22/03/2017

TPI fictif – Développement d’une application web

|  |  |
| --- | --- |
| Auteur et chef de projet | Philip Marshall |
| Supérieur professionnel | Steeve Droz |
| Client | Club nautique |
| Date de début de projet | 14/02/2017 |
| Date de fin de projet | 05/04/2017 |

Table des matières

[1 Introduction 3](#_Toc477868908)

[2 Documentation de développement 3](#_Toc477868909)

[2.1 Explications détaillées du projet 3](#_Toc477868910)

[2.2 Diagramme des cas d'utilisation 4](#_Toc477868911)

[2.3 Architecture du système 4](#_Toc477868912)

[2.4 Définition des conventions applicables 5](#_Toc477868913)

[2.5 Planning de livraison global 5](#_Toc477868914)

[2.6 Base de données 5](#_Toc477868915)

[2.7 Flux de Navigation 6](#_Toc477868916)

[3 Réalisation des cas d'utilisation 7](#_Toc477868917)

[3.1 Saisie d’une consommation 7](#_Toc477868918)

[3.1.1 Scénario 7](#_Toc477868919)

[3.1.2 Analyse du scénario 7](#_Toc477868920)

[3.1.3 Le planning de livraison 7](#_Toc477868921)

[3.1.4 La phase de programmation 8](#_Toc477868922)

[3.1.5 La phase de tests 8](#_Toc477868923)

[3.2 Saisie d’une consommation 8](#_Toc477868924)

[3.2.1 Scénario général 8](#_Toc477868925)

[3.2.2 Analyse du scénario 8](#_Toc477868926)

[3.2.3 Le planning de livraison 9](#_Toc477868927)

[3.2.4 La phase de programmation 10](#_Toc477868928)

[3.2.5 La phase de tests 10](#_Toc477868929)

[3.3 Gestion des factures 10](#_Toc477868930)

[3.3.1 Scénario général 10](#_Toc477868931)

[3.3.1.1 Scénario : Modification consommation 10](#_Toc477868932)

[3.3.1.2 Scénario : Suppression de consommation 11](#_Toc477868933)

[3.3.2 Analyse du scénario 11](#_Toc477868934)

[3.3.3 Le planning de livraison 12](#_Toc477868935)

[3.3.4 La phase de programmation 12](#_Toc477868936)

[3.3.5 La phase de tests 12](#_Toc477868937)

[3.4 Cas d'utilisation Superviseur 13](#_Toc477868938)

[3.4.1 Scénario 13](#_Toc477868939)

[3.4.1.1 Scénario : Modification utilisateur 13](#_Toc477868940)

[3.4.1.2 Scénario : Suppression d’utilisateur 13](#_Toc477868941)

[3.4.1.3 Scénario : Création d’un utilisateur 13](#_Toc477868942)

[3.4.2 Analyse du scénario 13](#_Toc477868943)

[3.4.3 Le planning de livraison 14](#_Toc477868944)

[3.4.4 La phase de programmation 15](#_Toc477868945)

[3.4.5 La phase de tests 15](#_Toc477868946)

[4 Mode d'emploi utilisateur 15](#_Toc477868947)

[4.1.1 Saisir une consommation + voir toutes les saisies - employé 15](#_Toc477868948)

[4.2 Modification d’une facture – caissier 17](#_Toc477868949)

[4.3 Gestion des utilisateurs 20](#_Toc477868950)

[4.3.1 Modification d’un utilisateur 20](#_Toc477868951)

[4.3.2 Suppression d’un utilisateur 21](#_Toc477868952)

[5 Problèmes rencontrés et solutions 22](#_Toc477868953)

[5.1 Bootstrap 22](#_Toc477868954)

[5.2 Sécurité 23](#_Toc477868955)

[6 Conclusion 24](#_Toc477868956)

[7 Annexes 24](#_Toc477868957)

[7.1 Journal de bord 24](#_Toc477868958)

[7.2 Cahier des charges 24](#_Toc477868959)

[7.3 Code source 24](#_Toc477868960)

[7.4 Références 24](#_Toc477868961)

# Introduction

Dans le cadre d’un entraînement au TPI, j’ai été mandaté à la gestion numérique des consommations dans une entreprise. Le club nautique XXX gère actuellement les consommations de ses employés grâce à des feuilles qu’ils remplissent en indiquant qui a consommé quoi. Le superviseur, celui qui s’occupe de rédiger la facture de chacun…

# Documentation de développement

Il s’agit d’une analyse préalable. Le but est de définir les objectifs que le programme doit remplir.

## Explications détaillées du projet

Les employés du club nautique remplissent sur une feuille, leurs noms, prénoms, puis mettent une croix dans la case qui correspond à ce qu’ils ont consommé. Chaque mois, le caissier regroupe les consommations par personne dans le but de connaître chaque mois toutes les consommations d’une personne. Facture qui sera envoyée à celle-ci.

Mon rôle dans ce projet a été d’informatiser ce système. Les fonctionnalités que j’ai créées sont les suivantes :

* Saisie de consommation - **tout le monde**
* Aperçu de « ma consommation » - **tout le monde**
* Aperçu des factures non payées - **caissiers et superviseurs (leur sert de « ma consommation »**
* Modification d’une consommation - **caissiers et superviseurs**
* Gestion des utilisateurs - **superviseur**

L’avantage en informatisant cette gestion des consommations est principalement d’éliminer le papier, de pouvoir trouver les informations qui nous intéressent plus rapidement et de permettre aux employés de ne pas devoir réécrire leurs noms et prénoms à chaque saisie car ils s’identifieront avec leurs numéros de badges sur la page d’accueil.

Pour la réalisation de cette application web, j’ai utilisé Firefox comme navigateur, puis développé en PHP, MySQL, puis HTML. Pour la mise en page, j’ai utilisé le framework Bootstrap car il m’a permis d’avoir de beaux éléments en très peu de temps ; cela m’a donc évité d’utiliser 5h de mon temps à faire du CSS qui, en soit, ne joue aucun rôle dans le fonctionnement de mon application. La raison pour laquelle j’ai choisi Firefox comme navigateur est parce que j’ai toujours développé dessus et que je me sens à l’aise avec. J’ai choisi ces langages car je maîtrise plutôt bien PHP, puis parce que je sais faire du MySQL. En ce qui concerne HTML et CSS, ce sont les langages les plus courants, même si j’ai utilisé Bootstrap pour mon CSS. Je ne voyais donc pas d’intérêt à chercher d’autres langages.

Le serveur web qui m’a été imposé est KSWEB. Pour des raisons de manque de temps à disposition, l’application sera finalement hébergée sur PC et, KSWEB n’était pas supporté par Windows, j’ai décidé d’utiliser Wamp64. Wamp est hébergé sur la machine faisant office de serveur et de client. L’application fonctionne donc en localhost.

