Sommaire:

1. Introduction
2. Description du projet
3. Solutions de mise en œuvre
   1. Conception
   2. Développement
      1. base de données
      2. choix de design
      3. partie questionneur
      4. partie répondeur
4. Problèmes rencontrés et leurs résolutions
5. Les algorithmes : solutions techniques
6. Synthèse des comptes rendus des points de rendez-vous
7. Conclusion
8. Glossaire
9. Annexes
10. Introduction

Nous sommes une équipe de cinq étudiants en deuxième année de DUT Informatique à l’Université Paris 13. Dans le cadre de notre troisième semestre, nous avons été amenés à choisir un projet tuteuré parmi ceux proposés par les différents enseignants. Nous avons opté pour le projet proposé par Mme Gayral, la gestion de QCM. En effet, le sujet nous donnait déjà une idée de la conception possible du site, et de la manière de l’implémenter.

De plus, le développement s’appuyant sur des langages de programmation que nous maîtrisons, nous avons choisi ce projet car il nous permettait d’améliorer nos compétences dans les langages de programmation abordés en cours.

1. Description du projet

Notre projet se présente sous la forme d’un logiciel web permettant de créer et de répondre à des QCM. Il est constitué de deux interfaces, prévues pour deux sortes d’utilisateurs différentes: le questionneur qui crée les QCM, et le répondeur qui répond aux QCM créés par les questionneurs. Ainsi un élève répondeur a accès à une interface « répondeur » alors qu’un enseignant questionneur a accès à l’interface « questionneur ».

Les langages à utiliser étaient imposés par le client : notre projet s’articule donc autour de deux interfaces implémentées en html et css pour le développement front-end\*, ainsi qu’en php avec l’extension PDO\* pour le développement back-end\*. Il s’appuie sur une base de données sous PostgreSQL.

Bien que le sujet fasse mention d’un développement en Javascript, à la demande de Mme Gayral ce langage n’a pas été utilisé car il n’a pas été abordé en cours.

1. Solutions de mise en œuvre
   1. Conception

A la suite de l’affectation finale de nos projets, nous avons tout d’abord défini au mieux notre travail. Cette première phase a constitué l’étape initiale de la conception, avec le recensement des différentes tâches à effectuer. Les langages de programmation étant imposés, il seulement fallu choisir les outils adaptés à leur mise en œuvre.

Une fois les tâches écrites sous forme de user stories\*, classées par ordre d’importance et leur chronologie déterminée (voir annexe Trello), nous avons réparti le travail entre les membres du groupe. Il a été décidé que le développement front-end serait assuré par Célia Aït Mammar, la création et la gestion de la base de données par Nathan Gatto, et le développement back-end par Lucie Godefert, Florian Kotecki et Charlie Didier.

Le site web est structuré suivant un schéma défini préalablement par une arborescence regroupant toutes les pages créées (voir annexe 3). Cette arborescence nous a servi pour répartir l’implémentation des pages entre les développeurs back-end.

* 1. Développement

Comme dit plus haut, l’implémentation a été scindée en deux parties, le développement de l’interface questionneur et le développement de l’interface répondeur, à partir de structures codées en html créées par le développeur front-end.

1. Base de données: conception et implémentation

Auteur : Nathan

Pour concevoir la base de données, j’ai utilisé le logiciel de SGBD PostgreSQL et pgadmin 3 pour exécuter les scripts et faire des tests. Tout d’abord, l’ensemble du groupe a discuté sur la structure de la base de données ainsi qu’à son contenu. Puis j’ai créé les scripts pour initialiser les tables et pour les supprimer en accord avec ce qui avait été décidé. Le script d’insertion est venu un peu plus tard car le logiciel PhpPgAdmin permet l’insertion simple et rapide de données dans une table mais cela ne convenait pas dès qu’il fallait insérer beaucoup de données dans la base de données.

Dès le premier essai, nous avons mis en place les relations principales comme qcm, question, réponse, utilisateur (qui regroupait répondeur et questionneur au départ) et domaines. Puis au fur et à mesure que l’on avançait dans le projet, la relation utilisateur devait nécessairement être séparé en 2 parties distinctes. De plus, la cliente a demandé la mise en place de sous-domaine pour spécifier un peu mieux les divers qcm disponibles.

Pour la structure, la première version était fonctionnelle mais pas très optimisé pour être utilisée dans le php. De plus, pour faciliter son intégration dans le php, certaines choses ont dû être modifiées comme le typage de la date ou celle du temps. Par la suite, j’ai donné à titre d’exemple certaines requêtes préparées.

La seconde version a apporté une meilleure stabilité et un changement majeur puisqu’elle signe la création d’un rôle limité pour l’utilisation du site web de QCM, là où auparavant le superadmin était utilisé pour effectuer les requêtes préparées. A la suite de ceci, j’ai ajouté et supprimé certaines permissions pour éviter les erreurs lors de l’accès à une page web contenant un lien avec la base de données.

La troisième version nous a permis de stocker les statistiques d’un répondeur pour chaque qcm qu’il a effectué (le répondeur peut bien entendu refaire le même qcm vu que la date est retenue comme clef primaire de cette table). Ceci est venu aider la création d’un profil propre aux répondeurs.

J’ai aussi basculé le domaine et sous domaine de qcm\_question vers qcm car il était illogique de conserver ces attributs dans la relation faisant le lien entre qcm\_question. A la base, la cliente souhaitait pouvoir utiliser une question dans plusieurs qcm de domaines différents d’où le nom de domaine et de sous-domaine dans la relation qcm\_question. Puis, pour faciliter le développement du site, cette spécificité a été abandonnée et les attributs ont donc été remis dans la relation qcm.

Pour faciliter les requêtes sql dans le code php, la base de données comporte quelques redondances notamment pour le nom du domaine et du sous-domaine afin d’éviter de surcharger $\_POST ou $\_GET.

La quatrième et dernière version en date ajoute peu de contenu par rapport aux précédentes versions mais se concentre essentiellement sur les contraintes d’unicité et les problèmes liés aux données pouvant être facultatives

1. Choix de conception du design : html/css

Auteur : Célia

Avant de commencer à imaginer l’apparence du site, nous avons réfléchi, en équipe, à la structure de celui-ci. Pour cela nous avons d’abord schématisé une arborescence qui nous a permis d’avoir une vision globale du site et de définir les chemins entre les différentes pages.

L’arborescence a connu des changements de structure au fil des semaines. Les changements sont dûs à l’ajout de nouvelles fonctionnalités pensées au cours du projet. Certaines de ces fonctionnalités ont impliqué la création d’une nouvelle page (ex : Modifier la visibilité). D’autres ont seulement eu besoin d’un lien supplémentaire (ex: Profil et Question) Vous pouvez maintenant voir le rendu final avec sa légende à l’annexe Arborescence (1 et 2) ou dans le fichier nommé Arborescence.

Pour concevoir le corps d’une page, j’ai d’abord discuté avec celui ou celle qui était chargé(e) du code php. Nous nous sommes mis d’accord sur la structure de la page ainsi que sa composition, afin que je puisse lui fournir un corps html et un css ‘test’. Lorsque le code php a été fini et fonctionnel, j’ai actualisé le css que j’ai dû finir de perfectionner entre temps (couleur, formes, ombre, contour, animation, police, harmonisation…). Il y a donc (quasiment) eu à chaque page deux versions de la feuille css.

L’élément inchangeable et indétrônable du site a été la barre de navigation. Simple mais pourtant très efficace d’un point de vue ergonomique. Elle a permis la fluidité du site (passage d’un onglet à un autre sans devoir retourner à la racine).

Passer derrière le code achevé de mes équipiers (avec la combinaison des miens par la suite) m’a permis de régler des petits problèmes d’affichage ou de repérer des problèmes dans les codes php. C’est ainsi que l’on a pu optimiser notre temps de travail et éviter d’empiéter sur le travail des autres.

Certes être celle qui repasse sur un code php pour intégrer le nouvelle class ou id en html et insérer le css peut sembler être une perte de temps. Mais cela permettait à mes camarades de se focaliser sur leur partie et de mon côté m’obliger à lire et vérifier soigneusement leur code. C’est une stratégie que j’ai opté et dont j’assumerais la responsabilité.

Une fois après avoir parlé avec mes camarades chargés du code php et de leur avoir fourni un corps test avec seulement ce qui m’a semblé être nécessaire. Je me suis mis à imaginer plus en profondeur l’apparence de chacune des pages. J’ai développé en html et css comme il a été imposé, par le biais de ces langages. Voici les différents designs qui ont été retenu.

Index – Accueil Répondeur – Accueil Question

J’ai choisi de créer une page d’accueil dans un style minimaliste pour qu’elle puisse convenir au plus grand nombre. On retrouve tous les éléments que doit contenir une page d’accueil de manière claire et synthétique. C’est aussi une stratégie pour que l’utilisateur se retrouve facilement et que l’ensemble du site réponde à la règle des trois clics.

Choix Domaine – Choix Sous-Domaine – Choix QCM – Choix

Le code php liste les domaines, sous-domaines et qcm avec une boucle. Il était donc impossible de personnaliser les différents domaines entre eux sans JavaScript, pareil pour sous-domaine et qcm. Définir 3 couleurs différentes pour chacune des étapes cassait le cheminement d’après moi. J’ai choisi de différencier les couleurs de questionneur et répondeur avec des dégradés de couleur pour chacun.

Exécuter – Statistique – VisualisationQCM – Question

Ces pages représentent ce que verrons le plus les utilisateurs : l’affichage du QCM. Il a été décidé que l’utilisateur questionneur et répondeur ont le même affichage. Ainsi le questionneur pourra savoir à quoi ressemble son QCM aux yeux d’un utilisateur répondeur. Alors que l’utilisateur répondeur n’a aucune distinction entre les cases à cocher, l’utilisateur questionneur différencie la bonne réponse (ou les bonnes réponses) des mauvaises.

Exécuter - Supprimer - Statistique - Profil

La particularité de ces pages est qu’elles regroupent plusieurs fonctionnalité en une page. Ce qui veut dire que ces pages ont plus qu’une seule apparence. En fonction de la condition dans laquelle on se trouve, j’ai du prévoir et créer une autre apparence qui donne l’impression parfois d’être sur une autre page, parfois il s’agit juste d’une continuité.

1. Partie Questionneur

Auteur : Florian

Les logiciels libres ont été mes principaux outils pour réaliser ce développement back-end qui consiste en l'implémentation de la partie questionneur du site, avec l’aide de Charlie.

