycée »
3. Page « Evénements »
4. Page « Offres d’emplois »
5. Profil

Interactions avec les autres utilisateurs

1.Page inscription.php



**On commence cette page internet par un code php au sommet du document. Il ne contient qu’une instruction : celle écrite en haut. Elle est écrite pour démarrer une session, ou même restaurer la session précédente via une requête passée GET ou POST ou restaurer une session communiquée par un cookie.**



**Cette capture est le commencement d’une syntaxe routinière des pages du site internet. Toutes les pages html du site sont introduites par le code ci-dessus. Dans la balise de l’en tête de la page, on importe les feuilles de style en CSS qui permettent le design mais aussi d’uniformiser ce même design pour toute les pages du site internet. Notre page de design CSS principale est la page « style.css ». Les autres pages CSS sont là principalement par norme, par exemple la page « bootstrap.min.css ». Dans la balise « title », on informe le titre de l’onglet que l’on veut mettre, en l’occurrence « UFA Robert Schuman | Inscription ».**



**A** **travers l’ouverture des div, on élabore la barre de navigation. Avec Bootsrap, on place les colonnes, on ouvre un div dans lequel on y met le logo, puis on met le texte « établissement catholique sous contrat d’association avec l’état, établissement habilité à recevoir la taxe d’apprentissage ». On met ensuite un autre logo de l’établissement.**



**On place les colonnes avec Bootstrap, on le fait pour inclure dans la barre de navigation deux boutons collés l’un à l’autre. Sur l’un est écrit « Se connecter » et redirige vers la page de connexion., et sur l’autre est écrit « S’inscrire » et ne redirige vers rien car nous sommes déjà sur la page d’inscription. Sur la forme, il s’agit d’un seul bouton composé d’une liste de deux tabulatrices.**



**On institue un titre sous le format H3 pour y afficher « S’inscrire ».**















2.Page cible\_inscription.php





3.Page confirmer\_inscription.php



**Cette capture est le commencement d’une syntaxe routinière des pages du site internet. Toutes les pages html du site sont introduites par le code ci-dessus. Dans la balise de l’en tête de la page, on importe les feuilles de style en CSS qui permettent le design mais aussi d’uniformiser ce même design pour toute les pages du site internet. Notre page de design CSS principale est la page « style.css ». Les autres pages CSS sont là principalement par norme, par exemple la page « bootstrap.min.css ». Dans la balise « title », on informe le titre de l’onglet que l’on veut mettre, en l’occurrence « UFA Robert Schuman | Inscription ».**











4.Page connexion.php



**On commence cette page internet par un code php au sommet du document. Il ne contient qu’une instruction : celle écrite en haut. Elle est écrite pour démarrer une session, ou même restaurer la session précédente via une requête passée GET ou POST ou restaurer une session communiquée par un cookie.**

****

**Cette capture est le commencement d’une syntaxe routinière des pages du site internet. Toutes les pages html du site sont introduites par le code ci-dessus. Dans la balise de l’en tête de la page, on importe les feuilles de style en CSS qui permettent le design mais aussi d’uniformiser ce même design pour toute les pages du site internet. Notre page de design CSS principale est la page « style.css ». Les autres pages CSS sont là principalement par norme, par exemple la page « bootstrap.min.css ». Dans la balise « title », on informe le titre de l’onglet que l’on veut mettre, en l’occurrence « UFA Robert Schuman | Connexion ».**















5.Page Manager\_User.php





**Comme cette page sert à faire le pont en notre page internet et notre base de données, nous mettons des variables avec le même nom que ce que l’on doit avoir à traiter comme informations. On commence ce code en ouvrant la classe et en la nommant « Manager\_User », n’utiliser qu’une seule classe par fichier, et en nommant le fichier et la classe avec le même nom. Tout ceci dans le plus strict respect des conventions. On fonde quatre attributs à savoir « nom », « prénom », « email » et « mdp ». Ces attributs sont en privé, ils ne sont pas accessibles en dehors de la présente classe.**