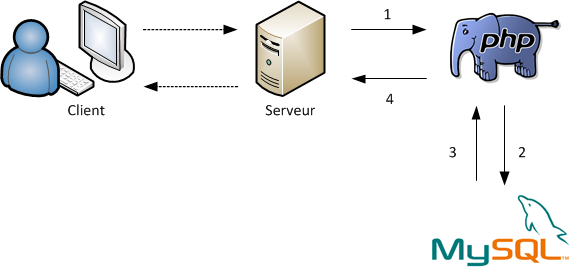
## Diagramme des cas d'utilisation

Les utilisateurs de mon application web possèdent différents droits en plus de celui de pouvoir se connecter, correspondants à des statuts. Voici une illustration des droits par types d’utilisateurs



## Architecture du système

Pour développer mon application, j’utilise plusieurs langages, dont le HTML, PHP et MySQL. Le serveur représente Apache si l’application est utilisée sur Windows ou KSWEB si elle est utilisée sur Android. Le client envoie ce qu’il veut via l’URL, le php fait ce que le client veut recevoir, si besoin il demande des infos au my sql, puis renvoie la page au client.



## Définition des conventions applicables

Chaque fichier contient des variables locales permettant l’envoi de valeurs à des fonctions qui exécutent des requêtes SQL, cependant l’application possède deux variables très importantes :

* $\_SESSION[‘user’][0 ou un] « 0 » contient une concaténation « nom » « prénom » de l’utilisateur connecté et « 1 » correspond à l’ID de cet utilisateur.
* $statut, correspond au statut hiérarchique des utilisateurs. Il peut valoir « emp » pour l’employé, « cai » pour le caissier ou « sup » pour le superviseur. Cet ID permet de gérer les droits d’acces aux différents fonctionnalités (saisie, factures, etc.).

## Planning de livraison global

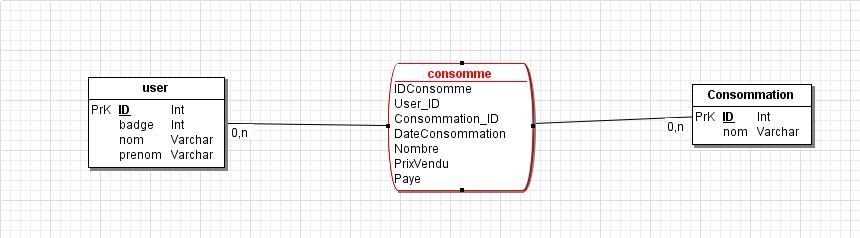
Mon objectif était de développer une fonctionnalité par jour. Le plus difficile a été de mettre en place les premières fonctionnalités comme la saisie, puis la gestion des factures, y compris leur modification. Je n’ai pas rencontré beaucoup de de difficulté pour la gestion des utilisateurs. Cette fonctionnalité ressemble beaucoup à celle des factures, à quelques détails près mais le principe est le même.

Un diagramme de GANTT présente en détail la planification de mon projet. Vous pouvez le trouver dans les annexes.

## Base de données

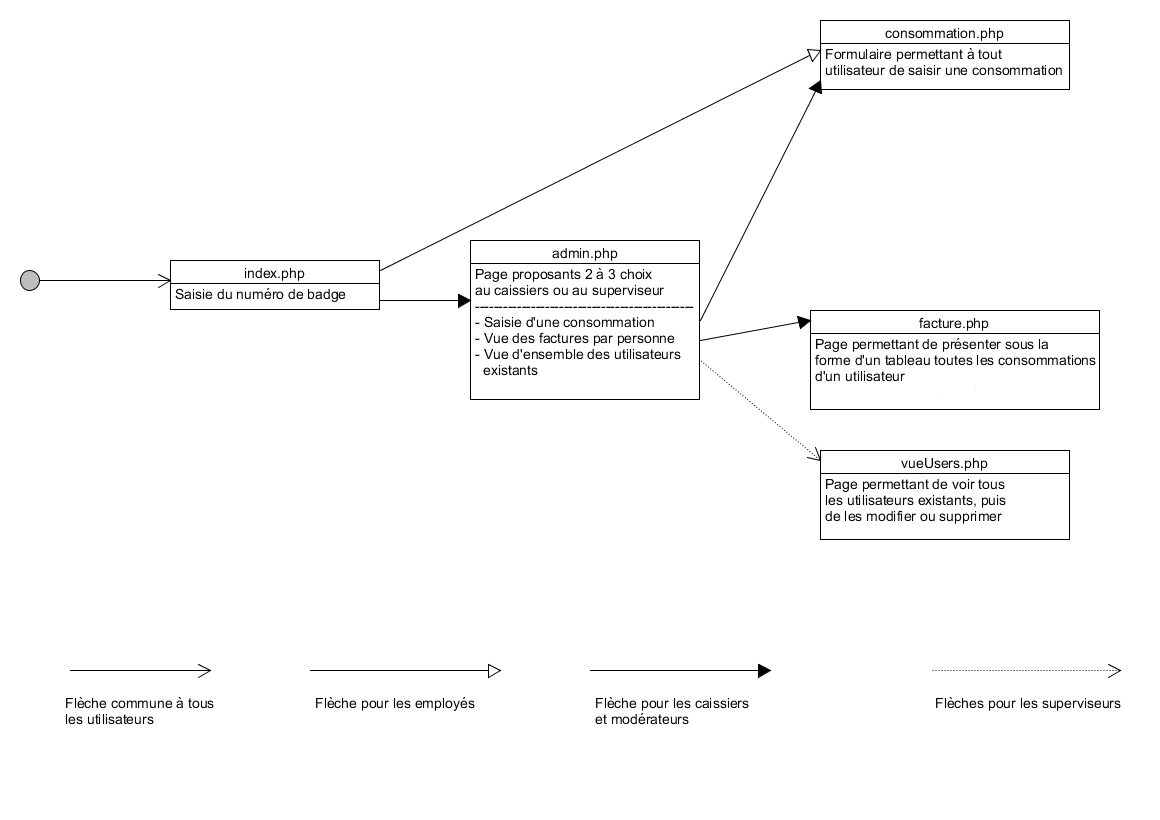
Ma base de données est hébergée sur mon localhost. On peut y accéder depuis phpMyAdmin.

Ci-dessous le model conceptuel de ma base de données :



## Flux de Navigation

Ci-dessous, le flux de navigation avec lequel nous pouvons constater qui peut faire quoi et en utilisant quel chemin :



# Réalisation des cas d'utilisation

## Saisie d’une consommation

Un employé souhaite se connecter

### Scénario

Voici maintenant un aperçu utilisateur/serveur :

1. L’employé souhaite se connecter sur l’application
2. Il ouvre firefox sur la tablette
3. La page web avec le formulaire de connexion s’ouvre
4. L’employé entre son numéro de badge et clique sur « valider »
5. Le système vérifie que l’employé existe et le redirige sur l’index correspondant à son statut ou le renvoie sur l’index où il pourra à nouveau essayer de se connecter

### Analyse du scénario

Nous nous trouvons sur l’index et l’utilisateur souhaite se connecter. Il entre son numéro de badge, puis clique sur le bouton de connexion. Le champ dans lequel il a entré son numéro de badge est un formulaire utilisant la méthode POST, comme tous les formulaires de l’application, qui appelle le fichier login.php, fichier appelé pour la connexion ainsi que pour la déconnexion. Login.php appelle la fonction login qui se trouve dans le fichier MySQL.php avec, comme argument, le numéro de badge entré précédemment.