Le php a été implémenté à l’aide de l’éditeur de texte notepad++, et partagé grâce au logiciel github desktop mettant à jour le répertoire commun très rapidement.

Entre la réalisation et le partage, les tests locaux ont été réalisés grâce à l'application phppgadmin et au système de gestion de base de donnée PostgreSQL, tous deux installés sur l’application Wamp permettant la fusion de la base de donnée au serveur php, tous les deux locaux.

La partie questionneur du site est composée de 11 pages dont Charlie et moi allons faire la description, et ce dans l’ordre d’apparition des scripts lors de l’exploration du questionneur à travers le site (en partant de l'accueil réservé au questionneur).

Cette exploration est divisée en deux parties, dont la première que nous allons présenter est la partie servant à créer un qcm, la deuxième étant celle où le questionneur pourra consulter son profil et interagir avec ses qcm créés.

Ayant pour but de compacter certaines fonctionnalités, et ainsi de réduire le nombre de scripts total, certains scripts affichent plusieurs pages au questionneur lors de son utilisation; pages que nous décrirons une par une. Ces différentes pages sont séparées entre elles dans le code par des formulaires, et communiquent via les données nécessaires envoyées par ces formulaires. Cette particularité ne procure rien de gênant pour l’utilisateur si ce n’est d’utiliser le même script tout en affichant des pages différentes.

ChoixQC.php :

Le script ChoixQC.php est le premier script que le questionneur utilisera si son intention est de créer un qcm. Ce script contient 3 pages guidant l'utilisateur à choisir le domaine et le sous domaine du qcm qu'il souhaite créer.

La première page affiche tous les domaines existants dans la base de données, proposant

alors à l'utilisateur d'en choisir un en cliquant dessus avant de passer à la page suivante. La page propose aussi un bouton permettant de créer un domaine si aucun ne convient au questionneur, envoyant alors sur le script CreerDomaine.php.

Le domaine sélectionné, Cette deuxième page affiche alors tous les sous-domaine de la base de données inclus dans ce domaine, en ajoutant un choix supplémentaire de sous domaine nommé « général » mais non inclus dans la base de donnée, proposant à l'utilisateur de ne pas relier son futur qcm à un sous domaine. Cette page propose aussi la création d'un sous domaine en envoyant également sur CreerDomaine.php.

Le sous domaine sélectionné (existant ou général), le qcm est alors créé dans la base de données en se voyant attribuer le domaine et le sous domaine sélectionné, le sous domaine du qcm étant nul si le questionneur a choisi « général ». La 3ème page affiche alors un message de confirmation de la création du qcm à l'utilisateur et propose un bouton qui envoie sur la page ChoixQQ.php afin de commencer la gestion de ce qcm fraîchement implémenté.

Auteur:Charlie

Pour réaliser les trois pages qui gèrent la création d’un QCM, CreerDomaine.php, CreationQuestions.php et Questions.php, j’ai utilisé l’éditeur de texte Nano intégré sur Linux, ainsi que le serveur Apache pour pouvoir interpréter le php ainsi que le SGBD PostgreSQL pour intégrer les requêtes de bases de données.

ChoixQQ.php:

Cette page affiche 2 boutons lorsque le questionneur commence la création de son qcm depuis ChoixQC.php. Le premier envoie sur CreationQuestions.php et l’autre sur Importer.php. Cette page ne sert qu’au début de la création du qcm, le reste de cette création est située autour de la page “centrale” Questions.php.

CreerDomaine.php:

Cette page gère la Création d’un domaine et d’un sous-domaine par le Questionneur. Elle s’affiche lorsque le Questionneur a cliqué sur le bouton “Créer domaine” de la page ChoixQC.php et affiche un champ de texte dans lequel le Questionneur entrera le nom du domaine qu’il souhaite créer. Ensuite il lui faudra cliquer sur le bouton “Créer domaine” pour sauvegarder son domaine dans la base de données et retourner vers ChoixQC.php.

Ensuite, si le Questionneur a souhaité créer un sous-domaine via ChoixQC.php, CreerDomaine.php s'affiche à nouveau et le Questionneur peut remplir le champ de texte avec le nom de son sous-domaine et cliquer sur “Créer sous-domaine” pour sauvegarder et insérer son sous-domaine dans la base de données.

Les boutons de retour de chaque formulaire sont utilisés pour retourner à la page ChoixQC.php si le Questionneur ne souhaite pas créer de domaine ou sous-domaine.

Questions.php:

Cette page gère l’affichage de la question nouvellement créée et sert de point central entre la validation d'un QCM, l’importation d’une question et la création d'une question.

Sur cette page est affiché en premier le Domaine et le sous-domaine (s’il y en a un) du QCM que le Questionneur est en train de créer. En-dessous est indiqué le numéro du QCM qui permet de savoir combien de QCM ont été créés précédemment.

Enfin, il y a 3 boutons :

* Le premier “Valider QCM” mène vers la page finaliser.php où le Questionneur pourra s'il le souhaite rendre visible son QCM.
* Le second bouton “Créer Question” conduit vers la page CreationQuestions.php où le Questionneur pourra créer une nouvelle question.
* Le troisième bouton “Importer Question” mène vers Importer.php où le Questionneur pourra importer une question déjà présente dans la base de données.

CreationQuestions.php:

Pour commencer, en arrivant sur la page, le questionneur doit remplir un champ de texte correspondant au nombre de réponses qu'il souhaite ajouter à sa question. Le questionneur doit donc rentrer un nombre supérieur ou égal à deux, car une question doit au moins avoir deux réponses possibles.

Une fois fait, ce champ de texte est retiré de la page et un formulaire complet est affiché, comprenant les différents champs de création de la question :

Dans le premier champ, le questionneur doit entrer l’intitulé de sa question. Dans la seconde partie du formulaire, les intitulés de réponses, plusieurs champs de texte sont affichés, le nombre de ces champs correspond au nombre de réponses que le questionneur à entré précédemment.

Ici, le questionneur doit indiquer l’intitulé de toutes les réponses qui seront proposées lors d'un QCM dans lequel cette question sera posée. À droite de ses champs de réponse, il y a une liste à deux choix, indiquant si la réponse correspondante est correcte ou non (il peut y avoir plusieurs réponses correctes).

Le champ suivant correspond au temps qu’un Répondeur devra mettre pour répondre à la question. Le Questionneur doit y entrer le temps en secondes.

Ensuite, une zone de texte permet au Questionneur d’indiquer des informations supplémentaires sur sa question, mais ce champ ne doit pas être obligatoirement complété. Le dernier champ est celui des points de la question. Le Questionneur y rentre le nombre de points qu'il souhaite sur sa question.

Une fois les champs remplis, le Questionneur clique sur le bouton de sauvegarde de la Question et sera redirigé vers la page Question.php.

Auteur:Florian

Importer.php:

Si le questionneur a choisi d’importer des questions déjà existantes dans la base de donnée, ce script aura la fonction de lui proposer un choix sous forme de boutons affichant des énoncés de question, lui permettant alors de choisir une question intéressante à importer parmi celles proposées.

Les questions proposées, grâce à une vérification dans la base de données à l’aide de l’id du qcm reçu de la page précédente, seront uniquement celles qui ont été reliées à des qcm du même domaine et sous domaine que le qcm dans lequel on importera ou non la question (uniquement le domaine si le sous-domaine du qcm est inexistant). De plus, les questions qui sont déjà reliées au qcm en cours de modification dans la base de données, seront ignorées.

Si l’utilisateur a choisi de ne plus importer de question, il lui est toujours possible de revenir à la gestion de son qcm grâce à un bouton retour. Sinon, son choix le dirige vers Visualisation.php où on transfère l’id de son qcm en cours afin qu’il en poursuive la modification ultérieurement.

Visualisation.php:

Ce script affiche de prime abord le domaine et le sous domaine du qcm en cours de modification ,et ce à partir de la base de donnée et de l’id du qcm transmis depuis Importer.php, avant d’afficher la question dont l’identifiant a aussi été reçu de la page précédente. De plus, toutes les réponses associées à cette question dans la base de données seront affichées.

Ce script permet alors de visualiser la question telle qu’elle apparaîtra dans son qcm si jamais l'utilisateur choisit de l’importer.

Pour cela deux boutons sont à sa disposition, l’un pour importer la question sélectionnée et l’autre pour retourner à la gestion du qcm sans importer quoi que ce soit. Dans les deux cas, l’utilisateur est redirigé vers Questions.php, ce dernier script allant insérer la question importée si le questionneur en a décidé ainsi.

Finaliser.php:

Ce script permet de prendre en compte dans la base de donnée les derniers choix du questionneur ayant choisi de valider son qcm créé. Ce script reçoit l’identifiant du qcm à finaliser et modifie premièrement la base de donnée pour considérer ce qcm comme “fini” (en modifiant l’état d’un booléen dans la base).

Ensuite ce script propose à l’utilisateur un choix permettant de rendre visible ou non le qcm finalisé aux répondeurs, en informant qu’il pourra modifier ce choix à partir de son profil (voir plus bas).

Le choix du questionneur est alors pris en compte dans la base de données (en modifiant l’état d’un autre booléen), puis le questionneur est redirigé vers l'accueil questionneur du site.

Profil.php:

Ce script implémente les premières pages que le questionneur utilisera s’il souhaite consulter son profil, contenant tous les qcms qu’il a créé auparavant. Ce script est divisé en 3 pages, dont l’utilisateur en utilisera 2 au minimum pour visualiser un qcm de sa création.

La première page affiche tous les domaines de la base de données dans lesquels l’utilisateur dispose d’au moins un qcm toujours existant de sa création antérieure. Les domaines dans lesquels l’utilisateur n’a jamais créé de qcm se verront ignorés (par la sélection en requête sql ) . Le questionneur doit alors choisir l’un de ces domaines représentés avec des boutons, menant à la deuxième page.

La deuxième page prend en compte le domaine sélectionné et affiche tous les sous domaines inclus dans ce domaine si l’utilisateur dispose d’au moins un qcm encore existant (ignorant donc les autres sous domaines de la base), et ce sous forme de boutons parmi lesquels l’utilisateur devra choisir. De plus, d’autres boutons s’ajouteront à ce choix, correspondant aux qcm créés par l’utilisateur dans le domaine sélectionné, mais n’ayant pas de sous domaine.

Ainsi les boutons correspondant aux sous domaines renverront vers la troisième page tandis que ceux correspondant aux qcm enverront l’utilisateur directement à VisualisationQCM.php.