**On ouvre une méthode en public, on la nomme « inscription », elle sert donc à l’inscription. On y met une variable interne nommée « inscrit ». Dans cette méthode, on appelle la classe « User », qui représente le model. Dans cette méthode, on utilise l’interface PHP Data Objects (PDO). Dans la première ligne de l’utilisation de PDO, on instancie un nouvel objet : PDO, on informe que l’on travaille en local, le nom de la base de données, en l’occurrence « projet\_lycee », puis on communique les identifiants avec lesquels on interagit avec la base de données, ici avec le nom d’utilisateur « root » et avec un mot de passe blanc. Au sein de la ligne suivante, on prépare la requête qui va interroger la base de données dans laquelle on communique la requête en langage SQL. Notre commande permet d’afficher dans la table « utilisateur » toutes les lignes parmi lesquelles la variable « email » est semblable à la colonne « email » de la base. L’avant dernière étape de cette utilisation de PDO est d’exécuter la commande dans un tableau, on demande d’obtenir ce que contient la ligne dont « email » est semblable, on obtient donc l’adresse email. Pour que cette commande préalablement peaufinée soit appliquée, il faut utiliser la commande « fetch » à la dernière ligne. Ainsi, la variable « donnee » doit contenir le contenu de la colonne « email » à la ligne ou l’adresse mail saisie est similaire à elle dans le cas où elle existe.**



**Ensuite, on met une condition. Comme dans la dernière ligne, la variable « donne » est censée contenir l’adresse mail saisie par l’utilisateur si et seulement si elle existe. Ainsi, cette condition permet de vérifier si la variable « donnee » est remplie donc à fortiori si l’adresse mail entrée est déjà associée à un compte. Si cette condition est vérifiée, on affiche sur la page html « L’email est déjà utilisée ». Après cela, on redirige vers la page « inscription.php » pour retenter une connexion.**



**Ce code est une condition inverse, ainsi, il ne s’applique uniquement dans le cas ou la condition précédente n’est pas vérifiée. En clair, cette condition s’applique si l’addresse mail entrée par l’uttilisateur n’existe pas déjà. Cette condition sers à l’envoi de mail, on y uttilise l’extension PHPMailer. On importe donc les modules qui sont uttilisés dans ce cadre à travers des balises « require » qui pointent vers des fichiers par défault de PhpMailer**.



**Ce code est issu de l’extension « PhpMailer », il sert à renseigner les informations que l’extension exige pour fonctionner. On encode l’affichage en UTF-8 pour faire supporter notamment les caractères spéciaux, on active le protocole SMTP qu’utilise PHPMailer pour envoyer des mails, on active l’authentification , on utilise le transfert sécurisé avec SSL, on renseigne le domaine de mails qui est supporté par notre fournisseur de mails, on renseigne le port que l’on veut utiliser pour envoyer des mails, on renseigne l’adresse mail d’expédition, on renseigne le mot de passe lié à cette adresse, on met dans l’objet du mail que l’on envoie « Création de votre compte réussie », on met le contenu du mail à travers une mise en forme.**



**Ceci est une condition. Cette condition ne s’applique que si le mail n’a pas été envoyé. Comme nos paramètres sont inchangés, cela ne peut que provenir d’une adresse mail invalide. Ainsi, on utilise un message d’erreur pour prévenir l’utilisateur de sa saisie invalide de l’adresse mail. On affiche donc « Erreur dans l’email ». On renvoie vers la page d’inscription pour une nouvelle tentative.**