Le principe de la fonction login est d’exécuter une requête permettant d’extraire l’utilisateur correspondant au numéro de badge entré. J’ai décidé de ne pas crypter le numéro de badge car il n’est pas censé être secret. La fonction peut retourner soit le nom, prénom, ID et le statut de l’utilisateur, soit false dans le cas où le numéro de badge saisi ne correspond à aucun utilisateur. Dans le cas où le numéro de badge est bien correct, la variable $connexion va contenir le tableau retourné par la fonction login et ne sera donc pas égal à false ce qui permet d’entrer dans le if qui sert à déclarer la variable de session décrite plus haut dans le rapport, puis déclare la variable de statut par rapport au statut de l’utilisateur qui vient de se connecter.

Afin de ne pas rester sur la page de login, toujours dans le if nous pouvons retrouver une redirection sur l’index correspondant au statut de l’utilisateur connecté. Cette redirection se trouve, elle aussi, dans un if qui détermine le statut de l’utilisateur.

Lorsque l’utilisateur se trouve sur l’index correspondant à son statut il peut constater dans l’en-tête de la page web qu’il y aura écrit « Connecté en tant que Nom Prénom ».

### Le planning de livraison

|  |  |
| --- | --- |
| Tâche | délais |
| Possibilité de se connecter | 1 jour |
|  |  |
|  |  |

### 

### La phase de programmation

Voir code en annexe

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Connexion | | | | |
|  | Connexion OK | Redirection index statut | OK |  |
| Saisie d’un numéro de badge erroné | Message d’erreur sur index connexion | OK |  |
|  | Saisie n° badge | 4 caractères max | OK |  |

## Saisie d’une consommation

L’employé Jimmy Hendrix souhaite saisir une consommation.

### Scénario général

* Jimmy se connecte à l’application
* Le système le redirige sur son index et affiche 2 options : Saisir et vue des dernières saisies
* L’employé clique sur « saisir »
* Le système le redirige sur le formulaire de saisie
* L’employé remplit les champs correspondant à ce qu’il a consommé, puis clique sur saisir
* Le système repère tous les champs qui ont été remplis, les enregistre sous la forme d’un tableau puis les envoie dans une requête SQL qui va les insérer dans la table consomme
* Le système réaffiche la page de saisie de consommation

### Analyse du scénario

La liste des consommations est présentée sous la forme d’un tableau HTML mis en forme avec bootstrap et le contenu de ce tableau est généré par une requête. Etant donné que le contenu de ces champs doit être récupérable, le tableau se trouve dans un formulaire. Le nom de chaque cellule ou consommation correspond à l’ID de celle-ci dans la table consommation, puis la valeur correspond au nombre de fois auquel la consommation a été prise.

Lorsque l’utilisateur a saisi puis validé ce qu’il a consommé, le formulaire redirige sur la page de saisie, le fichier redirige donc sur lui-même. Un if vérifie l’existance d’un input « valide » qui permet de savoir si le formulaire a déjà été validé ou pas. Nous ne traiterons que le cas où le formulaire a été validé.

Le formulaire est validé. Pour savoir ce que l’utilisateur vient de consommer, j’ai utilisé la fonction php « array\_filter » pour créer un tableau avec les consommations saisies avec leur quantité uniquement. Le but est d’utiliser la variable contenant ce tableau dans un foreach qui va s’occuper de remplir un tableau 2 dimensions contenant le nom du champ (ID), puis le nombre de fois consommé.

Une fois le tableau 2 dimensions ($concatConso) rempli, j’appelle la fonction SQL « SaisieConsommation ». En ce qui concerne la gestion d’erreur, les input dans le tableau sont de type int et de minimum 0 ce qui empêche les autres caractères que des nombres d’être saisis.

Avant d’exécuter la requête SQL, il faut l’écrire. Tout d’abord, il est important de savoir que une ligne dans la table « consomme » correspond à une consommation et que une consommation correspond à un article de consommation avec son nombre de fois consommé. Ceci veut dire que, si l’utilisateur a entré plusieurs consommations différentes, il faudra insérer plusieurs lignes.

Pour ceci, j’ai décidé de de séparer ma requête en 3 parties que j’assemble au dernier moment. La première partie ($debutSQL) contient le début de la requête jusqu’au mot « values ». La deuxième doit contenir les valeurs à insérer dans les champs entourés de parenthèses, exemple : (1,2,12/12/2017,10).

La requête doit insérer autant de lignes qu’il y a d’articles consommés, les values doivent donc être séparées par une virgule, puis être concaténées dans la variables « values » que j’ai prévue à cet effet.

La difficulté est, non seulement de générer ces parenthèses avec leur contenu, mais également de les séparer avec cette virgule. Une solution aurait pu être d’en mettre une à la fin de chaque values, mais si nous faisons comme cela, nous risquons de finir avec une de trop à la fin.

La solution que je propose est donc la suivante : J’ai créé puis affecté une variable « $i = 0 » qui me permettra de déterminer si je concatène la première ligne ou pas. Pour concaténer les données de saisie, j’ai mis en place un foreach qui s’occupe de la concaténation dont je parlais dans les paragraphes en-dessus. Ce foreach commence par une condition qui teste si $i = 0. Si la condition est respectée, cela veut dire que nous sommes dans le premier passage de la boucle et donc que la ligne qui attend d’être entrée dans la variable correspond à la première saisie. Après avoir affecté la variable de concaténation, j’ai affecté « $i = 1 » pour que la condition ne soit plus respectée. Dorénavant, les prochaines valeurs seront précédées d’une virgule.

Maintenant que les deux morceaux importants de la requête sont terminés, il ne manque plus que la troisième qui est un simple « ; ».

La dernière étape est de créer une variable « req » qui va contenir une concaténation des deux premières variables, puis du point-virgule qui n’est pas obligatoire, pour ensuite la préparer puis pour l’exécuter.

La requête est terminée et la saisie vient d’être insérée dans la table « consomme ».

### Le planning de livraison

|  |  |
| --- | --- |
| Tâche | délais |
| Générer tableau avec le bon contenu | 1 jour |
| Rendre saisie possible   * Envoi des données à la fonction SQL * Concaténation des values |

### La phase de programmation

Voir code en annexe

### La phase de tests

Expliquer les tests réalisés et le protocole de test. De plus faite TOUJOURS tester votre application par un utilisateur « lambda », il fera des manipulations que vous ne ferez jamais et ce sera bien ce type d’utilisateur qui utilisera votre application. Noter qui était le testeur et ce qu’il a découvert comme problème ou disfonctionnement.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Saisie d’une consommation zéro fois | Saisie pas prise en compte | OK | Je n’ai mis aucun message d’erreur |
| Possibilité de saisir un ou plusieurs articles de consommation | Ecriture des saisies dans la table « consomme » | OK |  |

## Gestion des factures

Le caissier Angus Young souhaite valider une facture qu’un employé vient de payer.

### Scénario général

Le caissier se connecte à l’application web.

* Le système affiche un page avec deux options. Le premier bouton le redirige sur la page de saisie de consommation donc nous avons parlée avant. Le deuxième bouton redirige sur une page dans laquelle il peut choisir la facture de l’employé qu’il souhaite voir. Par défaut, le système affiche les consommations de l’utilisateur connecté
* Le caissier ouvre la liste déroulante contenant le nom de tous les employés
* Le système contient déjà la liste des employés par ordre alphabétique
* Le caissier sélectionne l’employé dont il souhaite voir la facture
* Le système récupère le nom choisi, puis exécute une requête MySQL pour extraire toutes les consommations de l’employé en question, affiche le résultat sous la forme d’un tableau

#### Scénario : Modification consommation

Depuis le tableau contenant toute les consommations d’un employé durant le mois, le caissier a la possibilité de modifier ou de supprimer une saisie.