La troisième page quant à elle prend en compte le sous domaine sélectionné et affiche tous les qcm de ce sous domaine et du domaine sélectionné auparavant (au cas où il y ait des sous domaines du même nom dans des domaines différents) dont l’auteur est l’utilisateur. Encore une fois les qcms sont affichés sous forme de boutons, envoyant l'utilisateur sur VisualisationQCM.php.

VisualisationQCM.php:

Ce script sert à afficher l'intégralité du qcm sélectionné depuis le profil. Il affiche toutes les questions reliées à ce qcm dans la base de données ainsi que les réponses associées à ces questions,en distinguant les réponses bonnes des fausses.

Le script affiche ensuite 3 boutons,proposant à l'utilisateur soit de modifier la visibilité, de supprimer ou bien de modifier le qcm affiché précédemment. Ainsi,les 3 boutons renvoient respectivement vers les pages VisibiliteQCM.php, SupprimerQCM.php ainsi que Questions.php qui traiteront la demande de l'utilisateur.

VisibiliteQCM.php:

Ce script reçoit l'id du qcm, sélectionné depuis le profil et transféré à la visualisation de ce qcm.

Le script teste premièrement si ce qcm est terminé ou non en vérifiant la base de données. Si et seulement si ce qcm est fini, on propose alors d'en changer la visibilité, proposant alors de le rendre visible s'il ne l'était pas et inversement.Le questionneur a donc la possibilité d’inverser l’état de la visibilité du qcm, et ce, en testant la visibilité actuelle du qcm dans la base de données.

Quel que soit le choix de l'utilisateur, on lui affiche un message de confirmation avec un bouton permettant de retourner à la visualisation de ce qcm. Cependant si le qcm n'est pas fini, on ne propose pas d'en changer la visibilité, et propose à la place un lien pour retourner au profil.

SupprimerQCM.php:

Ce script sert à supprimer un qcm. Tout comme le script VisibiliteQCM, Il reçoit aussi l'id du qcm sélectionné depuis le profil, et propose la suppression de ce dernier avec un choix entre oui ou non exprimé par deux boutons.

Un message de confirmation de suppression est affiché si l'utilisateur a choisi oui, lui confirmant que le qcm a bien été supprimé de la base de données, incluant la suppression du qcm lui-même ainsi que de sa liaison avec toutes les questions qui le composaient. On lui propose ensuite un lien de retour vers le profil. Si l'utilisateur avait choisi non,on le renvoie directement sur la page de profil.

1. Partie Répondeur

Auteur : Lucie.

En tant que développeur back-end, j’ai développé l’interface répondeur en réponse aux exigences du client.

Afin de concevoir et d’implémenter cette partie, j’ai utilisé différents outils et logiciels libres. Le php a été écrit sur l’éditeur de texte libre Notepad ++. L’application web PhpPgAdmin ainsi que système de gestion de bases de données PostgreSQL installés sur la plateforme de développement WampServer m’ont permis de faire fonctionner localement mes scripts php.

La partie répondeur est constituée de sept pages html dans lesquelles sont inclus du php, et qui suivent un schéma linéaire du choix du QCM au résultat de l’élève.

Je vais aborder dans cette partie la fonction et l’utilité de chaque page de l’interface répondeur. Pour plus de clarté, j’explique sommairement le mode de fonctionnement des pages. Les détails plus précis sur le fonctionnement de certains codes sont eux disponibles dans la partie Algorithme du compte-rendu.

Page 1.ChoixRD.php

Le php de cette page sert à afficher les différents domaines de QCM présents dans la base de données. Le répondeur a le choix entre plusieurs catégories dans lesquelles seront classés les QCM. Cette page a été la première fonctionnalité développée pour cette partie, car le choix entre plusieurs domaines est le premier choix à faire pour le répondeur lors de la résolution d’un QCM.

Afin d’afficher les domaines, un requête préparée a été utilisée. L’affichage des domaines se fait à l’aide d’ ‘echo’ comme pour tous les affichages en php, et le passage à la page suivante (ChoixRDS.php) se fait avec un formulaire. Le formulaire prend en paramètres dans une variable $\_POST\* le nom et l’identifiant du domaine sur lequel clique le répondeur pour les utiliser ultérieurement.

Page 2.ChoixRDS.php

Après avoir choisi le domaine, le répondeur est amené à choisir soit un sous-domaine, soit directement un QCM. Un domaine peut en effet contenir un ou plusieurs sous-domaines : par exemple, le domaine ‘Informatique’ pourra contenir les sous-domaines ‘php’, ‘Java’, ou encore ‘Programmation Orientée Objet’.

Les QCM présents sur la même page que les sous-domaines sont des QCM sans sous-domaine, c’est-à-dire qui regroupent des questions générales sur le domaine ciblé. Dans le cas de l’exemple ‘Informatique’, on pourra par exemple trouver dans un QCM sans sous-domaine des questions portant sur plusieurs thèmes différents en informatique, comme Java, C, ou SQL.

Ces QCM et ces sous-domaines s’affichent sur la page ChoixRDS.php grâce à deux requêtes préparées. Une condition ‘if’ encadre le code PHP et permet de vérifier si le domaine est renseigné dans le $\_POST, afin d’éviter les exceptions liées à l’absence éventuelle de cette valeur.

* Si le domaine est renseigné, une requête préparée sélectionne et affiche tous les sous-domaines liés à ce domaine.
* Si le domaine ne contient pas de sous-domaines, le répondeur en est informé (voir les explications dans la partie Page 3.ChoixRQI.php pour plus de détails).

Un bouton ‘retour’ permet dans tous les cas de revenir à la page précédente si le répondeur s’est trompé de domaine, ou s’il a changé d’avis.

Une autre requête permet de sélectionner et d’afficher les QCM sans sous-domaine qui ont été rendus visibles par leur créateur (voir partie questionneur pour plus de détails sur la visibilité d’un QCM).

Les deux requêtes utilisent la valeur contenue dans le $\_POST pour sélectionner les sous-domaines et QCM à afficher.

Lorsque le répondeur clique sur le sous-domaine qu’il a choisi, un formulaire prend en paramètre les noms du domaine courant et du sous domaine sélectionné, et fait le lien vers la page ChoixRQI.php.

Si le répondeur clique sur un QCM sans sous-domaine, un autre formulaire fait le lien vers la page Executer.php qui permet de répondre directement au QCM, avec en paramètre les valeurs de l’identifiant du QCM (id\_qcm) et le nom du domaine courant.

Page 3.ChoixRQI.php

Dans le cas où le répondeur n’a pas choisi d’effectuer un QCM sans sous-domaine, il se trouve sur cette troisième page après le choix du sous-domaine et du domaine. De la même manière que la deuxième requête de la page précédente, cette page affiche les QCM présents dans le domaine et le sous-domaine courants, sélectionnés avec une requête préparée.

De même que pour le choix du sous-domaine sur la page ChoixRDS.php, si aucun QCM n’est répertorié dans le sous-domaine choisi, le répondeur en est informé. Pour cela, la requête affichant les QCM incrémente un compteur à chaque QCM trouvé dans la base. Si le compteur est égal à 0 à la fin de l’exécution de la requête, cela signifie qu’il n’y a aucun QCM dans ce sous-domaine.

Le formulaire activé lors du choix d’un QCM par le répondeur fait le lien avec la page Executer.php en passant en paramètre les valeurs utiles, soit l’id qcm, et les noms du domaine et sous-domaine auxquels appartient le QCM choisi.

Page 4.Executer.php

Après avoir choisi le QCM à effectuer, le répondeur arrive sur la page Executer.php. L’enregistrement de la date courante à l’aide de la fonction php prédéfinie time() constitue la première action de la page. Cette variable servira à calculer le temps mis par le répondeur pour effectuer le QCM. Le temps total alloué au QCM à effectuer est récupéré grâce à une première requête préparée.

La page Executer.php permet d’afficher toutes les questions du QCM dont l’id a été passée en paramètre, et toutes les réponses possibles pour chaque question. Le QCM s’affiche par l’imbrication de deux requêtes préparées : la première affiche les questions du QCM dont l’id a été passée en $\_POST, et la seconde utilise l’id de la question affichée pour sélectionner toutes les réponses qui s’y rapportent.

Les questions et les réponses ont un mode d’affichage sécurisé par l’emploi de la fonction prédéfinie en php htmlspecialchars, qui permet dans ce cas d’afficher les données sous forme de chaîne, sans interprétation du code. Par exemple, une réponse ‘</br>’ affichée avec htmlspecialchars ne sera pas interprétée par le script php comme un retour à la ligne, mais bien comme la chaîne de caractères ‘</br>’.

Toutes les réponses possibles possèdent une case que le répondeur peut cocher pour les choisir ; les cases cochées sont enregistrées dans un formulaire dans un tableau de $\_POST, de même que la date de début du QCM et son identifiant.

Un bouton ‘submit’ lié au formulaire permet au répondeur de valider le QCM lorsqu’il est satisfait des réponses cochées, et fait le lien avec la page Statistique.php.

Tout le code php est encadré par une condition qui vérifie le bon déroulement du choix du QCM, et l’appel de la page EviteMessageFormulaire.php interdit le retour à la page précédente pour empêcher l’affichage d’éventuelles erreurs ou exceptions\* .

Page 5.Statistique.php

Après avoir validé son QCM, le répondeur arrive sur la page Statistique.php. Cette page va calculer sa note et son temps, puis les enregistrer dans une table qui gardera le résultat.

La page Statistique.php affiche également l’identifiant du QCM, chaque question pour laquelle le répondeur a coché une ou plusieurs cases, et les réponses données. Dans le cas d’une bonne réponse, la réponse s’affiche en vert. Si le répondeur a mal répondu, sa réponse s’affiche en rouge, et la bonne réponse s’affiche en vert en-dessous.

A chaque bonne réponse de la part du répondeur, le score total, ou note, s’incrémente de la valeur décidée par le questionneur à la création du QCM (voir la partie questionneur pour plus de détails). Des requêtes imbriquées sont là aussi utilisées pour afficher les différentes réponses de chaque question : la première sélectionne tout d’abord les questions grâce aux valeurs enregistrées dans le $\_POST. Pour chaque question, la seconde requête préparée sélectionne les réponses s’y rapportant.

Quand une réponse cochée enregistrée dans le $\_POST correspond à une réponse correcte, le score est incrémenté. Sinon, si la réponse cochée est incorrecte, une troisième requête trouve et affiche les réponses justes, et le score n’est pas incrémenté. Enfin, l’incrémentation du score s’affiche à la fin du traitement de la question (ex : Score +1 si la question vaut un point).

