Rapport :

Module 151 – PHP /MySQL/ODBC



Table des matières

[Introduction 3](#_Toc403132920)

[Les choix techniques 3](#_Toc403132921)

[Le code 4](#_Toc403132922)

[Structure physique 4](#_Toc403132923)

[Répartition du travail 5](#_Toc403132924)

[Planification du travail 6](#_Toc403132925)

[Difficultés, problèmes et challenge 7](#_Toc403132926)

[Protection contre les injections 8](#_Toc403132927)

[Architecture logique 8](#_Toc403132928)

[Séparation Access et MySQL 9](#_Toc403132929)

[Modélisation SQL 9](#_Toc403132930)

[Le cas des adresses, un problème intéressant 9](#_Toc403132931)

[Technique SQL 12](#_Toc403132932)

[Normes, standards et syntaxe du code 12](#_Toc403132933)

[Références, aides et source 12](#_Toc403132934)

## Introduction

Le module 151 met en œuvre le langage de programmation PHP, le SQL avec la base de données MySQL et Microsoft Access par ODBC. C'est avec ces outils que nous allons créer un shop en ligne.

Nous sommes donc amenés à nous mettre en conditions professionnelles et développer dans un délai de 9 semaines selon un cahier des charges.

Ce travail de groupe requiert une bonne organisation et beaucoup de communication. En effet, il n’est pas forcément facile de travailler à plusieurs sur les mêmes fichiers.

## Les choix techniques

Pour le partage des fichiers et la gestion des versions du code, nous avons choisi d’utiliser Bitbucket ([https://bitbucket.org](https://bitbucket.org/)) en utilisant Mercurial comme gestionnaire de version et SourceTree comme client. Bitbucket fournit aussi une interface web similaire à GitHub, mais avec l’avantage d’avoir son dépôt de code « privé ».

Ainsi chacun à la possibilité de lire, commenter et modifier le code des autres en gardant un historique des modifications. Cela est essentiel pour un développement à plusieurs afin de pouvoir être fortement coordonnés et d'avoir toujours la dernière version du code.



## Le code

Pour le code, nous avons choisi de partir sur de l’orienté objet pour certains modules uniquement du site, principalement les accesseurs à la base de données (dénommés entrepôts). Pour le reste, nous avons conservé une approche classique.

La structure proposée est la suivante : chaque partie du site est décomposée en plusieurs fichiers, qui seront appelés dans un fichier central (cf. "Architecture Logique").

Le challenge de ce shop était d’utiliser deux bases de données différentes (cf. "Séparation Access et MySQL").

Au début, comme l'accès à la base Access n'était pas disponible, nous avons créé une interface IAuthentificateur, implémentée par un object FauxAuthentificateur qui répondait toujours par la positive à toutes les demandes d'authentification. De même pour les entrepôts des articles et des utilisateurs. Cela nous a permis de quand même développer l'interface utilisateur en utilisant des données générées aléatoirement pour l'affichage. Ensuite, quand la base de données fût prête, nous avons remplacé les "faux" par les classes définitives et avons supprimé les interfaces.

## Structure physique

[Shop] –
 [autoload] -
 EntrepotArticle.php
 EntrepotCategorie.php
 GestionnaireUtilisateur.php
 IAuthentificateur.php
 IEntrepotCommandes.php
 Shop.php

 [include] -
 [administration] -
 admin.php
 articles.php
 commandes.php
 commentaires.php
 bdd151.sql
 bdd151\_identificationUtilisateur.accdb
 bdd151Data.sql
 body.php
 commentaire.php
 config.php
 configbdd.php
 fluxRSS.xml
 fonctions.php
 foot.php
 head.php
 login.php
 panier.php
 payement.php
 rss.php
 style.css

 [user] –
 content.php
 function\_user.php
 [img] -
 captcha\*.png
 logo.png
 merci.png

 Index.php
 parametresbdd.html
 README.md

## Répartition du travail

La répartition du travail dans une équipe de développement n’est pas chose facile… Effectivement, certains travaux paraissent difficiles et complexes alors qu’ils sont finalement assez rapidement faits, alors que d’autres qui paraissaient plus simples, peuvent poser de multiples problèmes.

Nous avons choisi de séparer le site en trois parties :

* Partie « User » (M. Lehmann)
* Partie « Admin » (M. Liardon)
* Partie « MySQL et ODBC » (M. Pereira).

Cet objectif a très bien marché et nous a permis d’aider nos collègues une fois notre travail terminé. C’est aussi un gain de temps énorme pour éviter de faire du travail à double et de se marcher sur les pieds.