* Le caissier clique sur le bouton « modifier » dans la ligne qui correspond à la modification qu’il veut effectuer ;
* D’une façon ou d’une autre, le système permet la modification ;
* Le caissier valide sa modification ;
* Le système prend exécute une requête MySQL qui va modifier la ligne en question, puis redirige le caissier sur le tableau de consommation du même utilisateur qu’avant .

#### Scénario : Suppression de consommation

Suite à la demande d’un employé, le caissier souhaite supprimer une consommation.

* Le caissier clique sur le bouton « supprimer », à côté du bouton « modifier » ;
* Le système exécute une requête MySQL permettant de supprimer la ligne en question, puis une autre requête qui va reprendre les consommations de la personne en question pour les réafficher dans le même ordre qu’avant, mais sans la ligne supprimée.

### Analyse du scénario

Le caissier est connecté et se trouve sur l’index qui correspond à son statut. Les boutons affichés à l’écran (saisir, factures) sont contenus dans un formulaire qui, lorsqu’un bouton est cliqué, redirige sur la même page d’index. Un test détermine quel bouton a été cliqué en fonction du nom de celui-ci. Dans notre cas, l’option choisie est de voir les factures. J’ai donc fait une redirection sur la page des factures en n’oubliant pas de passer le statut en GET.

Le fichier « factures.php » s’occupe de l’affichage, puis de la modification d’une consommation.

Commençons par l’affichage des factures. Le principe pour afficher les consommations d’un utilisateur est de récupérer son ID que l’application récupère depuis la liste déroulante, puis de l’envoyer comme paramètre à une fonction SQL « facture » qui va extraire, à l’aide d’une requête, toutes les consommations que l’utilisateur choisi a saisies. La requête est un peu complexe car elle doit retourner un tableau contenant l’ID correspondant à sa consommation, donc celui de la table « consomme », l’ID de l’article qu’il a consommé, le nom de cet article ainsi que les autres champs de la table « consomme » et tout cela par rapport à l’ID de l’utilisateur sélectionné. La requête se trouve en annexe dans le code.

Un test vérifie si l’utilisateur vient d’ouvrir la page des factures ou s’il vient de sélectionner un utilisateur. Il faut savoir que la fonction détaillée dans le paragraphe précédent reçoit l’ID de l’utilisateur sélectionné ou connecté en paramètre. Si la page n’avait pas été ouverte avant, l’ID qui est envoyé à la fonction « facture » correspond à celui de l’utilisateur connecté. La liste des saisies qui s’affichera correspond donc à celles de l’utilisateur connecté.

Dans le cas où un utilisateur avait été sélectionné, le scénario serait le même, sauf que l’ID qui est envoyé à la fonction « facture » correspondrait à celui de l’utilisateur sélectionné.

Pour modifier une consommation, l’utilisateur clique sur « modifier », tout à droite du tableau. Ce lien modifier est contenu dans une balise <a> et redirige, sur l’événement click, sur une autre page « saisieModif.php » avec comme paramètres GET, le statut, ID de la consommation (de la table « consomme »), l’ID de l’article consommé, puis l’ID de la table retournée par la requête des factures. Autrement dit, cet ID correspond à celui du tableau affiché sur la page.

Le fichier « saisieModif » récupère les variables passées en GET, teste si le fichier est appelé pour la première fois, puis affiche le résumé de la consommation. Ce résumé contient La date de consommation, puis l’article consommé avec le nombre de fois qu’il a été consommé qui est modifiable. Une fois cette valeur modifiée et le formulaire validé, le formulaire appelle à nouveau la même page. Le test vu précédemment vérifie que le formulaire a bien été saisi, affecte la valeur entrée à la variable « $conso », puis appelle la fonction « editConsom » du fichier « MySQL.php » avec « $conso » en paramètre, ainsi que l’ID de la table « consomme », puis le statut.

La fonction SQL exécute la requête de modification qui modifie la consommation par rapport aux paramètres envoyés. Une fois la requête exécutée, l’utilisateur est redirigé sur la page des factures et revoit ses propres saisies.

### Le planning de livraison

|  |  |
| --- | --- |
| Tâche | délais |
| Affichage des utilisateurs dans la liste déroulante | 1 jour |
| Affichage des saisies |
| Modification d’une saisie |

### La phase de programmation

Tout le code se trouve en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Affichage des saisies de l’utilisateur connecté par défaut | Tableau présentant les saisies de l’utilisateur connecté | OK |  |
| Modification d’une saisie   * Impossible d’entrer une valeur autre que des chiffres ou inférieure à 1 | Modification prise en compte que si critère respecté | OK |  |

## Cas d'utilisation Superviseur

### Scénario

Le superviseur souhaite voir tous les utilisateurs existants dans le système. Il se connecte en entrant son numéro de badge.

* Le système affiche une page contenant trois choix : Saisir une consommation, affichages des factures par employé, puis affichage de tous les employés ;
* Le superviseur clique sur le bouton pour voir la liste de tous les utilisateurs ;
* Le système exécute une requête qui va retourner une liste de tous les utilisateurs dans l’ordre alphabétique. La liste est affichée sous la forme d’un tableau et le superviseur a la possibilité d’ajouter, de modifier et de supprimer un utilisateur ;

#### Scénario : Modification utilisateur

Depuis le tableau contenant tous les utilisateurs le superviseur a la possibilité d’en modifier ou d’en supprimer un.

* Le superviseur clique sur le bouton « modifier » dans la ligne qui correspond à la modification qu’il veut effectuer ;
* D’une façon ou d’une autre, le système permet la modification ;
* Le caissier valide sa modification ;
* Le système prend exécute une requête MySQL qui va modifier la ligne en question, puis redirige le superviseur sur le même tableau qu’avant.
* de consommation du même utilisateur qu’avant.

#### Scénario : Suppression d’utilisateur

Suite à la demande d’un employé, le superviseur souhaite supprimer un utilisateur.

* Le superviseur clique sur le bouton « supprimer », à côté du bouton « modifier » ;
* Le système exécute une requête MySQL permettant de supprimer la ligne en question, puis une autre requête qui va reprendre les consommations de la personne en question pour les réafficher dans le même ordre qu’avant, mais sans la ligne supprimée.

#### Scénario : Création d’un utilisateur

Le club nautique vient d’engager un nouvel employé. Le superviseur doit créer son profil dans le système.

* Le superviseur clique sur le bouton « Ajouter » en-dessus du tableau
* Le système le redirige sur une page avec un formulaire présenté dans un tableau
* Le superviseur remplit les champs « nom », « prénom », etc., puis valide
* Le système exécute une requête, puis crée le nouveau profil

### Analyse du scénario

L’utilisateur vient d’entrer dans la page de gestion d’utilisateurs. L’application appelle la fonction « extractUsers » depuis « gerer.php » avec, comme paramètre, la chaîne de caractère « all » qui signifie que l’on souhaite extraire tous les utilisateurs. Une condition teste cette chaîne puis exécute la requête voulue. Une fois la requête exécutée, une variable en reprend le résultat, puis le retourne au fichier « gerer.php ». Une boucle foreach s’occupe d’afficher, sous la forme du même tableau que celui des factures, les utilisateurs existants dans l’application.