La date actuelle enregistrée avec time() sert à calculer le temps passé sur le QCM, par une soustraction entre cette date actuelle et la date passée en $\_POST. Le répondeur est averti s’il a dépassé le temps alloué au QCM, et le score est décrémenté. Enfin, le résultat du répondeur est enregistré.

Lorsque le répondeur a fini d’effectuer son QCM et qu’il a vu la correction sur la page Statistique.php, Deux boutons lui permettent soit de refaire un QCM, soit d’aller consulter ses résultats aux QCM qu’il a fait précédemment sur la page ProfilR.php.

Page 6.ProfilR.php

Les résultats du répondeur sont présents sur la page ProfilR.php. Il peut revoir sa note et son temps ainsi que la date à laquelle il a effectué un QCM précis. Les QCM effectués sont classés par domaine et sous-domaine pour plus de compréhension.

Le script php est composé d’une succession de conditions qui permettent de passer d’un domaine sélectionné aux sous-domaines que celui-ci contient, puis de voir les QCM effectués. L’affichage par défaut est celui des domaines pour lesquels le répondeur a effectué des QCM.

Lors de la sélection d’un des domaines, un lien vers la même page est activé, et le domaine est enregistré dans un $\_GET\*. Lors du rechargement de la page, la condition qui vérifie si le domaine est renseigné devient vraie, et l’affichage des sous-domaines se fait.

L’affichage des QCM de chaque sous-domaine se fait suivant le même procédé. ProfilR.php permet également d’afficher le nombre total de QCM faits, la moyenne générale et le temps total. Les différents affichages sont ici aussi possibles grâce à des requêtes préparées.

Page 7.EviteMessageFomulaire.php

La page EviteMessageFormulaire.php est utile pour empêcher le retour à une page précédente lorsque ce retour n’est pas souhaitable pour le bon déroulement de l’exécution d’un QCM. Par exemple, une fois le QCM validé, le répondeur ne doit pas pouvoir y revenir pour modifier ses réponses. Cette page évite également le message d’erreur ‘Confirmer le nouvel envoi du formulaire’ lors du retour arrière.

Certaines des fonctionnalités de cette page n’ayant pas été abordées en cours, je me suis inspirée de l’exemple présent sur le site Openclassroom. EviteMessageFormulaire.php vérifie d’abord si des valeurs sont enregistrées en $\_POST et les sauvegarde dans un $\_SESSION\* le cas échéant.

La location courante est sauvegardée. Puis lors de l’appel à la page précédente le répondeur est redirigé sur cette page mais n’a plus accès à la modification de quoique ce soit. Les $\_POST sauvegardés sont ensuite remis dans $\_POST, et le script se termine par la destruction de la sauvegarde.

1. Problèmes rencontrés et leurs résolutions

Plusieurs problèmes ont été rencontré surtout dans le code php à cause de la coopération de plusieurs personnes sur le même code. Il est vrai que nous avons chacun notre propre manière de réaliser une fonctionnalité et c’est cela qui a empêché la résolution de bug par un autre développeur que celui qui avait écrit ce code. Par exemple, là où certains privilégient le passage de données d’une page à l’autre avec $\_GET, d’autres préfèrent le faire avec $\_POST.

Pour la partie base de données, vu que Nathan était tout seul, il n’y a pas eu beaucoup de soucis si ce n’est les noms et le type de données utilisés ainsi que l’utilisation des auto\_increment et des valeurs par défaut qui ont été mal interprétés par le reste du groupe.

La base de données a été sujette à plusieurs modifications afin de résoudre les problèmes des requêtes trop longues dans les scripts php mais cela a engendré quelques redondances dans les relations notamment recap répondeur.

Il a été demandé que lors de la création d’une question, le Questionneur devait préciser qu’une réponse était juste en cochant une case située à droite du champ de la réponse. Or, lors de différents tests, il a été remarqué que lorsqu’une case à cocher est non cochée, l’interpréteur php estime que cette case n’existe pas. Donc lors de l’affichage des réponses dans la page Question, si une case était non cochée, l’état de toutes les cases passe à décochée et donc le texte d’une réponse désignée incorrecte était affiché à toutes les réponses. C’est pour cela qu’une liste à deux choix a été utilisée pour cette fonctionnalité.

Le développement d'une partie du code de la partie questionneur du site réalisée par Florian a été soumise à des contraintes sûrement dû à une mauvaise installation de Wamp, l'empêchant de réaliser n'importe quelle requête en étant connecté avec le rôle secondaire créé par Nathan. La source du problème semblant irréparable car difficile à localiser, ce développement a dû se faire prudemment, c'est-à-dire continuer à tester le code à l'aide du rôle administrateur, mais vérifier qu'aucune requête interdite au rôle secondaire n'a été implémentée dans le code, requête qui fonctionnerait alors sur la machine de Florian mais pas sur celle des autres.

Le développement de la partie questionneur à été soumis a beaucoup de changements de la structure du code, dus aux précisions des exigences du client apportées au cours du projet, à un changement de la méthode de fonctionnement utilisée ou encore dû à une surcharge des données transmises entre chaque page du site.

Par exemple la méthode de fonctionnement utilisée au niveau des requêtes a souvent changé pour permettre la création d'un qcm, car la première méthode adoptée était de retenir toutes les questions et réponses créées ou importées pendant la création du qcm, jusqu'à la validation de celui-ci, et d'implémenter le qcm ainsi que ces questions dans la base de données lors de la finalisation uniquement. Cela surchargeant les données à transférer entre les pages, nous avons opté pour créer le qcm dans la base de données dès le début afin d'y ajouter les questions créées et importées au fur et à mesure.

Cette méthode résout aussi le problème de déconnexion intempestive de l'utilisateur pouvant alors perdre son travail réalisé, au lieu de cela il peut retrouver son travail inachevé à partir de son profil et le modifier.

Enfin,cette méthode était convenable aux précisions des exigences apportées par le client étant de pouvoir modifier la visibilité d'un qcm créé auprès des répondeurs,car un qcm inachevé n'est parfois pas présentable.Ainsi l'utilisateur peut aussi retrouver son travail estimé inachevé dans la base de donnée à partir de son profil pour en modifier la visibilité.

Les autres problèmes rencontrés sont mineurs car ils ne sont que des soucis de requêtes erronées provoquant des bugs, comme des bugs d'affichages dupliqués de réponses à une question car cette question est utilisée par plusieurs qcm à la fois dans la base, ou encore un bug lors de l'importation d'une question, proposant d'importer une question déjà présente dans le qcm, violant les contraintes des tables de la base lors de l'importation. Le premier cas a été résolu en changeant la requête erronée tandis que le deuxième cas a été traité par du php, « complétant » ainsi la sélection de la requête.

Le majeur problème rencontré sur la partie répondeur est lié à l’absence de l’utilisation de javascript. Pendant l’exécution d’un QCM, il était prévu de stopper celui-ci si le répondeur dépassait le temps alloué. Mais la manière d’implémenter le calcul du temps passé (voir partie ‘Répondeur’, page ‘Executer’) n’a pas permis de calculer le temps durant un QCM : avec du javascript, nous aurions pu avoir un calcul ‘dynamique’ du temps qui passe pendant l’exécution du QCM. Sans, nous ne pouvions pas détecter le moment où le répondeur passait au-delà de la limite, et ainsi arrêter le QCM. Une des améliorations possibles serait donc d’ajouter du javascript pour rendre cela possible. A part ce problème, la partie Répondeur n’a pas présenté d’autres difficultés majeures dans son implémentation.

1. Les algorithmes : solutions techniques

Partie Répondeur :

J’ai abordé la façon globale dont fonctionnent les scripts de la partie Répondeur dans la partie Développement-Répondeur. Je vais donc seulement commenter plus en détail un de mes scripts qui me paraît le plus intéressant au niveau algorithmique, celui de la page Statistique.php.

Il s’agit de la fonctionnalité pour calculer le score du répondeur à partir des réponses données. Une requête parcourt les réponses des questions. On cherche d’abord le nombre de bonnes réponses à une question (lignes 60 à 69 dans l’annexe Statistique1). Pour chaque réponse, si sa valeur ‘correct’ est à true dans la base, la réponse est vraie et la variable $reponsetrue s’incrémente (ligne 68). Cette variable sera utilisée dans la suite du script.

Après avoir compté le nombre de bonnes réponses, une boucle (foreach l.71) permet de parcourir les réponses données par le répondeur enregistrées dans le tableau. Si l’id de la réponse correspond à l’id de la réponse enregistrée (l.73, cad si le répondeur a coché cette réponse), la suite de l’algorithme va incrémenter plusieurs compteurs en fonction des réponses données. Le compteur $vrai qui enregistre les bonnes réponses s’incrémente si la réponse donnée est vraie (correct =true dans la base, l.78). Sinon, la requête l.82 sélectionne la ou les réponses vraies pour les afficher (l.82 à 123 sur Statistique2). Un compteur de réponses fausses $faux s’incrémente.

Après avoir vérifié ainsi toutes les réponses du répondeur à la question, le l’algorithme va déterminer l’ajout-ou non- de points au score suivant les différents cas de figure.

Il y en a trois:

* l.98 à 106 : le répondeur n’a pas de réponse fausse à la question, et il a coché toutes les réponses vraies ($faux est à 0 et $reponsetrue est égal à $vrai). Le score s’incrémente de la valeur allouée à la question (l.103-106.)en récupérant celle-ci dans une requête préparée (l.99).
* l.108 à 120 sur Statistique3: le répondeur n’a pas de réponse fausse mais il n’a pas répondu à toutes les réponses vraies ( $faux est à 0 mais $reponsetrue est différent de $vrai). Le score ne s’incrémente pas et l’état incomplet de la réponse est affiché, ainsi que la ou les bonnes réponses (l.112-117).
* l.120-123: le répondeur n’a pas répondu à la question ($vrai et $faux sont égaux à 0). Le score n’est pas incrémenté et un message s’affiche (l.121).

Les trois variables l.125-127 se remette à 0 pour la prochaine question de la boucle. Hors de la boucle, le score est décrémenté de 5 si le temps est dépassé (récupération du temps l.132-138, vérification si il est dépassé l.140-143). Si le score devient inférieur à 0, il est remis à 0 l.147-149.

Enfin, le score est mis sur 20 par la requête l.151-156 et est affiché.