Dans les faits M. Liardon aura finalement travaillé sur la partie admin, MySQL et ODBC (correction et modification de la modélisation, implémentation). M. Lehman sur la partie user, la mise en page et la structure du site, ainsi que des requêtes SQL et le CSS. M. Pereira sur le modèle de données de base, la base Microsoft Access, le CSS pour l'impression, le flux RSS, des requêtes SQL et de la logique PHP.

## Planification du travail

Pour plus de clarté, nous avons mis en forme la planification sous forme de schéma de Gantt et Pert.

Le schéma de Gantt permet d’avoir une vision sur le temps :



Alors que le schéma de Pert permet d’avoir une vision par entité et sous-entité :



## Difficultés, problèmes et challenge

La première difficulté apparente dans le cahier des charges était la double base de données. Mais une fois cet obstacle passé, nous avons pu avancer plus rapidement.

Le challenge suivant fut le captcha. Nous avons hésité à utiliser une API Google ou un module préfabriqué. Mais cela nécessitait de connaitre et de comprendre tout le code importé, ce qui aurait finalement pris beaucoup de temps. Nous l’avons alors écrit nous-mêmes, et cela s’est plutôt bien passé.

Dans le cahier des charges, il était demandé d’ajouter un « grain de sel » à la création de l’utilisateur. Ce « grain de sel » unique serait ensuite stocké quelque part.
Nous avons décidé de faire autrement : un « grain de sel » unique pour chaque utilisateur à la place d’un seul pour l’ensemble des entrées. Cela dans le but de renforcer la sécurité et avec accord de l'enseignant. En ayant un salt unique à chaque utilisateur, il n'est pas possible de savoir si deux d'entre eux ont le même mot de passe. Avec un salt unique, cela serait possible.
Références: <https://crackstation.net/hashing-security.htm> et <http://www.sitepoint.com/password-hashing-in-php/>

Les commandes SQL étaient aussi une autre difficulté. En effet, nous avons choisi de faire certains appels en PDO, ce qui change considérablement la compréhension de la structure du code.

La dernière chose que nous avions envie de mettre en place était l’installation automatique de la base de données. C’est un procédé que l’on peut retrouver sur Wordpress et qui permet d’ouvrir une page html et d’installer son site via une interface web.
C’est une fonctionnalité très agréable pour une personne voulant essayer notre shop puisqu’il est maintenant très facile de paramétrer la base de données au démarrage, en ne fournissant que le nom d'hôte du serveur, le nom d'utilisateur et le mot de passe.

## Protection contre les injections

Le shop est protégé contre les injections SQL par l'utilisation généralisée des filtres pour la lecture des variables superglobales POST et GET ainsi que par l'utilisation de requêtes préparées pour communiquer avec la base de données MySQL.

Le filtrage permet d'empêcher l'introduction de code PHP dans la logique du programme ainsi que la validation du type des données reçues (entier, chaîne de caractères, etc.). Par exemple, un identifiant sera la plupart du temps filtré vers un entier.

L'utilisation de requêtes préparées avec des paramètres empêche l'injection de code SQL en faisant clairement distinguer à la base de données la requête des données utilisées. En effet, ces dernières lui sont transmises de manière séparée. Cela neutralise par exemple l'effet d'éventuels caractères d'échappement introduits par un utilisateur malveillant et se charge d'échapper correctement les chaînes de caractères insérées.

## Architecture logique

L'architecture de l'application est peu élaborée. Le fichier index.php contrôle la majeure partie de la logique décisionnelle de l'application. C'est lui qui redirige et effectue les demandes de création de comptes, de modification d'articles, de création de la base de données, d'authentification, etc.

Le fichier index.php, après avoir exécuté sa logique, et s'il n'a pas redirigé vers une autre page, inclus alors head.php, body.php et foot.php.

head.php Affiche la barre en haut du shop avec certains boutons sous certaines conditions (la personne est authentifiée, elle a des droits d'administration, etc.). Le fichier foot.php affiche uniquement le pied de page et ne fait rien d'autre de particulier.

Tout comme index.php, body.php concentre une partie substantielle de la logique. Contrairement à index.php qui effectuera plutôt des opérations, body.php inclura d'autres fichiers pour modifier l'affichage en fonction de la requête (liste des articles, liste des comptes, des commandes, etc.).

Les fichiers inclus par body.php se chargent ensuite d'interpréter de petites modifications d'affichage en fonction de la requête (filtre sur une liste, commandes selon leur statut, etc.).

