

Architecture applicative de l'application Web

« GSB-AppliFrais-MVC»

ARCHITECTURE APPLICATIVE DE L'APPLICATION WEB « GSB-APPLIFRAIS-MVC»	1
Remarques préalables.....	1
Un développement guidé par les cas d'utilisation.....	1
Fonctionnement de l'application.....	6
Utilisation d'une classe d'accès aux données.....	8

Remarques préalables

Nous ne présentons pas ici les avantages de la structuration du code relevant de l'architecture Modèle-Vue-Contrôleur ; de nombreux documents se penchent sur la question.

Des frameworks (Zend, Symfony, PhpCake) fournissent les classes mettant en œuvre cette technologie. Nous avons fait le choix ici de faire le travail « à la main » ; la dimension didactique nous a guidé (enfin, on l'espère).

Nous nous bornerons à préciser certains choix faits. Ces choix se sont inspirés de deux travaux :

- Un document de **Serge Tahé** (2004-2008) qui présente une implémentation détaillée de l'architecture MVC pour php (en mode procédural -sans objet-) : <http://tahe.developpez.com/web/php/mvc/>
- Le travail d'**Olivier Cappuozzo** autour du contexte *festival* : <http://www.reseaucerta.org/cotecours/cotecours.php?num=363>

Un développement guidé par les cas d'utilisation

C'est le propre de l'architecture MVC ; le système (l'application) doit répondre aux sollicitations de l'utilisateur. Les cas d'utilisation sont les moyens textuels de décrire ces sollicitations et les réponses.

Prenons l'exemple du cas d'utilisation de la page suivants :

PROJET : Application web de gestion des frais	Description cas d'utilisation
--	--------------------------------------

Nom cas d'utilisation : Renseigner fiche de frais
Acteur déclencheur : Visiteur médical
Pré conditions : Visiteur médical authentifié
Post conditions : néant
<p>Scénario nominal :</p> <ol style="list-style-type: none"> <i>L'utilisateur demande à saisir un ou plusieurs frais pour le mois courant.</i> Le système retourne les frais actuellement saisis – éléments forfaitisés et hors forfait - pour le mois courant. <i>L'utilisateur modifie une ou des valeurs des frais au forfait et demande la validation.</i> Le système enregistre cette ou ces modifications et retourne ces valeurs à jour. <i>L'utilisateur ajoute un nouveau frais hors forfait en renseignant les différents champs – date d'engagement, libellé, montant - et valide.</i> Le système enregistre la ligne de frais hors forfait.
<p>Exceptions :</p> <p>2.a- C'est la première saisie pour le mois courant. Si ce n'est pas encore fait, le système clôt la fiche du mois précédent et crée une nouvelle fiche de frais avec des valeurs initialisées à 0. Retour à 3.</p> <p>4.a. Une valeur modifiée n'est pas numérique : le système indique 'Valeur numérique attendue '. Retour à 3.</p> <p>6.a Un des champs n'est pas renseigné : le système indique : 'Le champ date (ou libellé ou montant) doit être renseigné'.</p> <p>6.b La date d'engagement des frais hors forfait est invalide : le système indique 'La date d'engagement doit être valide'. Retour à 5.</p> <p>6.c La date d'engagement des frais hors forfait date de plus d'un an. Le système indique 'La date d'engagement doit se situer dans l'année écoulée'. Retour à 5.</p> <p><i>7. L'utilisateur sélectionne un frais hors forfait pour suppression.</i></p> <p>8. Le système enregistre cette suppression après une demande de confirmation.</p>
Contraintes :

L'utilisateur sollicite à 4 reprises le système (points 1, 3, 5 et 7 en **italique gras**). Le contrôleur (fichier spécifique) doit donc répondre à ces 4 sollicitations :

```

21  $action = filter_input(INPUT_GET, 'action', FILTER_SANITIZE_STRING);
22  switch ($action) {
23  + case 'saisirFrais': {...5 lines }
28  + case 'validerMajFraisForfait': {...9 lines }
37  + case 'validerCreationFrais': {...17 lines }
54  + case 'supprimerFrais': {...4 lines }
58  }
59  $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur, $mois);
60  $lesFraisForfait = $pdo->getLesFraisForfait($idVisiteur, $mois);
61  require 'vues/v_listeFraisForfait.php';
62  require 'vues/v_listeFraisHorsForfait.php';

```

Remarque : le code des cases a été plié ici pour se concentrer sur l'essentiel.

