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

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

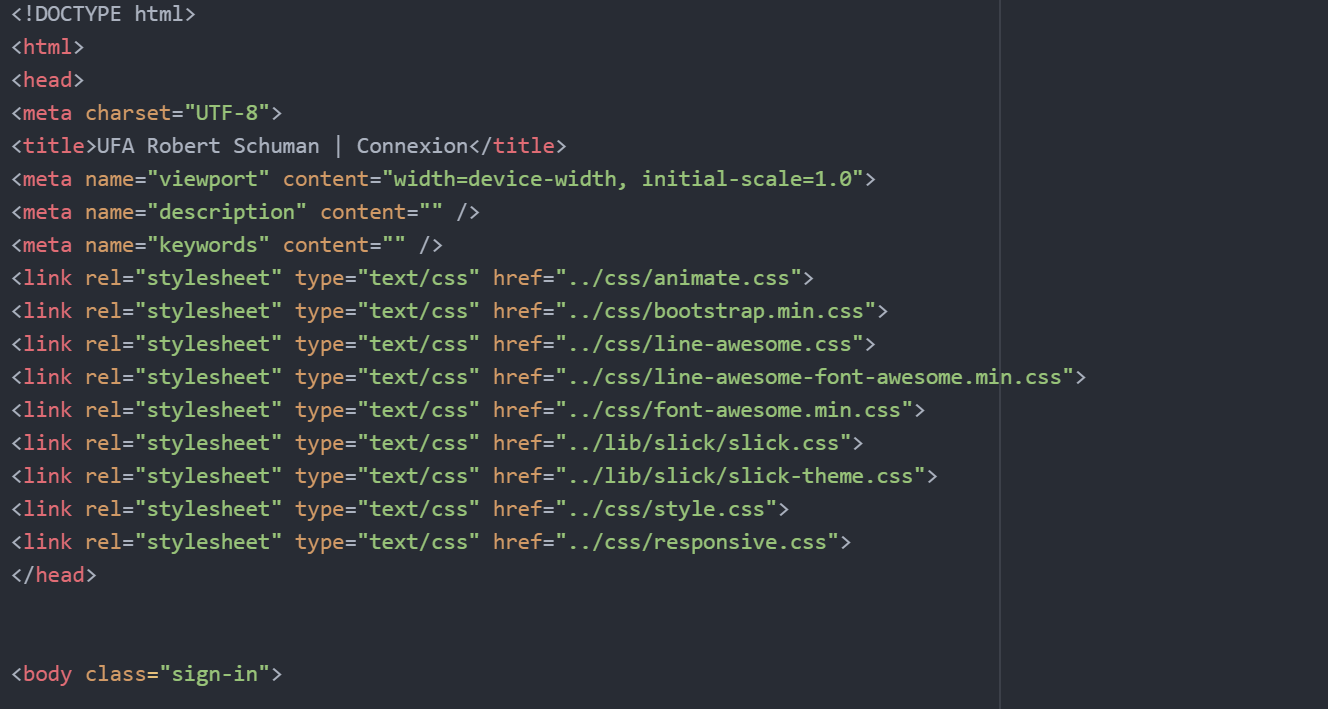


1. Inscription/Connexion
2. Page « Présentation du Lycée »
3. Page « Evénements »
4. Page « Offres d’emplois »
5. Profil

Interactions avec les autres utilisateurs

1.Page inscription.php





















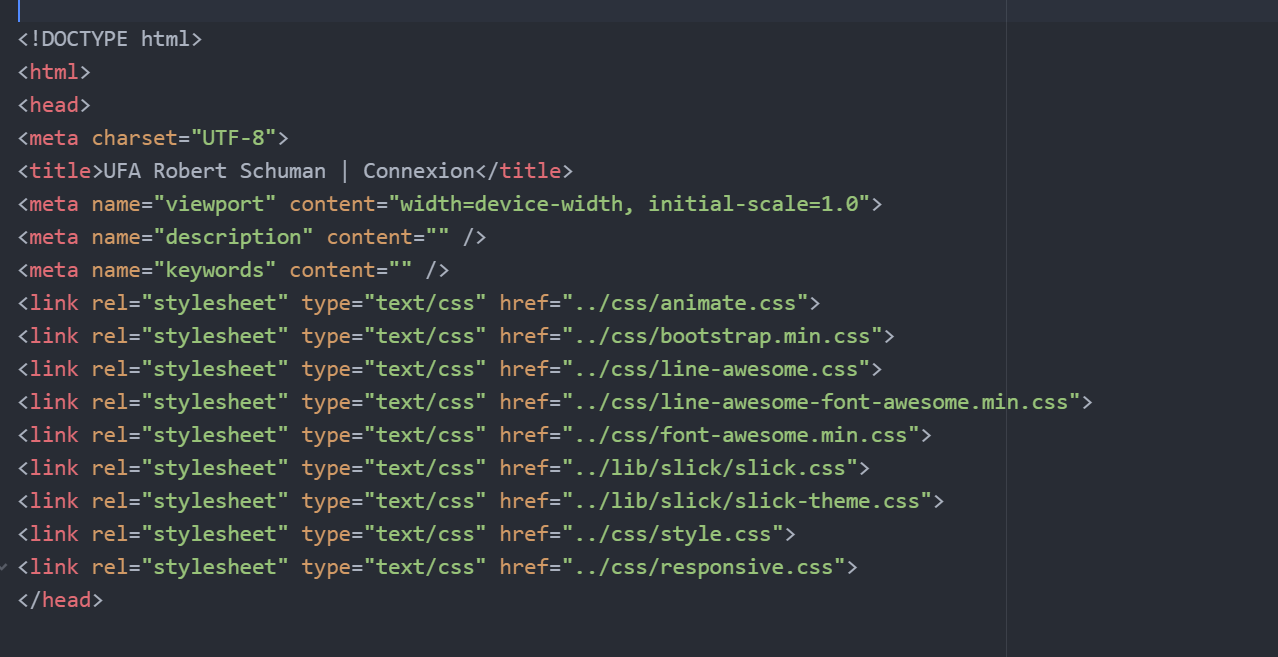


2.Page cible\_inscription.php





3.Page confirmer\_inscription.php













4.Page connexion.php



















5.Page Manager\_User.php



































6.Page User.php









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**







**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 »**