Partie questionneur

La partie questionneur est composée d'un bon nombre de script proposant plusieurs pages tout en restant sur le même script, il semble donc nécessaire d'en décrire l'algorithme en montrant comment les échanges d'informations entre les pages sont réalisés et comment ces pages sont séparées les unes des autres.

De plus la plupart des scripts de la partie questionneur interagissent avec la base de données de la même manière, c'est pour cela qu'un script tel que ChoixQC.php, réunissant requêtes sql, traitement de la sélection de cette requête et plusieurs pages réunies en un seul script, sera présenté, même si sa méthode pour échanger des informations entre ses pages est différent des autres scripts puisqu'on y utilise le $\_GET au lieu du $\_POST.

Comme montré sur l'annexe ChoixQC1 , le code de la première page de ce script se retrouve à la toute fin du code php puisqu'il commence par un else ligne 95 qui termine la condition expliquée plus bas, condition qui regroupe la quasi-totalité du code php.

Cette première page ne reçoit aucune donnée envoyée via un formulaire d'une page précédente. Dans cette page on interagit avec la base de données avec une requête préparée sélectionnant toute la table des domaines (ligne 97-98).

Une fois la requête exécutée on parcourt à l'aide d'une boucle while toutes les tuples sélectionnées avec la requête, et ce grâce à la méthode fetch(PDO::FETCH\_ASSOC) qui associera à la variable $ligne un tableau correspondant à la tuple parcourue. Cette méthode renvoie false quand il n'y a plus de tuples à parcourir, permettant alors de sortir de la boucle while (ligne 99).

Dans cette boucle de cette première page, on utilise pour chaque domaine parcouru dans la table un formulaire renvoyant sur ce même script mais sur la deuxième page à l'aide d'un $\_get inséré dans l'url de l'action du formulaire qui prend la valeur du nom du domaine, en représentant ce domaine par le bouton « submit » du formulaire (ligne 103 à 105).

Sur l'annexe ChoixQC2 on peut apercevoir la condition qui sépare les pages de ce script les unes des autres. En effet, ligne 45 la condition (et toute premiere du code php) teste à l'aide de la fonction isset() et trim() si il existe une variable du nom de « d » dans le $\_GET, « d » correspondant au nom du domaine transmis depuis la première page dans le formulaire du domaine sélectionné par l'utilisateur. La variable « d » étant aussi présente dans la troisième page on teste aussi la fonction isset() d'une variable nommé « sd » qui est dans la troisième mais pas dans la deuxième page(ligne 45).

Cette deuxième page reprend à peu près le même algorithme que la première,en implémentant un formulaire pour chaque sous domaine du domaine reçu en $\_GET qui enverra vers la 3ème page à l'aide de deux variables passées en $\_GET(ligne 56 à 63). Mais cette page propose un formulaire de plus permettant à l'utilisateur de créer son qcm sans sous domaine, ce formulaire implémente la deuxième variable avec la valeur nommée « general » et réalise la même action que les autres formulaires (ligne 52 à 54).

La troisième page, représentée par l'annexe ChoixQC3, est enclenchée quand un sous domaine (dont le sous domaine nommé « général ») est sélectionné. Si la condition ligne 65 testant la fonction isset et trim des deux variables transmises depuis la deuxième page est vraie, cette troisième page utilisera une requête préparée (différente si le sous domaine est « général ») qui instaurera un qcm du domaine et du sous domaine reçu en $\_GET dans la base de données (ligne 67 à 77).

Le script récupère l'id du qcm fraichement créé en parcourant la table et en prenant le dernier id de qcm de la table(ligne 81 à 83), avant d'afficher un bouton appartenant à un formulaire qui transmettra à la page ChoixQQ.php cet id ainsi que le domaine et le sous domaine s'il n'est pas général, et ce en $\_POST(ligne 85 à 92).

Je vais expliquer les requêtes SQL de la page Question.php une fois que toutes les valeurs de la question ont été spécifiées dans CreationQuestion.php. Toutes les lignes que j’indiquerai seront celles du script Question.php

Requête 0 : Si la modification est activée, la valeur de ‘fini’ qcm du qcm se remet à false, il n’est plus visible pour le réopndeur.

Requête 1 : L’insertion de l’intitulé de la question

Cette requête possède deux formes :

* l’une à quatre bindValues comprenant l’intitulé de la question, la valeur, le temps et l’explication. (l.69)
* l’autre à trois bindValues, identique au précédent avec la donnée de l’explication en moins (l.77).

Cette variation est due au caractère facultatif de l’explication d’une question.

La question est insérée dans la table question.

Requête 2 : La récupération de l’Id de la question

Cette requête permet de stocker l’Id de la question qui permettra plus tard de relier cette question à un QCM.

Cette requête est un select de la table question (l.86).

Requête 3 : La récupération du nombre de points de la question

Cette requête permet de stocker le nombre de points de la question qui seront utilisés plus tard pour augmenter le score du QCM, via un select de la table question (l.94).

Requête 4 : L’incrémentation du score du QCM

Ici, la requête est un update de la table qcm où le nombre de points récupéré précédemment est ajouté au score total du QCM. (l.101).

Requête 5 : L’insertion des réponses de la question

Cette requête nécessite au préalable de créer un tableau contenant les valeur vrai ou faux des réponses.(l.106) Ensuite ce tableau est parcouru et selon l’état vrai ou faux, la requête d’insertion est différente :

Si la réponse est juste, le bindValue correct est true et la réponse est insérée dans la table réponse avec comme attributs son id et son intitulé. (l.116). Si la réponse est fausse, le bindValue correct est false et la réponse est insérée avec les mêmes attributs. (l.124).

Requête 6 : Le lien entre la question et le QCM

Avec cette requête, on utilise l’Id de la question et l’Id du QCM, pour insérer les deux dans la table qcm\_question qui permet de faire le lien entre le qcm et la question (l.133).

Toutes ses requêtes sont exécutées si le questionneur a créé une question.

Les requêtes qui suivent sont exécutées si le questionneur importe une question :

Requête 7 : La récupération des points de la question (l.142).

Cette requête a le même fonctionnement que la requête 3.

Requête 8 : L’incrémentation du score du QCM (l.150)

Cette requête a le même comportement que la requête 4.

Requête 9 : Le lien entre la question et le QCM (l.156)

Ici, la requête est la même que la requête 5.

Requête 10 : L’affichage de la question

Cette requête est un select des tables qcm, qcm\_question et question. Lors de l’exécution de la requête, on crée une boucle qui permettra d’afficher toutes les questions du qcm, avec leurs points, leur temps et leur explication. Une fois que toutes les questions du QCM sont affichées la boucle se termine. (l.168-181)

Requête 11 : L’affichage de réponses

Cette requête est un select des tables réponse, question, qcm\_question et qcm. Cette requête affiche les réponses des questions en les affichant en vert si elle sont justes et en rouge si elles sont fausses. (1.184-195)

Requête 12 et 13: Affichage des questions et réponses quand une question a été supprimée.

Ces requêtes fonctionnent de la même manière que les requêtes 10 et 11.

1. Synthèse des comptes rendus des points de rendez-vous

Tout au long du projet, les rendez-vous avec le client ont marqué les étapes de la conception puis de l’implémentation. Du 21 septembre 2016 au 5 janvier 2017, huit rendez-vous ont été pris régulièrement pour faire part de l’avancement du projet et des améliorations à y apporter. A chaque rendez-vous à partir du second, une démonstration de l’avancement du site était présentée.

21/09 - Le premier rendez-vous portait sur des précisions de la définition du sujet de projet« Gestion des QCM ». En reprenant le sujet point par point, nous avons demandé des précisions sur les différents aspects du projet. Ont été abordés : la forme du logiciel web, le mode d’affichage des résultats aux QCM, le design du site.

28/09 - Le deuxième rendez-vous portait sur la répartition des tâches et la base de données. L’établissement d’ébauches de la base de données et de la création de QCM a été planifié.

05/10 - Le troisième rendez-vous portait sur la présentation du site et la création de la base de données. Nous avons abordé plus en détail l’implémentation de l’interface questionneur, et plus particulièrement la façon d’inclure des questions existantes lors de la création d’un QCM. Nous avons aussi soumis une première version de la base au client.

19/10 - Le quatrième rendez-vous portait sur l’hébergement du projet et la présentation de la fonctionnalité de tri des QCM. Il a été décidé que l’hébergement se ferait sur le serveur utilisé par un enseignant à la fin du projet. La démonstration portait essentiellement sur la fonctionnalité de tri de questions de la partie questionneur et de leur inclusion dans la base, qui est à présent fonctionnelle.

09/11 - Le cinquième rendez-vous portait sur la présentation de la connexion à la base de données et l’établissement de rôles sécurisés, la création de question pour le rôle questionneur, le déroulement d’un QCM pour le rôle répondeur, et la modification du design.

30/11 - Le sixième rendez-vous portait sur l’avancement du site et le suivi des fonctionnalités abordées précédemment. Nous montrons l’avancement de l’implémentation et le déroulement des différentes fonctionnalités.

14/12 - Le septième rendez-vous pris le portait sur la finalisation des fonctionnalités des deux interfaces et sur l’aspect du site. Des modifications sont apportées à l’ergonomie du site. De nombreux tests permettent d’améliorer l’utilisation du site.

05/01 - Le huitième rendez-vous portait sur la finalisation de l’ensemble du projet. Nous avons discuté des dernières modifications à effectuer et de ce qu’il fallait rendre (rapport et soutenance).

1. Conclusion

En conclusion, malgré les problèmes rencontrés, le projet a atteint son stade de développement final. Même si des améliorations sont toujours possibles, le site est fonctionnel et répond aux exigences du client. Faire partie d’une équipe de développement nous a appris à travailler en coordination sous la direction d’un chef de projet, et à répartir les tâches en fonction des compétences de chacun. La communication, essentielle entre les différentes parties et au sein du groupe projet, nous a permis d’arriver ensemble plus rapidement au but désiré.

1. Glossaire

Développement front-end : Partie du développement concernant la forme de l’application ou du logiciel. Le développeur front-end utilise des langages de structuration et de présentation comme html et css.

Développement back-end : Partie du développement concernant le fond de l’application ou du logiciel. Le développeur back-end utilise des langages de programmation comme php, javascript, ajax par exemple.

PDO (php avec l’extension) : PDO, ou PHP Data Objtects, est une extension du langage php, définissant une interface d’accès à une base de données, et permettant son utilisation au sein du php.

User stories (ou récit utilisateur): Phrase écrite dans un langage simple qui permet de décrire le contenu d’une fonctionnalité à développer sous la forme « En tant que <qui> je souhaite <quoi> afin de <pourquoi> ».

