

SurfWave - Dossier technique

Pauline Moncoiffé-Brisset

The screenshot shows the homepage of the SurfWave website. At the top, there is a navigation bar with links to BOUTIQUE, LOCATION, COURS DE SURF, L'HISTOIRE, and CONTACT. There is also a search bar and a connexion button. Below the navigation bar, there are three promotional banners: 1) A banner for the "Festival Manapany 18-19 juillet 2017" with a call to action "Réservez votre place". 2) A banner for custom t-shirts with the text "Pour 25 € REPARTEZ avec votre tee-shirt à VOTRE image !". 3) A banner for Oakley sunglasses with the text "NOUVEL ARRIVAGE à partir de 50 €". Below these banners, there is a section titled "SURF WAVE VOUS RECOMMANDE" featuring three products: 1) Two surfboards labeled "Demiibu". 2) A black backpack labeled "Origami". 3) A black wetsuit labeled "Surfco 6/3mm - Combinaison de surf". The background of the page features a scenic beach and ocean view.

Sommaire

Sommaire	2
Résumé de la situation	2
Kit de démarrage	3
Installation	3
Conseils d'utilisation	4
Vue d'ensemble du projet	4
Organisation des fichiers (arborescence et nomenclature)	6
Organisation de l'affichage	7
Gestion des données saisies en front-end via le Controller	9
Contrôle de saisie	9
Récupération de la donnée du front vers la Database	9
Parcours de la data	10
Organisation des Models	12
En bref	12
Héritage	12
Cas particulier des classes Database et Controller	13

Résumé de la situation

Je n'ai pas réalisé la totalité du projet, mais seulement les parties A, B et C. (Affichage dynamique des tarifs de location, CRUD pour gérer ce tarif, et affichage dynamique de la zone 7. Cette version est "stabilisée".

TODOS:

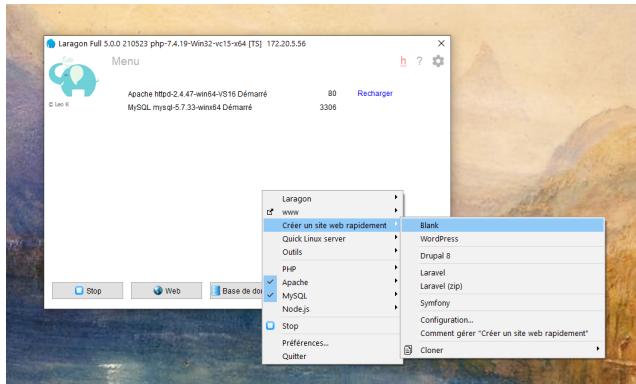
- Gérer plus l'aspect sécurité : éviter d'avoir des messages qui renseignent sur l'architecture du projet lorsque l'on appelle un controller qui n'existe pas
- Sécuriser les form notamment avec un premier contrôle en javascript, puis un second en php
- Faire apparaître normalement les logos/éléments de style dans les vues "modifier", "supprimer", "nouveau"

- Prévoir un affichage pratique pour la présentation et la gestion des tarifs même lorsqu'il manque un tarif
- Renvoyer vers la vraie homepage en cas de déconnection de l'utilisateur
- Optimiser les méthodes existantes, en abusant par exemple de concat() côté SQL
- Réaliser les étapes D et E

Kit de démarrage

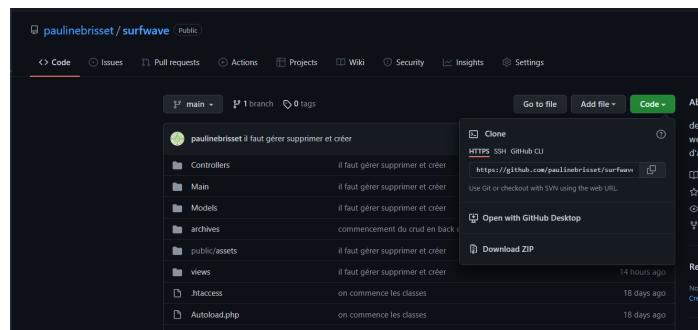
Installation

Pour installer le site:



1. Ouvrir Laragon
2. Faire un clic droit sur la fenêtre
3. Sélectionner "créer un site web rapidement", puis "blank"
4. Récupérer le code du projet sur GitHub via le lien suivant :
<https://github.com/paulinebrisset/surfwave.git>
 choisir "dowload zip".

5. Extraire tous les fichiers téléchargés dans le dossier "surfwave"
6. Pour la base de données : le script nécessaire à la construction de la database est dans le dossier "archives" du projet, et nommé sql.sql. Il est à faire exécuter tel quel par phpmyadmin.
7. Les identifiants de connexion à la base de données sont définis comme suit : id : root / mot de passe : vide. Ces codes peuvent être modifiés dans Main/Database
8. ouvrir "surfwave.test" dans le navigateur
9. Les identifiants de connexion à l'interface du gestionnaire sont définis comme suit : id: Admin@admin.fr ; mdp : Admin.