Pour chacune des sollicitations, le système réagit et agit en conséquence, par exemple pour la demande de saisie des frais :

```

21  $action = filter_input(INPUT_GET, 'action', FILTER_SANITIZE_STRING);
22  switch ($action) {
23  + case 'saisirFrais':
24  +     if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
25  +         $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
26  +     }
27  +     break;
28  + case 'validerMajFraisForfait': {...9 lines }
37  + case 'validerCreationFrais': {...17 lines }
54  + case 'supprimerFrais': {...4 lines }
58  }
59  $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur, $mois);
60  $lesFraisForfait = $pdo->getLesFraisForfait($idVisiteur, $mois);
61  require 'vues/v_listeFraisForfait.php';
62  require 'vues/v_listeFraisHorsForfait.php';

```

Le système teste (ligne 9) si c'est la première fois que l'utilisateur accède à cette demande de saisie de frais –cf extension 2.a- et va chercher en base (lignes 42-43) les données concernant les frais forfaitisés et non forfaitisés afin d'afficher les deux vues demandées (lignes 44-45). Ici, ces affichages sont communs aux autres cases.

Les deux vues affichées sont ici :

Renseigner ma fiche de frais du mois 08-2017

Éléments forfaitisés

Forfait Etape

12

Frais Kilométrique

562

Nuitée Hôtel

6

Repas Restaurant

25

Ajouter

Effacer

Vue « Liste des frais
au forfait »

Descriptif des éléments hors forfait			
Date	Libellé	Montant	
12/08/2017	Achat de fleurs	29.90	Supprimer ce frais
14/08/2017	Taxi	32.50	Supprimer ce frais

Nouvel élément hors forfait

Date (jj/mm/aaaa):

Libellé

Montant :

 €

Ajouter

Effacer

Vue « Liste des frais hors forfait »

Nous avons fait le choix de présenter deux vues distinctes – nous aurions pu bien sûr mettre ce code dans un seul fichier- pour éventuellement réutiliser une de ces vues dans un autre cas d'utilisation.

Dans cette architecture, l'affichage des vues est provoqué par un ordre **include** (ou require) *nomVue*.

Pour respecter l'indépendance des couches (vue, modèle), le modèle (fichier php) retourne des tableaux :

```

169 public function getLesFraisForfait($idVisiteur, $mois)
170 {
171     $requetePrepare = PdoGSB::$monPdo->prepare(
172         'SELECT fraisforfait.id as idfrais, '
173         . 'fraisforfait.libelle as libelle, '
174         . 'lignefraisforfait.quantite as quantite '
175         . 'FROM lignefraisforfait '
176         . 'INNER JOIN fraisforfait '
177         . 'ON fraisforfait.id = lignefraisforfait.idfraisforfait '
178         . 'WHERE lignefraisforfait.idvisiteur = :unIdVisiteur '
179         . 'AND lignefraisforfait.mois = :unMois '
180         . 'ORDER BY lignefraisforfait.idfraisforfait'
181     );
182     $requetePrepare->bindParam(':unIdVisiteur', $idVisiteur, PDO::PARAM_STR);
183     $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
184     $requetePrepare->execute();
185     return $requetePrepare->fetchAll();
186 }
```