$\_POST : Tableau de variables en php, qui permet de stocker des valeurs et les passer d’une page html à une autre, sans faire apparaître ces valeurs dans l’url de la page.

$\_GET : Tableau de variable en php qui permet de faire passer des données d’une page web à une autre via l’url de la dite page

Exception : Erreurs lancées par PHP lorsqu’une anomalie de traitement est détectée.

$\_SESSION : De la même manière que $\_POST, $\_SESSION permet de stocker des valeurs, mais au lieu d’être accessible sur une seule page, ces valeurs sont accessibles sur toute page contenant la ligne ‘session\_start()’ dans le script php.

1. Annexes

1) Lien vers Tableau Trello <https://trello.com/b/scTm5Kdr/qcm-project>

2) Arborescence du site + Légendes

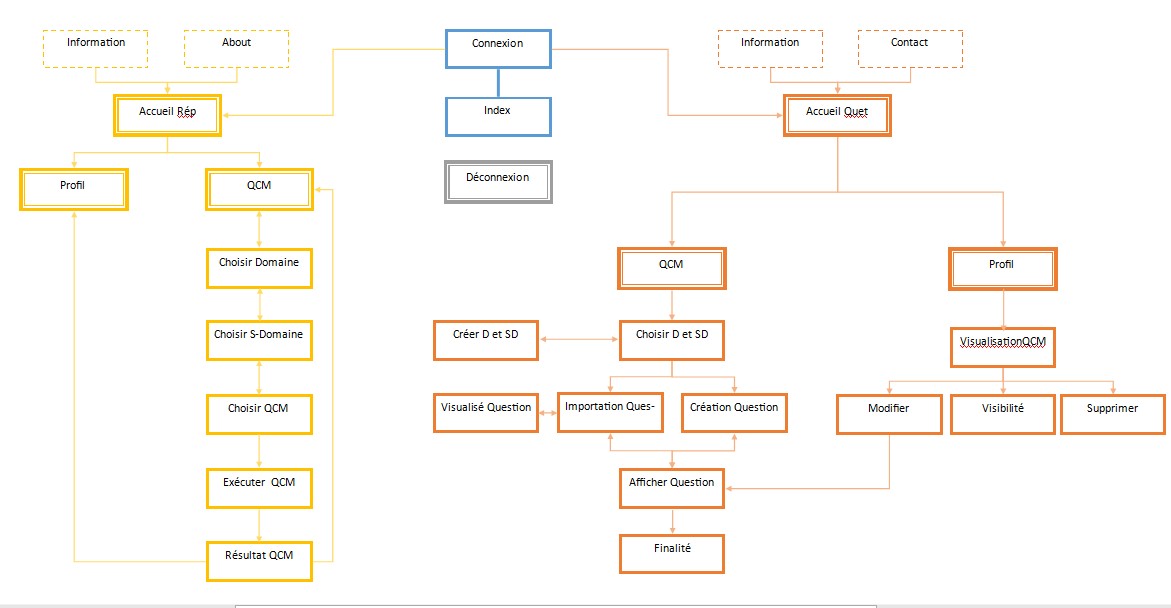
3) Schéma relationnel BDD (Modèle Logique de Données)

4) Capture d’écran de Statistique.php

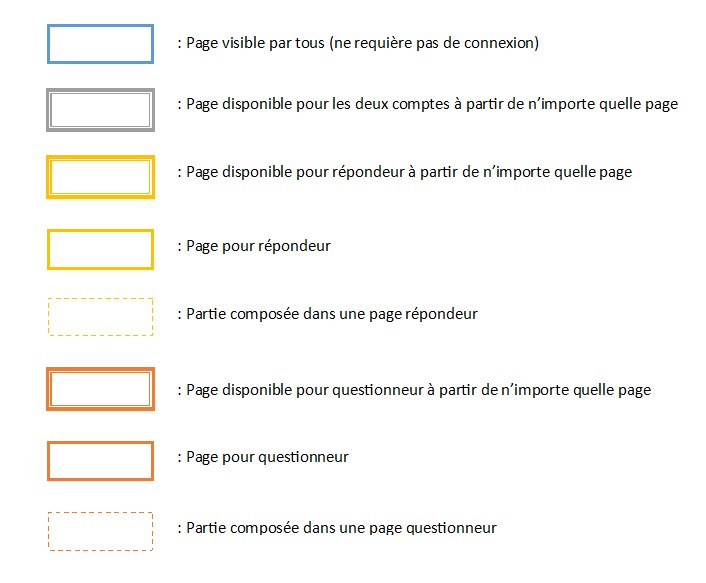
5) Capture d’écran de ChoixQC.php

6) Capture d’écran de Questions.php

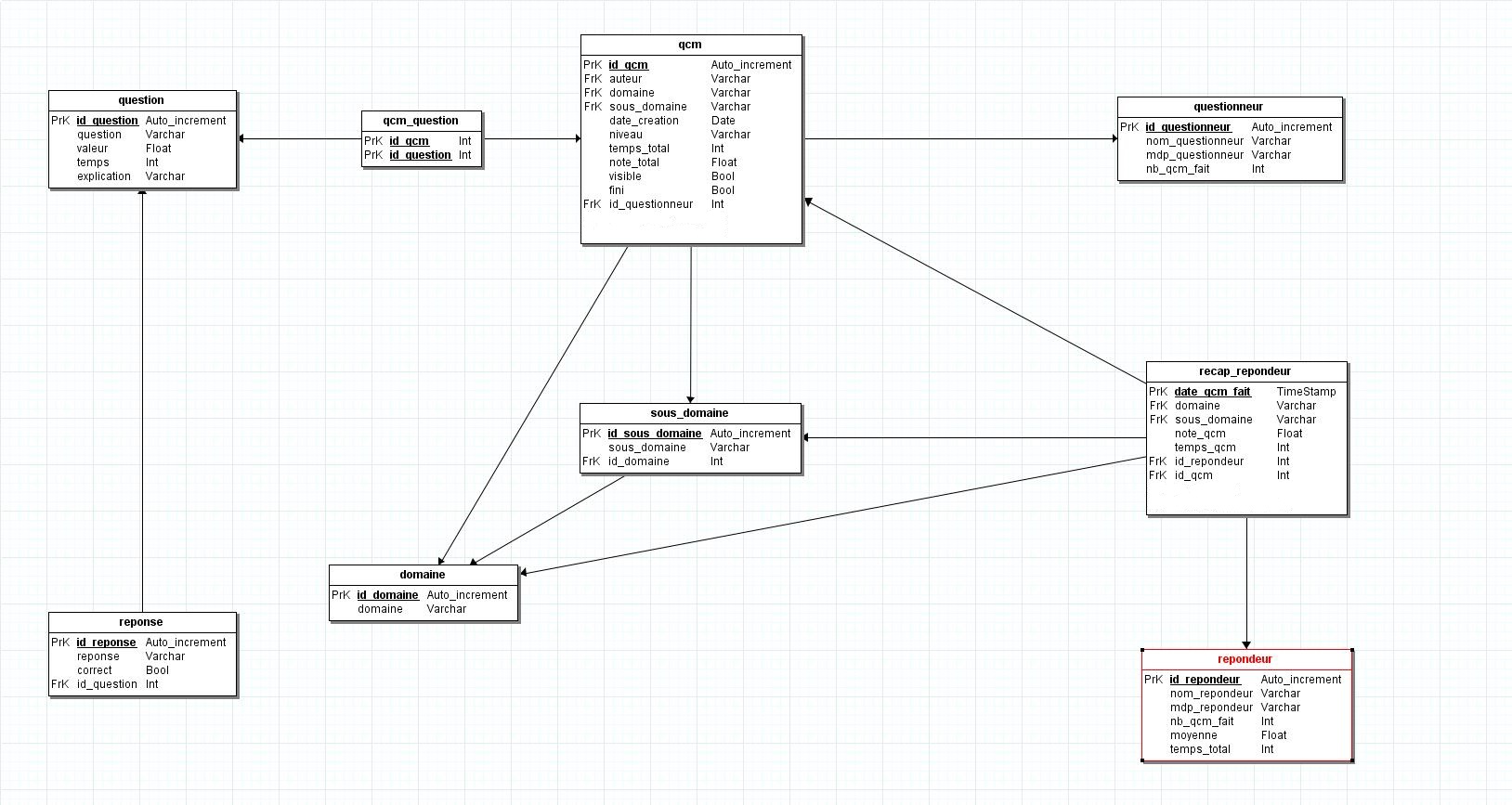
* + 1. Arborescence du site (1-schéma)



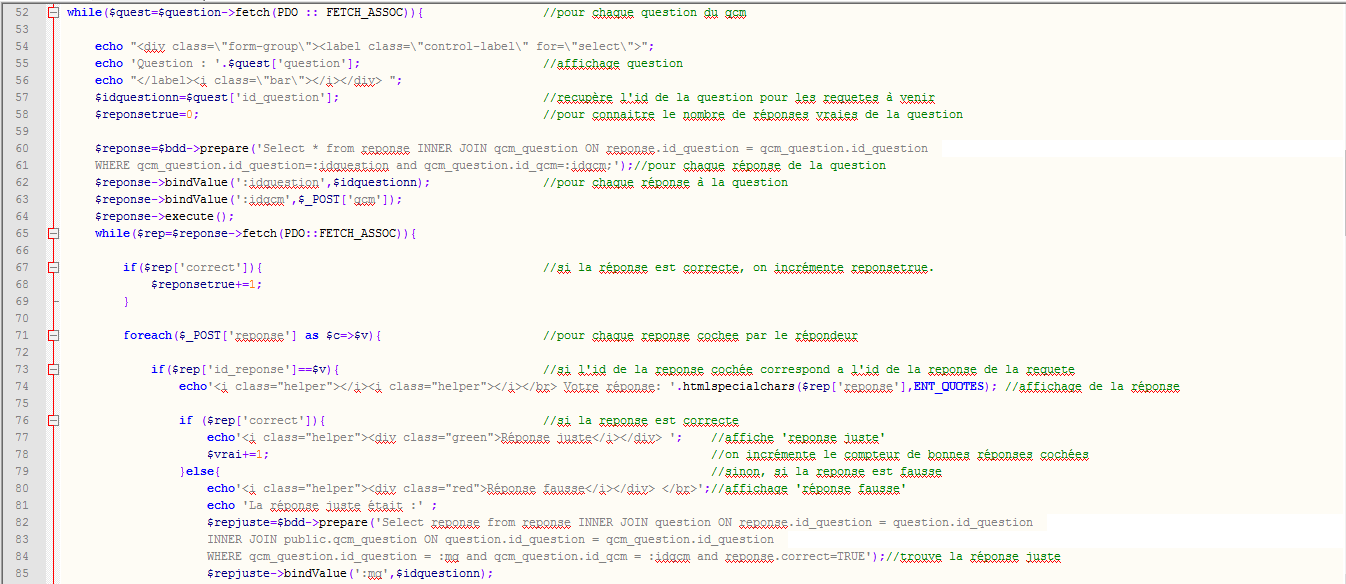
1. Arborescence du site (2-légende)