Lorsque le superviseur clique sur « modifier », l’application le redirige sur la page de modification d’utilisateur avec les paramètres GET : « statut » et l’ID de l’utilisateur. Une fois arrivé dans la page de modification d’utilisateur, le fichier « modifUser.php » récupère les variables passées en GET, puis rappelle la fonction « extractUsers » avec l’ID de l’utilisateur que le superviseur veut modifier. La fonction exécute la requête qui ne retourne que l’utilisateur que l’on veut modifier.

Une fois le tableau contenant un utilisateur retourné, un foreach affiche sous la forme du même tableau que toute à l’heure, les données de l’utilisateur écrits dans des champs texte, tous modifiables. Le statut hiérarchique de l’utilisateur se trouve dans une liste déroulante et le statut affiché correspond, par défaut, à celui de l’utilisateur que le superviseur souhaite modifier.

Le superviseur vient de modifier l’utilisateur, puis de valider sa modification. Le formulaire appelle à nouveau la même page, une condition vérifie que le formulaire a bien été validé, puis récupère les variables passées en POST pour ensuite les envoyer à la fonction SQL « editUser ». La fonction « editUser » teste tout d’abord si le numéro de badge entré existe déjà ou pas et, s’il n’existe pas, la requête de modification s’exécute.

Si le superviseur clique sur « supprimer », dans le cas où il souhaite supprimer un utilisateur, le lien « supprimer » le redirige sur la page « deleteUser.php » avec « statut » et l’ID de l’utilisateur qu’il veut supprimer en paramètre. Le fichier cible récupère les variables passées en GET et appelle la fonction « delUser » dans « MySQL.php ».

La fonction « delUser » exécute une requête permettant de supprimer l’utilisateur choisi par le superviseur, pus redirige sur la page de gestion d’utilisateur avec le statut en paramètre.

Si l’utilisateur souhaite créer un utilisateur, celui-ci est redirigé sur la page de création d’utilisateur, entre les données correspondant au nouveau venu, puis valide le formulaire. Les données saisies sont transmises à la fonction « addUser » qui vérifie si le numéro de badge entré n’existe pas. Si le numéro de badge est valide, la fonction exécute la requête de création d’utilisateur, puis redirige sur la page « gerer.php ». Le superviseur peut constater que le nouvel utilisateur vient d’être créé puisqu’il se trouve dans le tableau contenant tous les utilisateurs.

Dans le cas où le numéro de badge saisi existait déjà, la requête de création d’utilisateur ne s’exécute pas et la fonction retourne un message d’erreur qui s’affiche en-dessous du champ « badge » expliquant que celui-ci existe déjà.

Si le superviseur souhaite modifier son statut, une variable « $admin » se met à true ce qui affiche le son statut en texte plutôt que dans une liste déroulante. Le superviseur n’a donc pas le droit de modifier son propre statut.

### Le planning de livraison

|  |  |
| --- | --- |
| Tâche | délais |
| Affichage de tous les utilisateurs dans le tableau | 2 jours |
| Possibilité de modification d’utilisateur |
| Possibilité de suppression d’utilisateur |
| Possibilité de création d’utilisateur |

### La phase de programmation

Programmation en annexe.

### La phase de tests

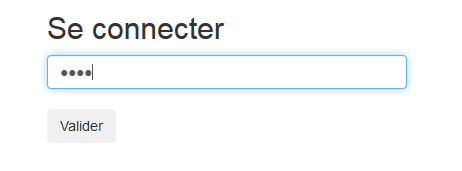
Expliquer les tests réalisés et le protocole de test. De plus faite TOUJOURS tester votre application par un utilisateur « lambda », il fera des manipulations que vous ne ferez jamais et ce sera bien ce type d’utilisateur qui utilisera votre application. Noter qui était le testeur et ce qu’il a découvert comme problème ou disfonctionnement.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Possibilité d’ajouter/modifier un utilisateur   * Saisie d’un numéro de badge existant | Création/modification impossible | OK | Possibilité d’entrer des caractères spéciaux dans numéro de badge |

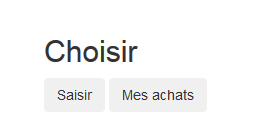
# Mode d'emploi utilisateur

### Saisir une consommation + voir toutes les saisies - employé

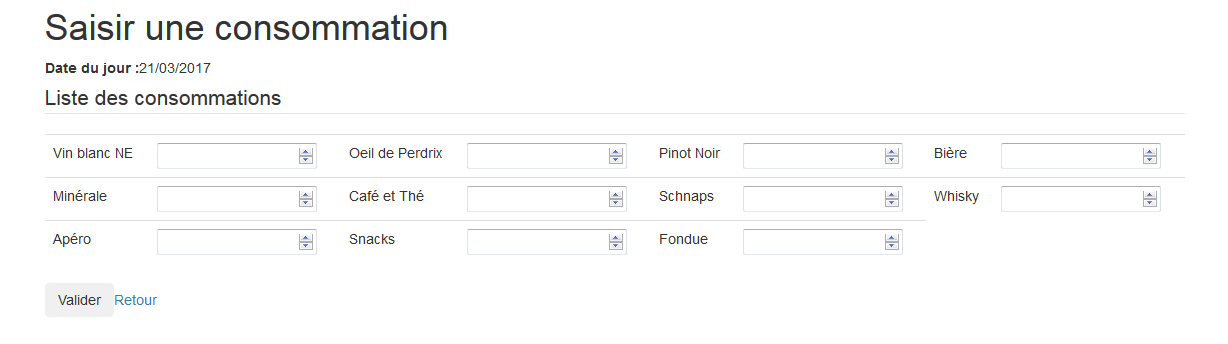
L’employé Angus Young souhaite saisir dans l’application ce qu’il vient de consommer. Pour cela, il se connecte en entrant son numéro de badge :



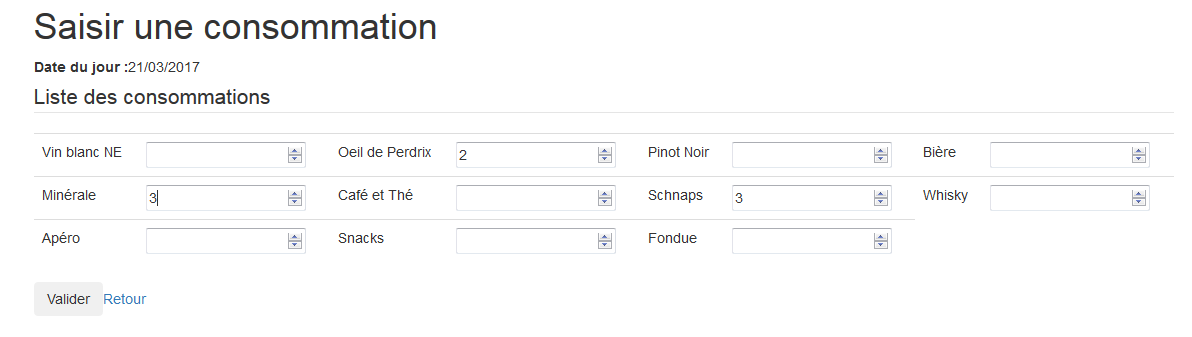
Après avoir cliqué sur « Valider », il est redirigé sur la page suivante :