La vue construit le code HTML à partir du tableau retourné :

```

17 <div class="row">
18 <h2>Renseigner ma fiche de frais du mois
19 <?php echo $numMois . "-" . $numAnnee ?>
20 </h2>
21 <h3>Eléments forfaitisés</h3>
22 <div class="col-md-4">
23 <form method="post"
24     action="index.php?uc=gererFrais&action=validerMajFraisForfait"
25     role="form">
26 <fieldset>
27 <?php
28     foreach ($lesFraisForfait as $unFrais) {
29         $idFrais = $unFrais['idfrais'];
30         $libelle = htmlspecialchars($unFrais['libelle']);
```

```

31      $quantite = $unFrais['quantite']; ?>
32      <div class="form-group">
33          <label for="idFrais"><?php echo $libelle ?></label>
34          <input type="text" id="idFrais"
35              name="lesFrais[<?php echo $idFrais ?>]"
36              size="10" maxlength="5"
37              value="<?php echo $quantite ?>"
38              class="form-control">
39      </div>
40      <?php
41      }
42      ?>
43      <button class="btn btn-success" type="submit">Ajouter</button>
44      <button class="btn btn-danger" type="reset">Effacer</button>
45  </fieldset>
46  </form>
47 </div>
48 </div>

```

Fonctionnement de l'application

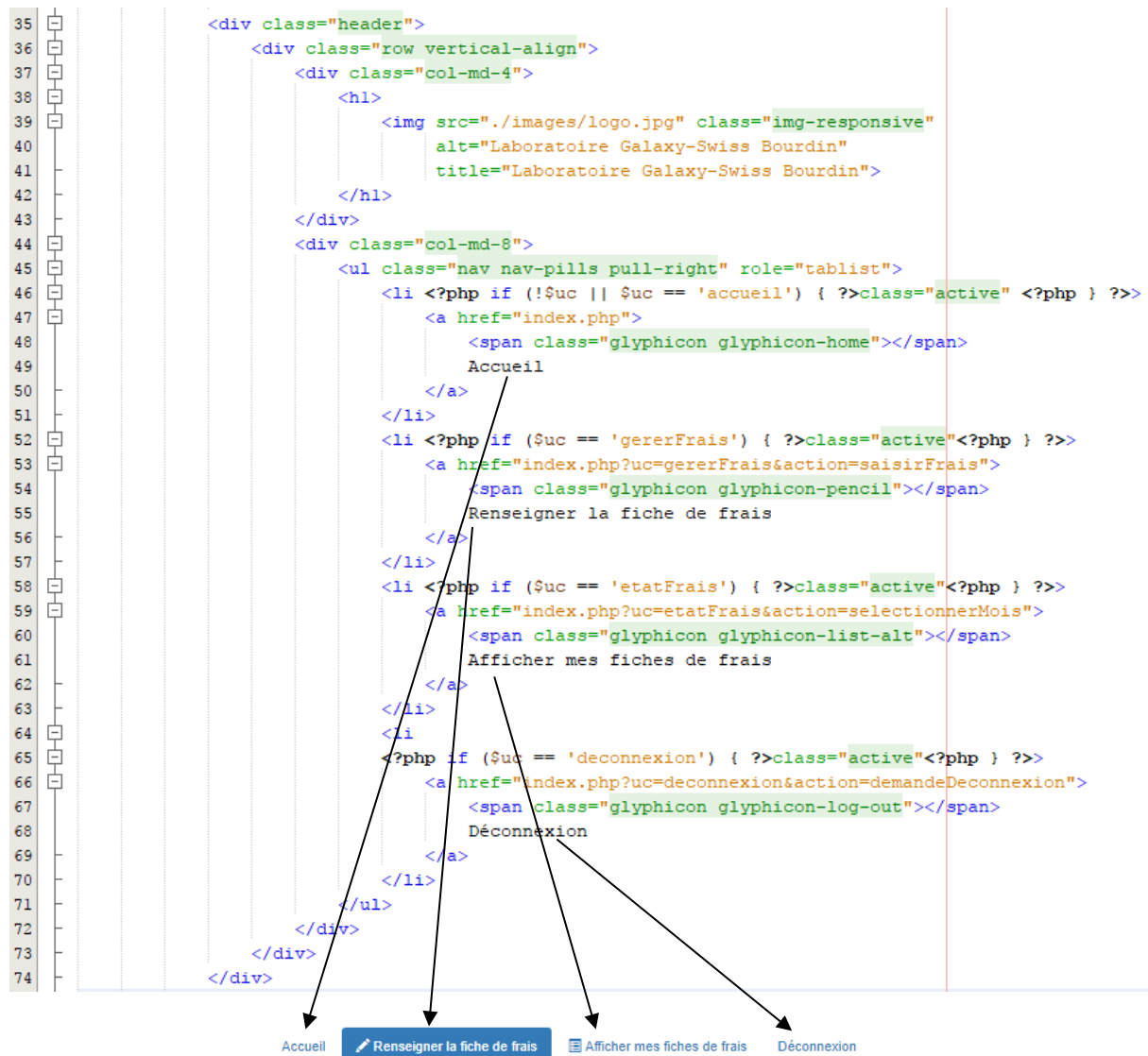
C'est la page index qui sert d'aiguilleur principal et oriente vers un contrôleur de cas d'utilisation :

```

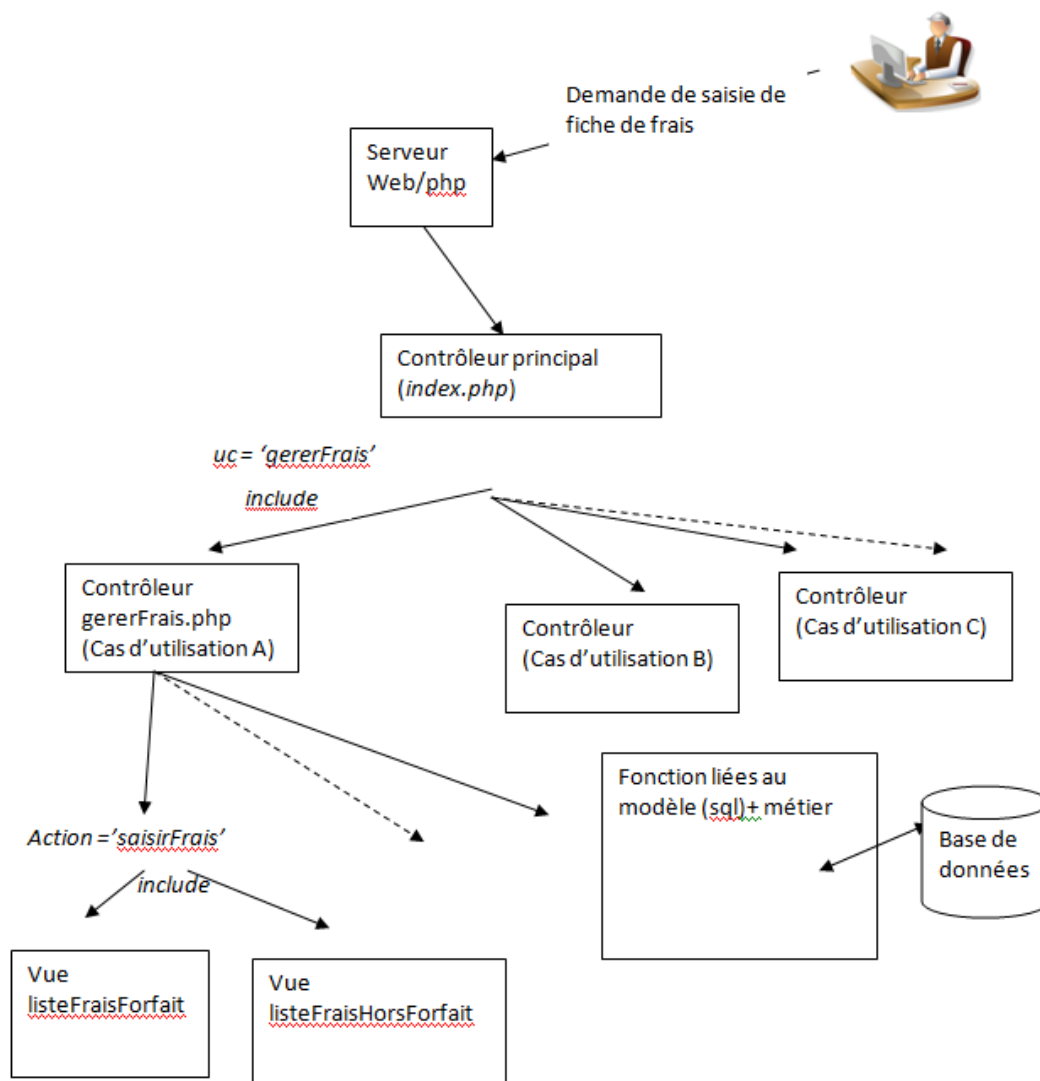
17  require_once 'includes/fct.inc.php';
18  require_once 'includes/class.pdogsb.inc.php';
19  session_start();
20  $pdo = PdoGsb::getPdoGsb();
21  $estConnecte = estConnecte();
22  require 'vues/v_entete.php';
23  $uc = filter_input(INPUT_GET, 'uc', FILTER_SANITIZE_STRING);
24  if ($uc && !$estConnecte) {
25      $uc = 'connexion';
26  } elseif (empty($uc)) {
27      $uc = 'accueil';
28  }
29  switch ($uc) {
30  case 'connexion':
31      include 'controleurs/c_connexion.php';
32      break;
33  case 'accueil':
34      include 'controleurs/c_accueil.php';
35      break;
36  case 'gererFrais':
37      include 'controleurs/c_gererFrais.php';
38      break;
39  case 'etatFrais':
40      include 'controleurs/c_etatFrais.php';
41      break;
42  case 'deconnexion':
43      include 'controleurs/c_deconnexion.php';
44      break;
45  }
46  require 'vues/v_pied.php';