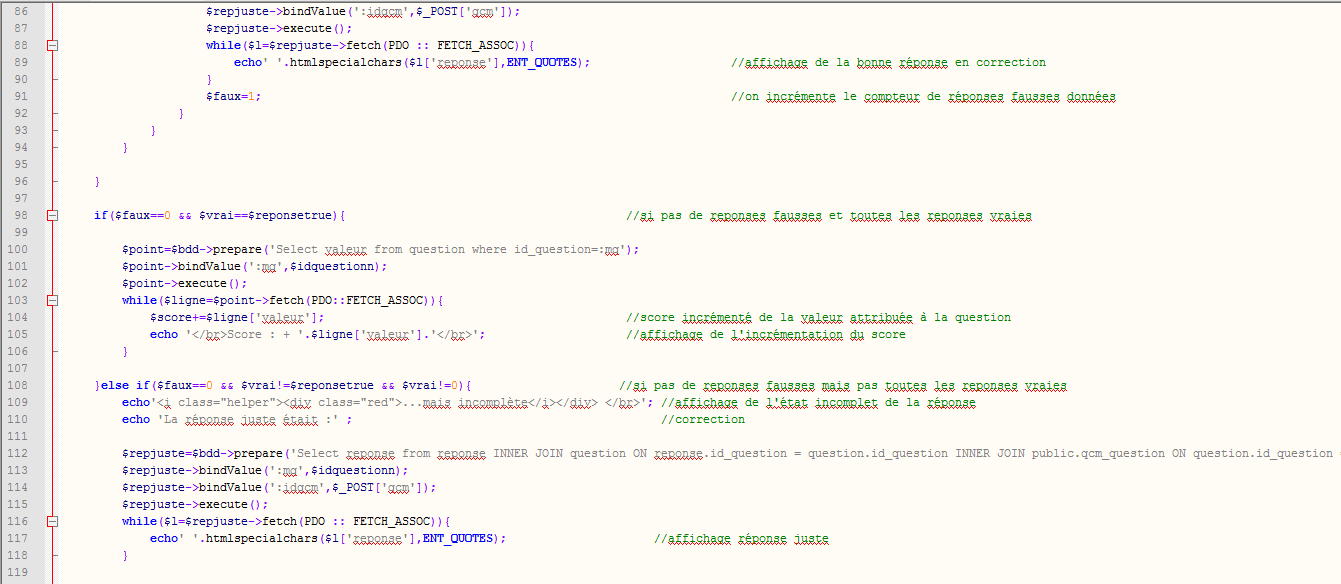
2. Schéma relationnel de BDD (MLD)



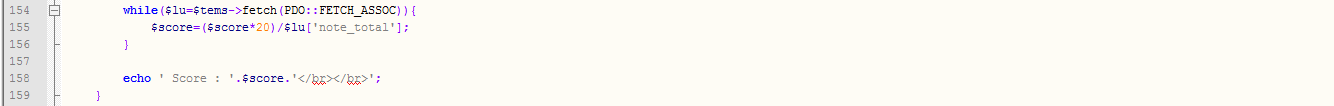
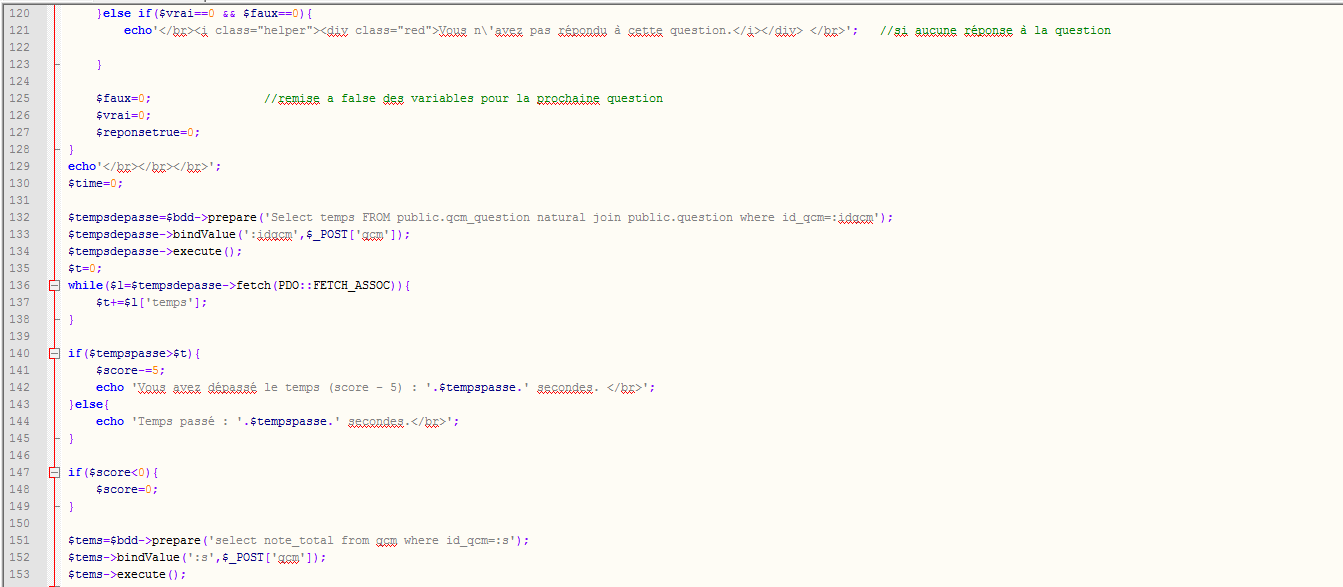
3. Statistique (1)



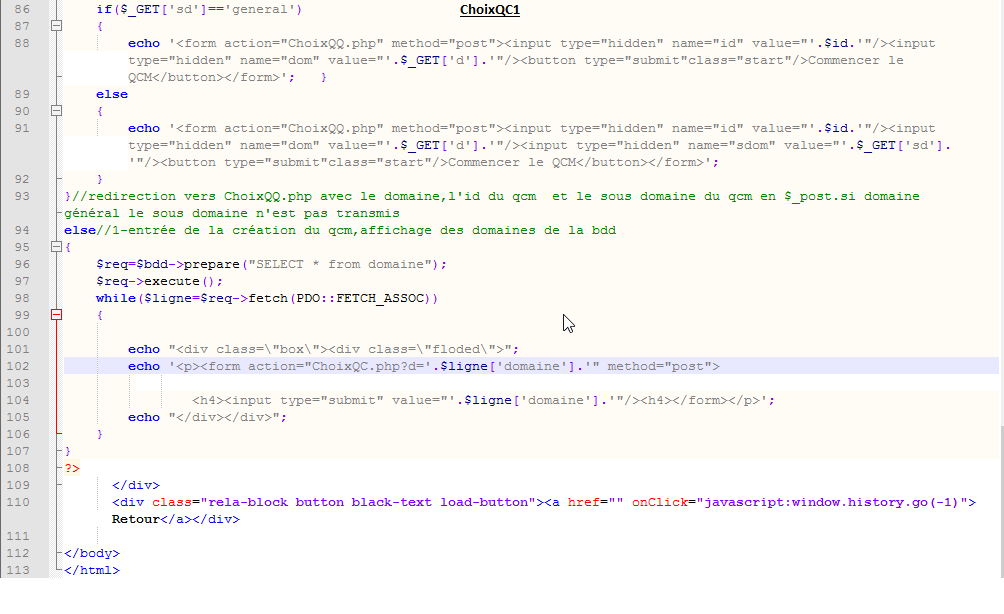
3. Statistique (2)



3. Statistique (3)



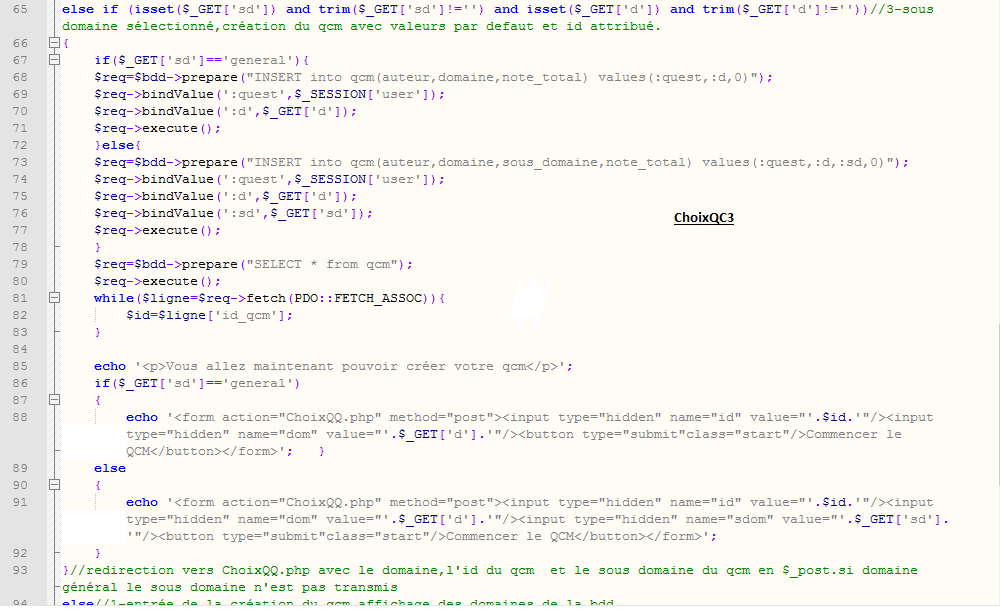
4. ChoixQC (1)