**Ce code est une condition, il ne s’applique que si la condition précédente n’est pas vérifiée. En clair, ce code ne s’applique que l’adresse mail saisie est valide. On commence par informer l’utilisateur que l’envoi est réussi en affichant « message has been sent ». Ensuite, on utilise PHP Data Object pour interagir avec la base de données. On commence par utiliser la fonction « prepare » comme la syntaxe de PDO l’exige, on y affiche notre commande SQL à savoir « INSERT into utilisateur (nom, prenom, email, mdp) value (?,?,?,?) », cette commande permet d’aller dans la table « utilisateur », et de remplir les colonnes « nom », « prenom », « email », « mdp », avec les valeurs respectives que l’on renseignera après. A la ligne de code suivante, ont émet un tableau associatif dans lequel on relie le getter de chaque attribut renseigné dans le model, avec les valeurs fournies par l’utilisateur. Il est a noté que l’on hashe préalablement « mdp » avec l’algorithme SHA1 avent de l’inscrire dans la base de données.**



















6.Page User.php



**Cette page php est un model, on l’utilise pour faire le pont entre la base de données et les informations entrées dans les formulaires par l’utilisateur. On ouvre la balise php, ou ouvre une classe nommée « User » et on nomme le fichier « User.php » pour que le fichier ait le même nom que la classe. Tout ceci pour appliquer les conventions en vigueur. Ensuite, on déclare nos attributs : « nom », « prenom », « email » et « mdp ».**







6.Page Projet\_lycee.sql

**Ce fichier héberge la base de données à importer pour pouvoir utiliser le site correctement. Il faut aller à la racine du projet, puis dans le dossiers « bdd », et on trouve donc le fichier Projet\_lycee.sql**





**Commençons cette base de données par la première instruction que l’on peut observer, elle vise à supprimer la table « annonces » de la base de données dans le cas ou elle existerait. En effet, si l’on importe la base de données déjà existante, il faut pouvoir s’assurer de bien mettre à jour les tables. La seconde commande, plus longue, vise justement à ouvrir la table « annonces », si elle n’existe pas déjà. P ar le biais de la même commande, on insère cinq colonnes dans cette base de données, à savoir « id » de type entier à onze caractères, « id\_utilisateur », de type entier à dix caractères, « titre » de type varchar à quarante caractères, « description », de type text et « date », de type date. Toutes ces colonnes sont en NOT NULL. Par convention pour les cases « id », on lui donne l’attribut UNSIGNED car nous sommes surs d’avoir des valeurs positives. On ajoute une clef étrangère à la colonne « id\_utilisateur », car cette colonne doit être relié à une autre table pour connaitre l’utilisateur qui est derrière l’annonce, cette clef étrangère est ainsi liée au nom de cette personne.**



**Dans l’éventualité où une table « discussion » existe déjà, on demande de la supprimer à travers la première instruction, et ce pour les mêmes raisons que dit plus haut. On ouvre une table « discussion », si elle n’existe pas. On insère trois colonnes dans cette table, la colonne « id » de type entier avec onze caractères, la colonne « id\_user1 » de type entier à dix caractères et la colonne « id\_user2 » de type entier à dix caractères. On identifie la colonne « id » avec une clef primaire. Pour retrouver l’utilisateur derrière les id, les colonnes « id\_user1 » et « id\_user2 » sont ajoutés dans des clefs primaires. Toutes les colonnes sont en NOT NULL.**



**Cette liste d’instructions sert à instituer la table « evenements » de la base de données. La première ligne de code sert à supprimer la table « evenements » dans le cas où elle existerait déjà. La seconde instruction demande d’ouvrir une table « evenements » dans le cas où elle n’existerait pas déjà. On demande les colonnes suivantes dans la table « evenements » : « id » de type entier à onze attributs, « id\_utilisateur » de type entier à dix attributs et en UNSIGNED, « titre » de type VARCHAR à 40 caractères, « description » de type text, « date » de type date et « Comm » de type text. Toutes les colonnes sont en NOT NULL et doivent donc être pleines. Les colonnes « titre », « description » et « Comm » sont encodées en UTF-8, elles supportent ainsi les caractères spéciaux.**