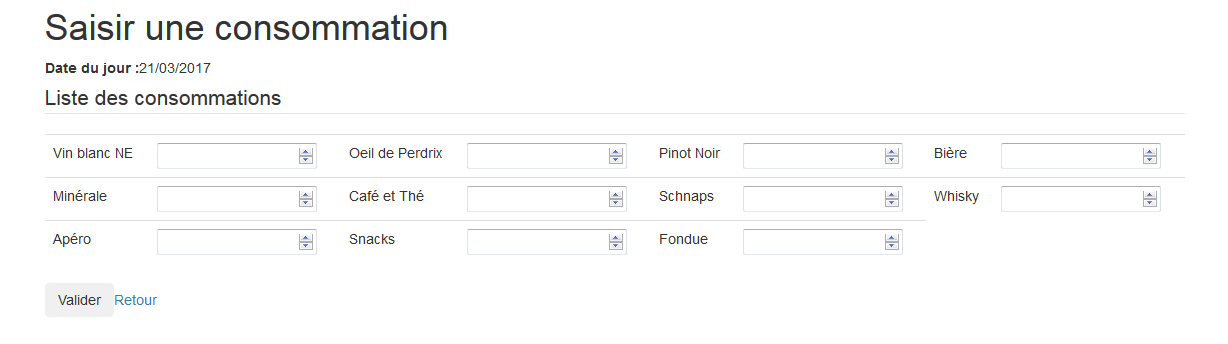
Comme Angus souhaite saisir sa consommation, il clique sur « Saisir », puis est redirigé sur le formulaire suivant :



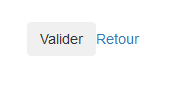
Il saisir sa consommation, soit en insérant des chiffres dans les champs, soit en cliquant sur les petites flèches sur le bord de ceux-ci :



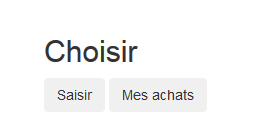
Nous pouvons constater qu’Angus vient de consommer 3 minérales, 2 « œil de Perdrix » et 2 schnaps. Il valide sa consommation et retrouve sur formulaire vide, comme ceci :



Notre employé clique sur « retour » en bas à gauche, à côté de « valider » :



Angus est redirigé sur sa page d’accueil, rappel :



Maintenant que l’employé Angus Young vient de valider sa saisie, celui-ci souhaite voir toutes ses saisies. Il clique alors sur « Mes achats » :



On constate qu’Angus a consommé 3 articles le 21 mars 2017 qui sont 3 schnaps, 3 minérales et 3 « œil de Perdrix ». Ces 3 consommations correspondent à celles qui viennent d’être saisies et ce sont les seules que l’employé à saisies depuis que son compte existe.

## Modification d’une facture – caissier

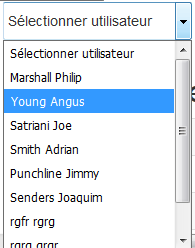
Pour modifier une facture ou consommation, le caissier doit se connecter à son compte, puis arrive sur l’index suivant :



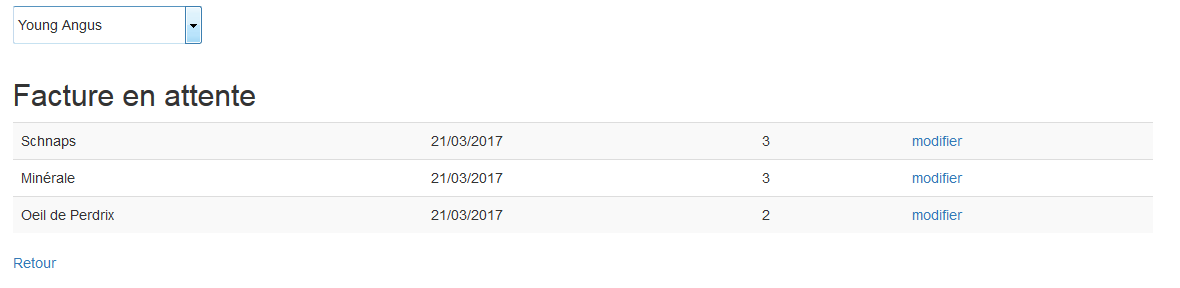
Il doit cliquer sur « Factures », puis est redirigé sur la page suivante. Les consommations affichées par défaut sont celles de l’utilisateur connecté, donc les siennes :



Si, par exemple, le caissier souhaite modifier une consommation d’Angus Young, il se rend dans la liste déroulante :



Puis sélectionne « Young Angus ». Le caissier peut à présent voir les saisie de l’employé qu’il vient de choisir :



Admettons qu’Angus Young se soit est trompé dans sa saisie et qu’il a consommé 2 schnaps au lieu de 3. Le caissier doit cliquer sur « modifier », puis arrive sur la page suivante :



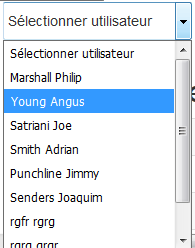
Le caissier change « 3 » par « 2 » :

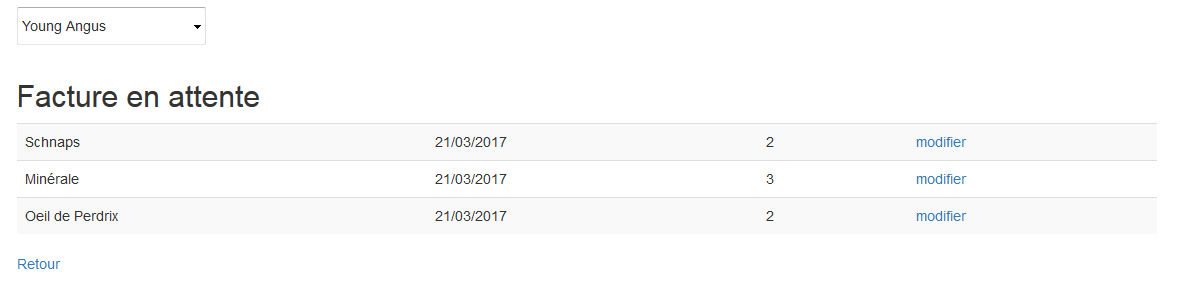


Il valide en cliquant sur « valider », puis est redirigé sur la page des factures avec ses propres consommations par défaut.