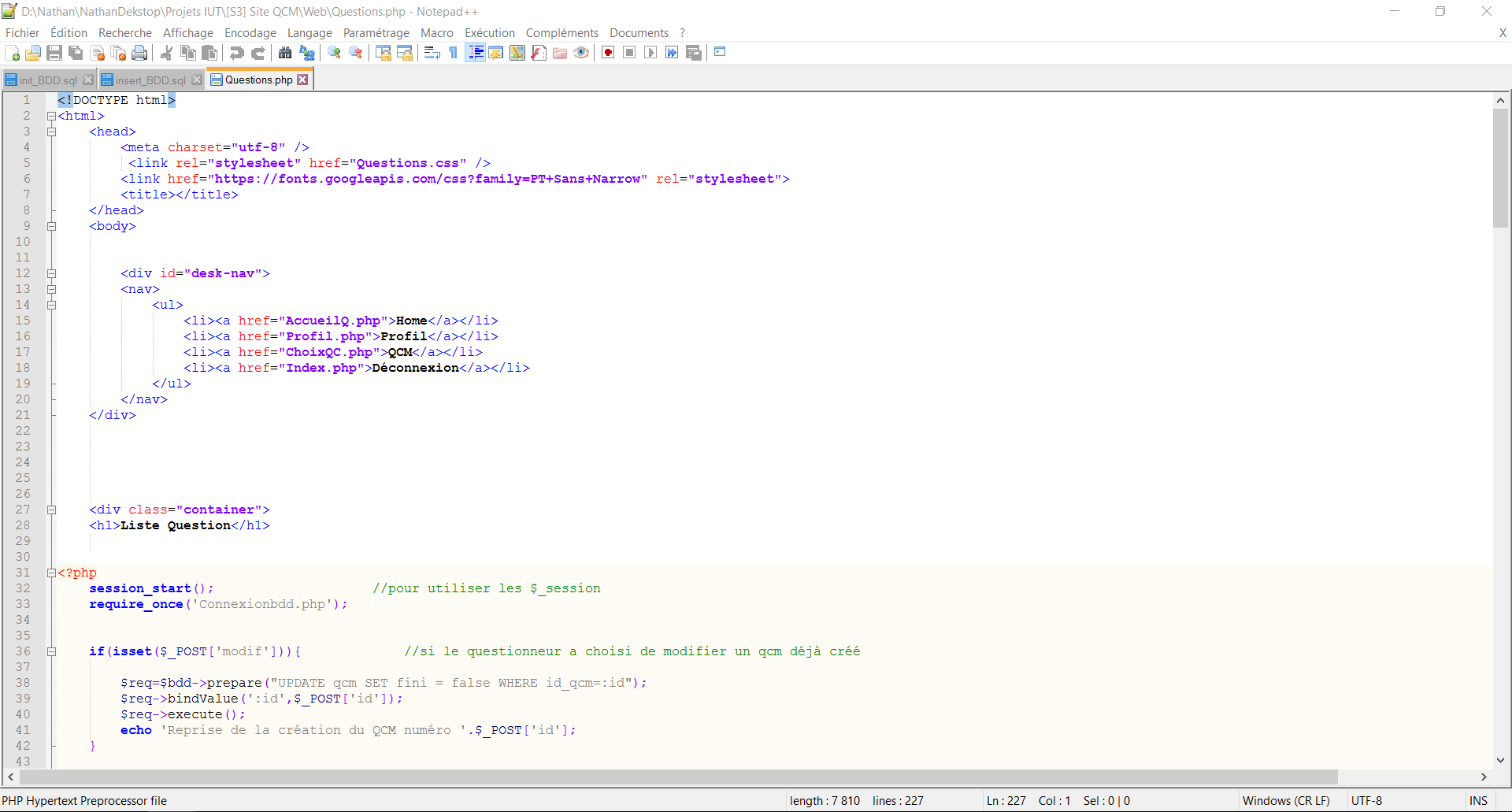
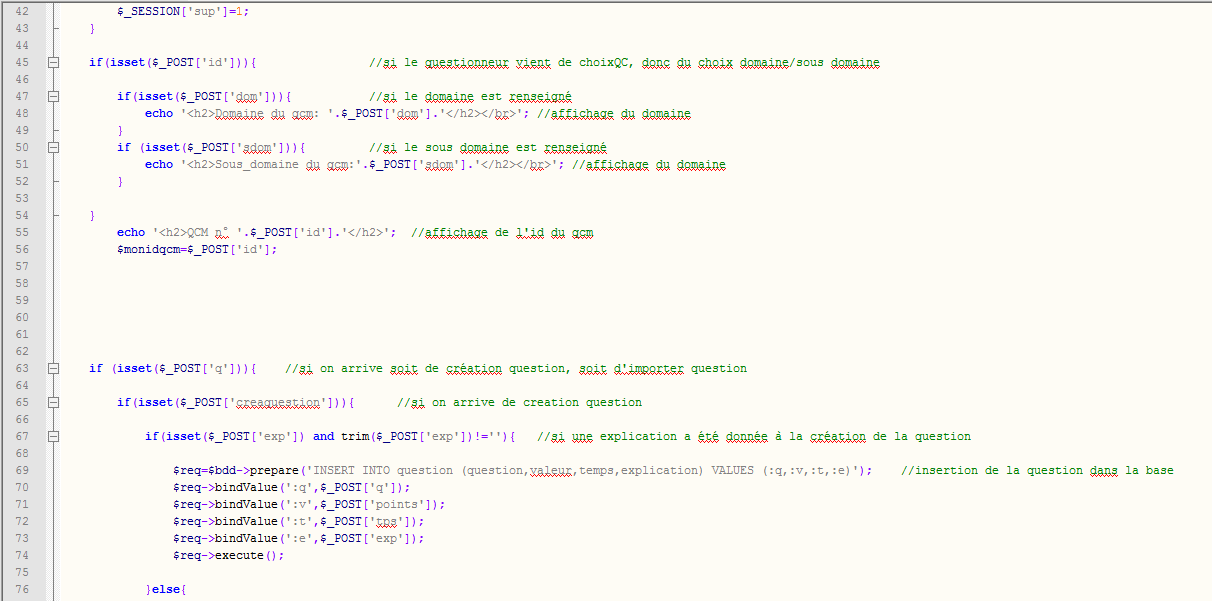
4. ChoixQC (2)



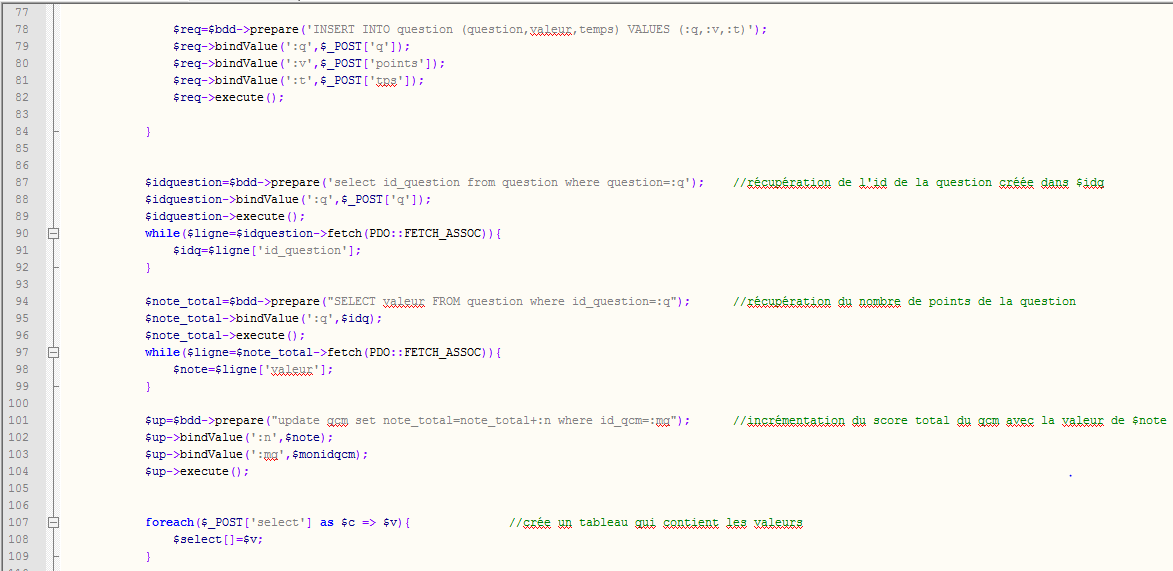
4. ChoixQC (3)



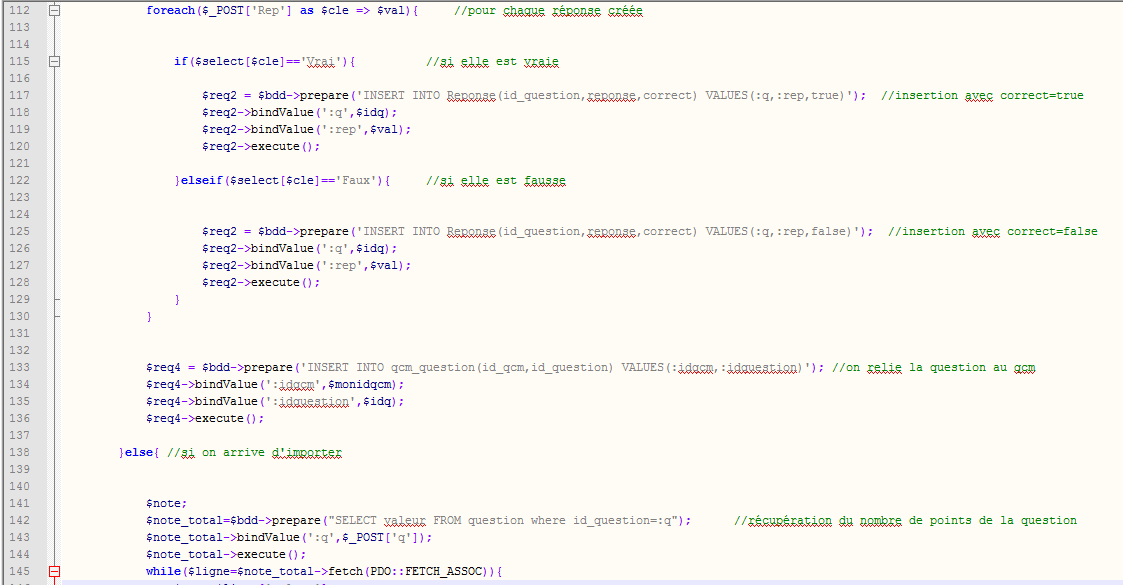
5. Questions (1)

5. Questions (2)



5. Questions (3)



5. Questions (4)



5.Questions (5)