```

Ceci est à associer à ce que l'utilisateur sélectionne dans le sommaire :










On peut résumer cette cinématique par un schéma :



Ainsi chaque page reçue est construite à partir de l'index comme une succession de fichiers *include* selon le cas d'utilisation. L'action demandée entraîne un traitement, à partir de la base de données et des règles métier (responsabilité de la couche Modèle) et expose les vues associées.

L'arborescence du site reflète cette architecture :

 controleurs	23/08/2017 14:38	Dossier de fichiers
 images	23/08/2017 14:38	Dossier de fichiers
 includes	23/08/2017 14:38	Dossier de fichiers
 styles	23/08/2017 14:38	Dossier de fichiers
 tests	23/08/2017 14:38	Dossier de fichiers
 vues	23/08/2017 14:38	Dossier de fichiers
 index.php	23/08/2017 14:34	Fichier PHP

Le répertoire *include* contient les fichiers utiles au *modèle* : accès à la base, fonctions métier, gestion des erreurs.

Utilisation d'une bibliothèque d'accès aux données

Php contient en standard une classe PDO, d'accès aux données ; elle a l'avantage d'être générique-indépendante du SGBD- et propose des services performants qui évitent la plupart du temps de faire soi-même les boucles de parcours des jeux d'enregistrements :

```

188  /**
189      * Retourne tous les id de la table FraisForfait
190      *
191      * @return un tableau associatif
192      */
193  public function getLesIdFrais()
194  {
195      $requetePrepare = PdoGsb::$monPdo->prepare(
196          'SELECT fraisforfait.id as idfrais '
197          . 'FROM fraisforfait ORDER BY fraisforfait.id'
198      );
199      $requetePrepare->execute();
200      return $requetePrepare->fetchAll();
201  }

```

Cette classe n'est pas associée au modèle MVC, mais elle allège grandement la gestion des données.

Dans l'application GSB, la classe PDO est encapsulée dans une classe PdoGsb.