Il retourne sur l’utilisateur « Angus Young » :

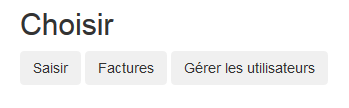




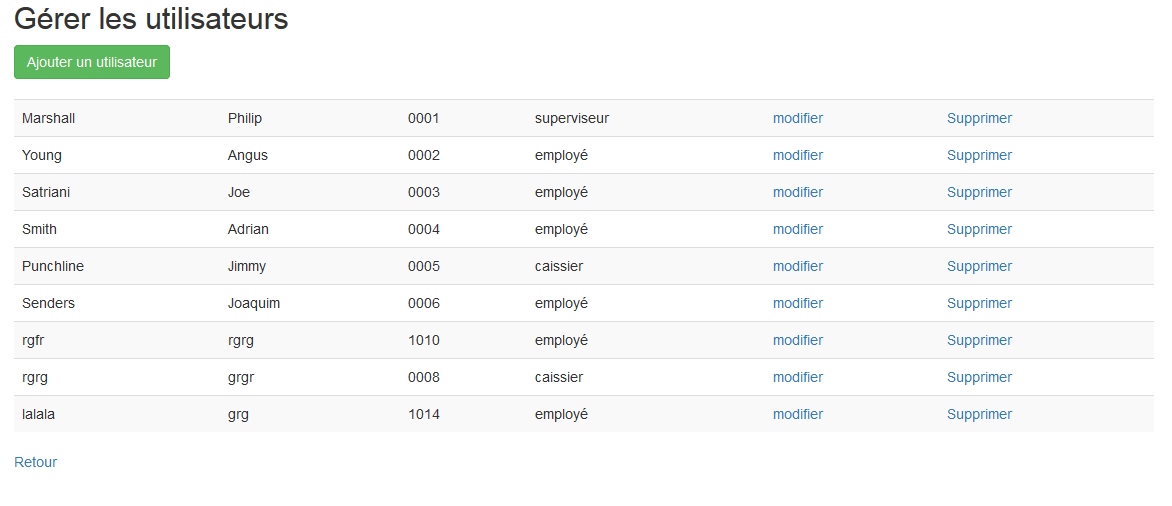
Nous pouvons à présent constater que la modification a bien été prise en compte.

## Gestion des utilisateurs

Le superviseur souhaite voir tous les utilisateurs existants sur l’application. Il se connecte et clique sur « Gérer les utilisateurs » :

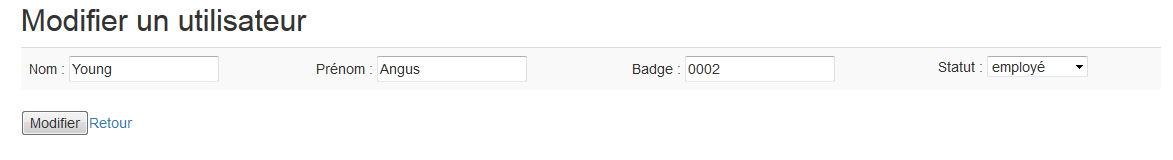


Le superviseur est redirigé sur la page suivante et peut directement voir la liste des utilisateurs :

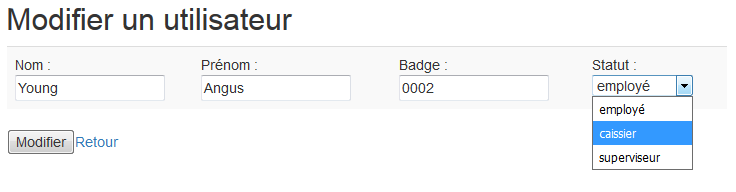


### Modification d’un utilisateur

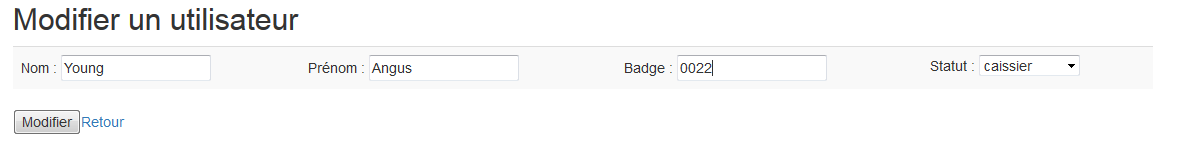
Le superviseur, Philip Marshall souhaite modifier le statut d’Angus Young pour le passer en caissier. Philip clique sur « modifier », puis est redirigé sur la page suivante :



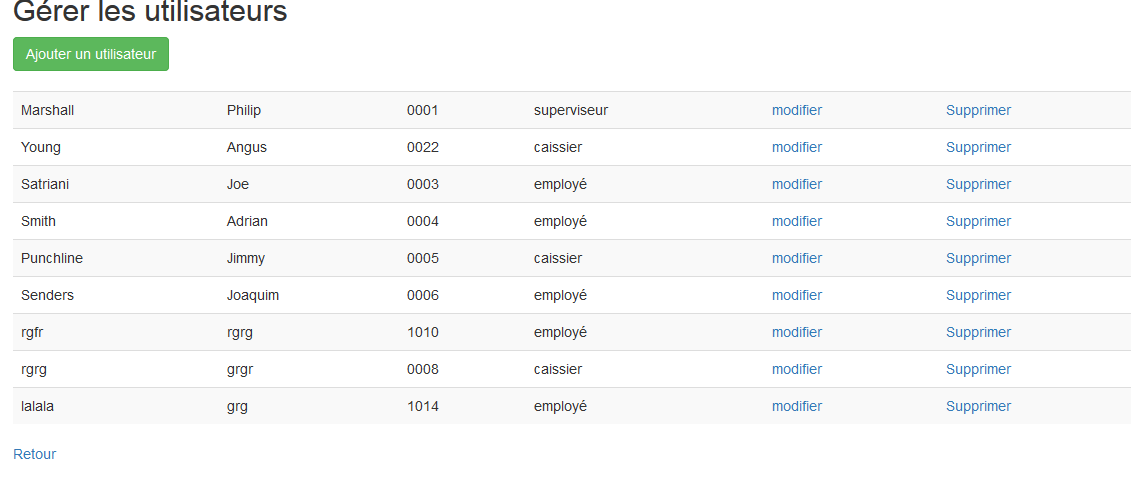
Philip peut à présent modifier tous les champs qu’il souhaite mais ne veut modifier que le statut. Il ouvre la liste déroulante à côté de « statut » :



Il sélectionne « caissier » et change le numéro de badge en « 0022 » :



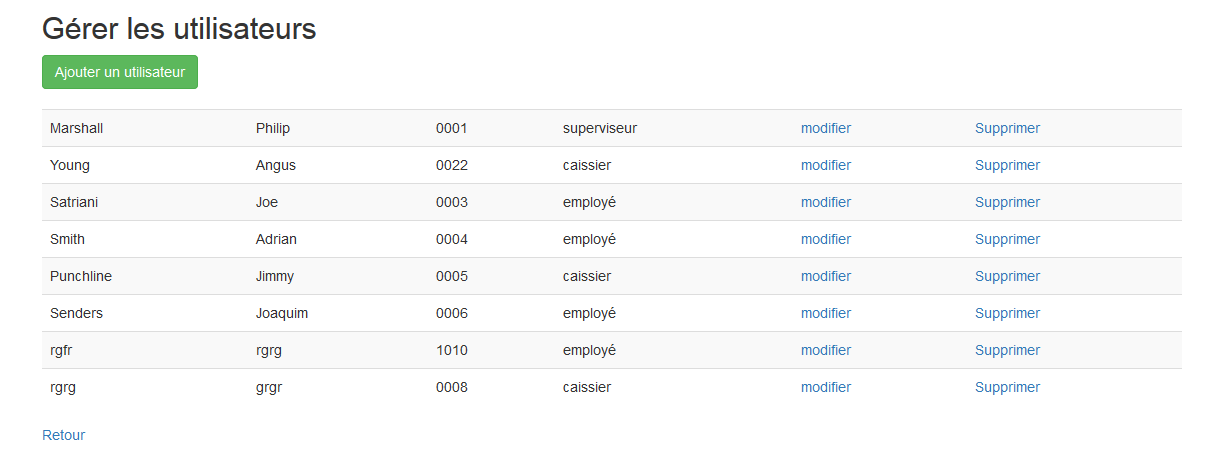
Après avoir cliqué sur « modifier », Philip est redirigé sur la page de gestion d’utilisateurs et constate que la modification a bien eu lieu :



### Suppression d’un utilisateur

Supprimer un utilisateur est certainement la chose la plus simple à faire de l’application. Il suffit de cliquer sur « supprimer » sur la ligne de l’utilisateur que le superviseur souhaite supprimer.