## Séparation Access et MySQL

Comme demandé dans le cahier des charges, l'authentification des utilisateurs se fait en utilisant une base de données Microsoft Access. Celle-ci possède une table contenant une liste de personnes et trois colonnes : le login, le hash du mot de passe et le salt correspondant. La base MySQL (cf. MLD) possède aussi une table de personnes, mais elle ne peut pas être utilisée pour authentifier une personne, car elle n'a aucune colonne pour le mot de passe. Le "lien" entre la partie de la personne stockée dans Access et celle dans MySQL et fait en utilisant le login de la personne.

Quand un utilisateur s'authentifie, la base Access est donc utilisée. Ensuite, quand il crée des commandes, commentaires, ou qu'il modifie certaines parties de son profil (nom, prénom, adresse, e-mail), c'est la base MySQL qui est utilisée. S'il modifie son mot de passe, ce sera à nouveau Access.

La base Access, en étant séparée de la partie "métier", agit donc un peu comme une source d'authentification externe qui se charge uniquement d'indiquer si une tentative d'authentification a échoué ou non.

## Modélisation SQL

En suivant les règles de Merise, le MCD a été dessiné pour modéliser conceptuellement les données du site. Il y a des personnes, qui sont d'un certain type (utilisateur, administrateur). Il y a aussi des articles qui sont en vente et sont d'une seule catégorie. Les personnes commandent des articles avec une quantité et à un certain prix. La commande résultante a une adresse de facturation et une de livraison qui correspondent à celles de la personne. La commande a aussi un statut. La personne peut aussi commenter les articles. Ces commentaires sont affichables ou non par l'administrateur.
Une fois qu'une commande a été faite, la modification du prix de vente d'un article ou d'une adresse de la personne n'aura pas d'effet sur celle-ci, car ces données ont été préalablement copiées dans les tables intermédiaires de relation.

Le cas des adresses, un problème intéressant
L'entité des adresses n'est pas normalisée à 100%. En effet, comme nous nous trouvons dans le cas d'un shop, il n'est pas important d'avoir une table séparée pour les localités, les rues ou encore les numéros postaux. Une telle granularité serait utile pour une entreprise telle que la poste ou une commune. Cependant ce n'est pas le cas ici, car il faut juste avoir une adresse de livraison et de facturation. Il pourrait même être envisageable de mettre l'adresse complète dans un champ texte avec des retours à la ligne, car c'est le client qui sait le mieux comment formater l'adresse correctement pour les services postaux de son pays. Il peut en effet avoir des différences entre pays, et maintenir un modèle de données prenant en compte une centaine de systèmes postaux ainsi que la logique dans l'application aurait un coût prohibitif.
Cependant, il aurait quand même fallu avoir le pays dans une table séparée et pré-remplie, cela parce que le pays doit être reconnaissable par les services postaux du pays d'envoi. En effet, avoir un nom écrit en arabe ou mandarin ne doit pas être très pratique pour la poste suisse, alors que le nom de la rue ou de la personne ne pose pas de problème, car c'est la poste de destination qui s'en charge.

Voici le modèle conceptuel de données :



Ci-après le modèle logique de données :



## Technique SQL

Le moteur utilisé est InnoDB, il nous permet de faire respecter, entre autre, les contraintes référentielles et de pouvoir utiliser des transactions.
Les chaînes de caractères sont encodées en UTF-8. Le jeu de caractères utf8mb4 est utilisé pour avoir un support complet d'UTF-8 par MySQL (cf. <http://dev.mysql.com/doc/refman/5.5/en/charset-unicode-utf8mb4.html>).

## Normes, standards et syntaxe du code

Afin d’avoir un code propre et « uniforme », nous avons décidé de se référer au site internet « PHP the right way » (<http://www.phptherightway.com/#code_style_guide>) .

Grace à cette méthodologie, nous avons pu mieux nous retrouver dans le code des autres et ainsi optimiser le temps passé lors les éventuels débogages.

Par exemple, pour la nomination des variables, nous avons choisi le « CamelCase » (une majuscule sépare les mots entre eux). Ces variables sont aussi déclarées en français.

L’encodage est bien entendu en UTF-8 pour éviter de s’embêter avec les accents et autres imprévus dans le texte.

## Références, aides et source

Les principaux sites qui nous le plus aidé pour construire ce projet ont été :

* PHP.net , la référence en matière de PHP
* StackOverFlow, qui nous a permis de résoudre certains problèmes survenus
* OpenClassRoom, site en français avec de bons tutoriels. Vous noterez également que la partie flux RSS est basée sur du code provenant de ce site.