**Ce bloc d’instructions sert à insérer des données dans la table évènements. Concrètement, l’objectif est d’avoir des publications par défaut dans sur la page « Evenements ». La première instruction insère dans la table « evenements pour les colonnes « id », « id\_utilisateur », « titre », « description », « date » et « Comm les valeurs respectives 1, 3, « AMINE », « NAKHIL », « 2020-10-05 » et « TEST ». On ajoute une autre ligne dans laquelle on insère ces valeurs pour les mêmes colonnes : « 2 », « 3 », « Développeur Informatique Fullstack H/F », « À propos d'AtosAtos est un leader international de la transformation digitale avec plus de 110 000 collaborateurs dans 73 pays et un chiffre d'affaires annuel de plus de 11 milliards d'euros. Numéro un européen du Cloud, de la cybersécurité et des supercalculateurs. », « 2020-10-21 », et « AGEFIPH- Espace Emploi ». Pour clouer cette instruction, on ajoute une ultime ligne à la table avec les valeurs respectives : « 4 », « 3 », « stage Developpeur informatique - H/F », « Qui sommes-nous ? Avec 90 000 collaborateurs présents sur les cinq continents, SUEZ est un leader mondial dans la gestion intelligente et durable des ressources. Le Groupe fournit des solutions de gestion de l'eau net des déchets qui permettent aux villes. », « 2020-06-09 » et « Jobijoba ».**



**On supprime la table « messages » dans l’éventualité ou elle existe, puis on ouvre une du même nom dans l’éventualité ou elle n’existe pas. Tout cela pour les mêmes raisons que pour les précédentes requêtes. On fonde cinq colonnes : « id » de type entier avec onze caractères, « id\_discussion » de type entier avec onze caractères, « id\_utilisateur de type entier avec dix caractères, « message » de type text, « date » de type datetime. Toutes les colonnes sont en NOT NULL. « id\_uttilisateur » est en UNSIGNED, en est sûr que ce sera positif. On utilise une clef primaire avec la colonne « id » pour relier cette colonne à l’expéditeur ou au destinataire des messages.**



**Dans la première instruction, on supprime la table « utilisateur » dans le cas où elle n’existe pas, puis on fonde une nouvelle table « utilisateur » dans le cas où elle n’existe pas déjà. On fonde les colonnes « id » de type entier à dix caractères, « nom » de type varchar à 40 caractères, « email » de type varchar à 40 caractères, « mdp », de type varchar à 40 caractères, « role », de type varchar à 10 caractères, « date\_connexion » de type date, « verif » de type tinyint à un caractère. Toutes les colonnes sont en NOT NULL à part « date » et « rôle ». Verif est par défaut de « 1 » et en UNSIGNED. On utilise une clef primaire pour la colonne « id »**



**Cette instruction sert à insérer des utilisateurs par défaut de sorte à ce qu’il n’y ait pas aucuns utilisateurs lorsque l’on commence l’utilisation du site.Il n’y a qu’une seule instruction, elle vise à ajouter des valeurs dans la table « utilisateurs ». On alimente les colonnes suivantes « id », « nom », « prenom », « email », « mdp », « role », « date\_connexion » et « verif ». On insère les valeurs suivantes dans l’odre respectif des colonnes précédemment citées : « 1 », « a », « a », « a@a », « 86f7e437faa5a7fce15d1ddcb9eaeaea377667b8 », [on n’affecte pas de données à cette colonne au sein de cette ligne], [on n’affecte pas de données à cette colonne au sein de cette ligne] et 1. On ajoute la ligne suivante dans le même ordre : 2, « b », « b », « b@b », « e9d71f5ee7c92d6dc9e92ffdad17b8bd49418f98 », [on n’affecte pas de données à cette colonne au sein de cette ligne], [on n’affecte pas de données à cette colonne au sein de cette ligne] et 1. Dans même ordre, on ajoute une ligne avec les valeurs suivantes : 3, « Nakhil », « Amine », « nakhila@orange.fr », « 265545635d1704996ba4dbb482377aa542cdf5c0 », [on n’affecte pas de données à cette colonne au sein de cette ligne], [on n’affecte pas de données à cette colonne au sein de cette ligne] et 1.**