Admettons que l’utilisateur « lalala » ne fait plus partie du club nautique ou que pour une raison x ou y, son compte n’a plus lieu d’exister dans l’application ; Philip, depuis la même page, clique sur « supprimer » :



Philip constate que l’utilisateur « lalala » n’existe plus dans la liste des utilisateurs.

# Problèmes rencontrés et solutions

## Bootstrap

J’ai rencontré quelques petits souci été l’intégration de bootstrap dans mon application web. Etant donné que mon application gère plusieurs fonctionnalités, j’ai dû créer des fichiers pour les gérer. Certaines fonctionnalités redirigent même sur d’autres pages.

Afin de m’y retrouver plus facilement, j’avais créé un répertoire par fonctionnalité. Pour lier toutes les pages entre elles, toutes incluent le fichier « head.php » qui permet à l’utilisateur de se déconnecter et qui affiche « Connecté en tant que Prénom Nom » dans l’en-tête. Le fichier « head.php » s’occupe également de joindre la feuille de style bootstrap et comme tous les autres fichiers incluent « head.php », ils héritent de la feuille bootstrap.

Le fait d’inclure le fichier « head.php » a comme conséquence d’avoir le contenu de ce fichier dans le fichier que l’on visite acutellement. Si l’on regarde le code source avec firefox, nous pouvons voir le lien utilisé pour lier bootstrap. Voici un screenshot pour illustrer mes propos :



Comme nous pouvons le constater, le fait d’inclure le fichier « head.php » dans « indexAdmin.php » dans notre cas, copie le contenu html du « head.php » dans l’ « indexAdmin.php ». Tous les fichiers ne se trouvaient pas dans le même répertoire, il fallait donc un chemin parfois différent accéder à bootstrap, sachant également que l’ « index.php » pour se connecter se trouve dans le même répertoire que le « head.php ».

La solution pour laquelle j’ai opté a été de mettre tous les fichiers de toutes les fonctionnalités dans le même répertoire. De cette façon, le chemin d’accès à « bootstrap.css » est le même pour tous les fichiers.

## Sécurité

Lors de mes test, je me suis rendu compte que n’importe qui pouvait, sans se connecter, accéder à n’importe quelle page de l’application via l’URL. Pour régler ce problème, j’ai d’abord mis une condition en début de chaque fichier (sauf l’index de connexion) qui teste si un utilisateur est bien connecté ou non. Si personne n’est connecté, l’utilisateur est redirigé sur la page d’accueil pour se connecter.

Après avoir réglé cette faille de sécurité, je me suis rendu compte qu’un employé pouvait très bien accéder à toutes les pages des administrateurs, de nouveau via l’URL. Pour empêcher cela, j’ai ajouté une deuxième condition qui teste si la session de l’utilisateur existe et si son statut (que j’ai mis dans la variable de session car je l’ai extrait lors de la connexion) correspond bien au statut nécessaire pour accéder à la page où il souhaite se rendre.

# Conclusion

C’est la première fois depuis longtemps que j’ai eu à réaliser un projet en très peu de temps. Je n’ai pas pu bénéficier des 110 périodes que j’aurai lors de mon vrai TPI car je suis retourné en mi-février en atelier au CPLN. Je n’ai finalement eu qu’une septantaine de périodes pour

fournir le meilleur travail possible avant les vacances de Pâques.

//REVOIR PARAGRAPHE ET ESSAYER D EN PARLER AVANT

J’ai été beaucoup aidé par M. Droz lors de la phase d’analyse, puis par M. Lorenzin pour la modélisation de ma base de données ainsi que pour de l’algorithme. M. Droz m’a également présenté le framework « codeIgniter » qui permet de gagner énormément de temps dans le développement PHP. J’en ai tiré la conclusion que, si je maîtrisais bien ce framework, je pourrais faire en une journée le même développement qu’en 3 semaines de PHP. J’ai fait quelques tests à la maison et j’ai finalement décidé de faire le code PHP à la main car j’aurais certainement pris plus de temps à maîtriser le framework qu’à coder à la main.

//PLANNING

Le cahier des charges de base était trop gros pour le temps que j’avais à disposition. Je l’ai donc allégé en me passant de certaines choses : l’application était censée être hébergée en localhost sur une tablette samsung utilisant KSWEB comme serveur web. J’ai finalement développé sur WAMP64 j’ai mis de côté la tablette samsung. Une autre fonctionnalité que je n’ai pas développée était la possibilité de sauvegarder la base de données, puis de la restaurer.

# Choses à améliorer

Premièrement, je vais parler de ma façon de travailler. Au début, j’avais prévu de développer une fonctionnalité, puis l’écrire dans le rapport. De cette façon, je n’aurais eu que la conclusion ainsi que le guide utilisateur à rédiger à la fin du projet. J’ai quand-même avancé mon rapport au début du projet mais la raison pour laquelle j’ai décidé d’en rédiger la plupart à la fin était pour avoir le temps de développer un maximum de fonctionnalités sur mon application. Pour le vrai TPI, je tâcherai de plus équilibrer la rédaction du rapport et le développement de mon application.

En ce qui concerne les fonctionnalités, il faudrait améliorer plusieurs éléments :

* Possibilité de valider ou de supprimer des saisies
  + Eventuellement afficher 2 tableaux, l’un avec les factures validées, l’autres avec les factures en attente et la validation pourrait se faire en tirant la ligne du tableau dans l’autre tableau. Le développement d’une telle fonctionnalité nécessiterait du javascript que je ne maîtrise pas
* Modifier une saisie/un utilisateur directement depuis le tableau : cela permettrait de ne plus avoir de la page de modification de saisie/utilisateur
* Demander confirmation avant de supprimer un utilisateur
* Afficher une boîte de dialogue pour confirmer une saisie ou une quelconque modification d’utilisateur ou de saisie
* Changer la façon de traiter les saisies
  + Un utilisateur peut saisir plusieurs consommations à la fois, mais elles sont toutes affichées séparément. Une solution aurait pu être de les afficher ensemble sur une date.
* Améliorer la mise en page des index principalement
* Mise en place de la sauvegarde comme dans le cahier des charges de base
* Migration de l’application sur KSWEB hébergé sur une tablette samsung

# Annexes

## Journal travail

Pour retrouver mon journal de travail, veuillez vous rendre sur le lien suivant : <http://devinter.cpln.ch/lorenzin/index.php?title=JDT_MP#Semaine_du_13_au_15_mars>

## Code source

Voici mon code source organisé par fichier

## Sources externes

Communication client – serveur PHP - SQL

<https://www.google.ch/search?q=communication+client+serveur+php&client=firefox-b-ab&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj5n63c4ufSAhXoAcAKHe1YD20Q_AUIBigB&biw=1920&bih=945#imgrc=3c6D43zHVfp2EM>:

## Cahier des charges