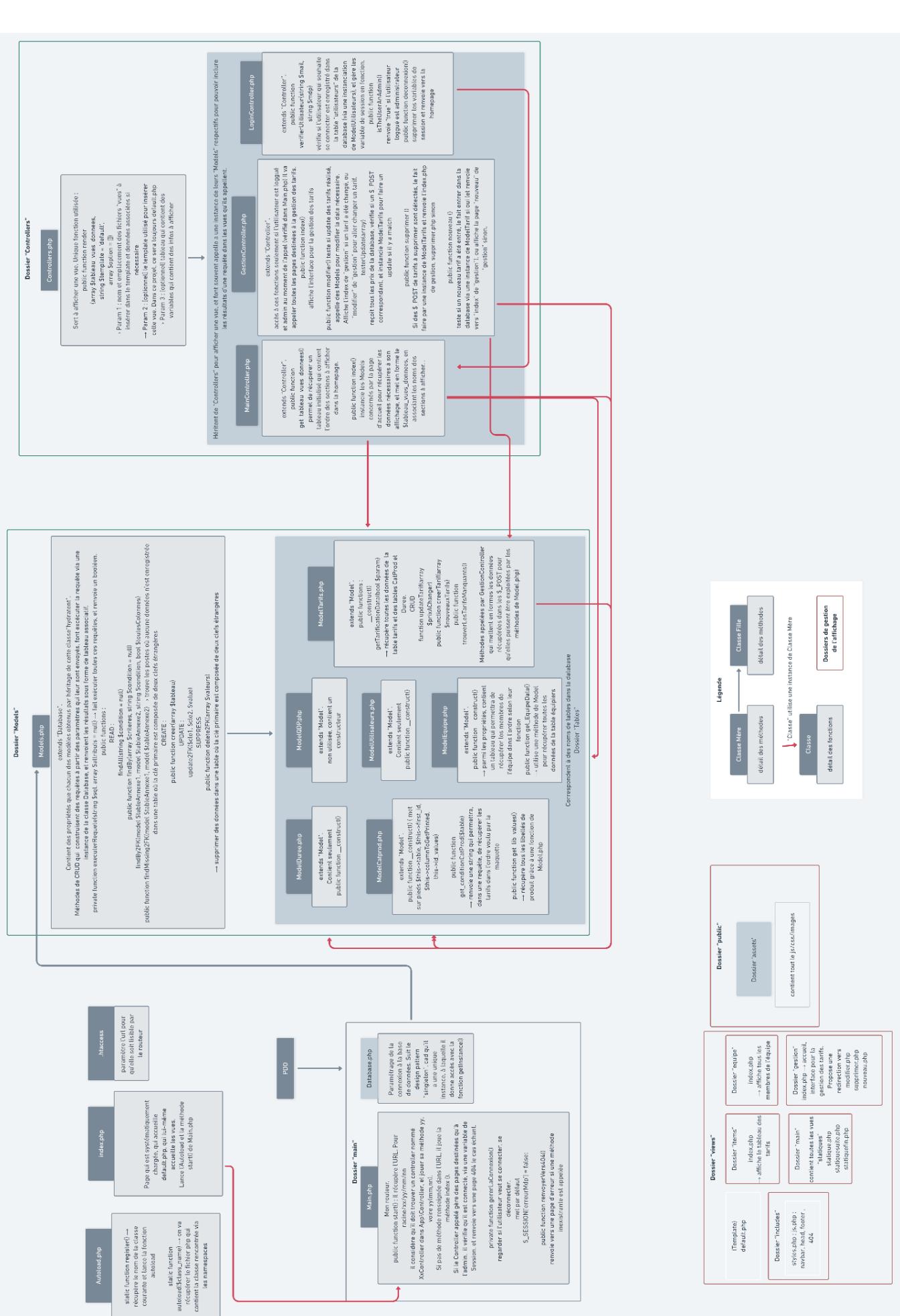


Conseils d'utilisation

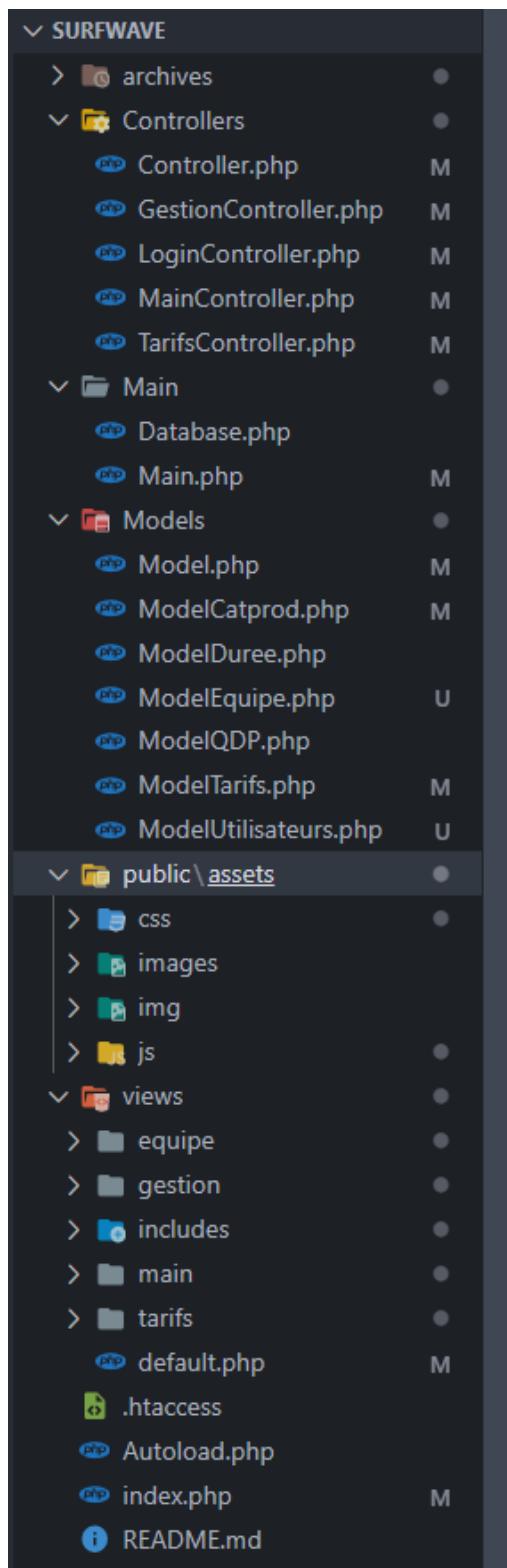
- > Les tarifs devront impérativement être renseignés pour tous les produits sur toutes les plages horaires présentées pour garantir un affichage optimal
- > Les photos des membres de l'équipe devront toutes être mises au format surnom.jpg

Vue d'ensemble du projet

page suivante

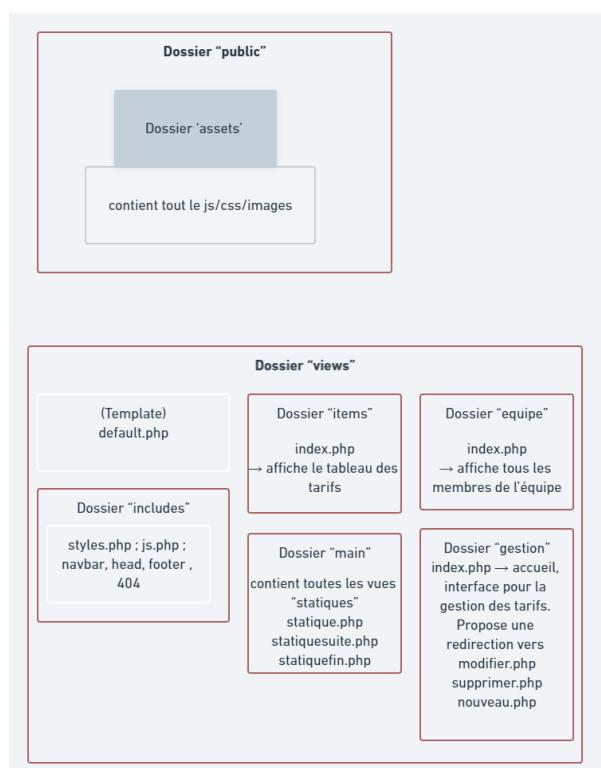


Organisation des fichiers (arborescence et nomenclature)



- un dossier "Archives" contient le script SQL, le dossier technique, etc.
- Models contient Model.php et toutes ses instances (6, dont 5 utilisées dans cette version). Ces instances correspondent à des tables de la database.
- Controllers contient Controller.php et toutes ses instances (4)
- Dans Main on gère l'accès à la base de données (Database.php), et Main.php sert de routeur
- Des namespaces sont utilisés pour toutes les classes du projets. L'organisation de leur arborescence correspond à celle de leur emplacement physique, avec App\ pour l'Autoload, à la racine, puis App\Main ; App\Controllers et App\Models.
- Il utilise pour cela htaccess, à la racine
- public contient, dans "assets", le css, javascript, les fichiers bootstrap, les images

- à la racine, index.php et la page qui est toujours lancée, et Autoload permet de charger les classes nécessaire lorsqu'elles en ont besoin
- dans le dossier "views", on trouve plusieurs dossiers qui comprennent des fichiers généralement nommés "index.php" ou répondant au nom d'une fonction du controller qui se charge de les faire apparaître. Le dossier "main" de "views" contient les vues qui sont restées "statiques" par rapport à la maquette fournie. Elles sont insérées dans la page de la même façon que les vues dynamiques.
- Toujours dans "views", le seul fichier qui sort du lot est default.php. Il me sert de template. Il accueille par avance toutes les parties "générales" du site, rangées dans includes (la barre de navigation, le footer...) et a un espace réservé pour accueillir \$content, soit une variable qui contiendra l'ensemble du contenu à afficher, qui est paramétrée dans la fonction render() de Controller.php



De l'url à l'affichage

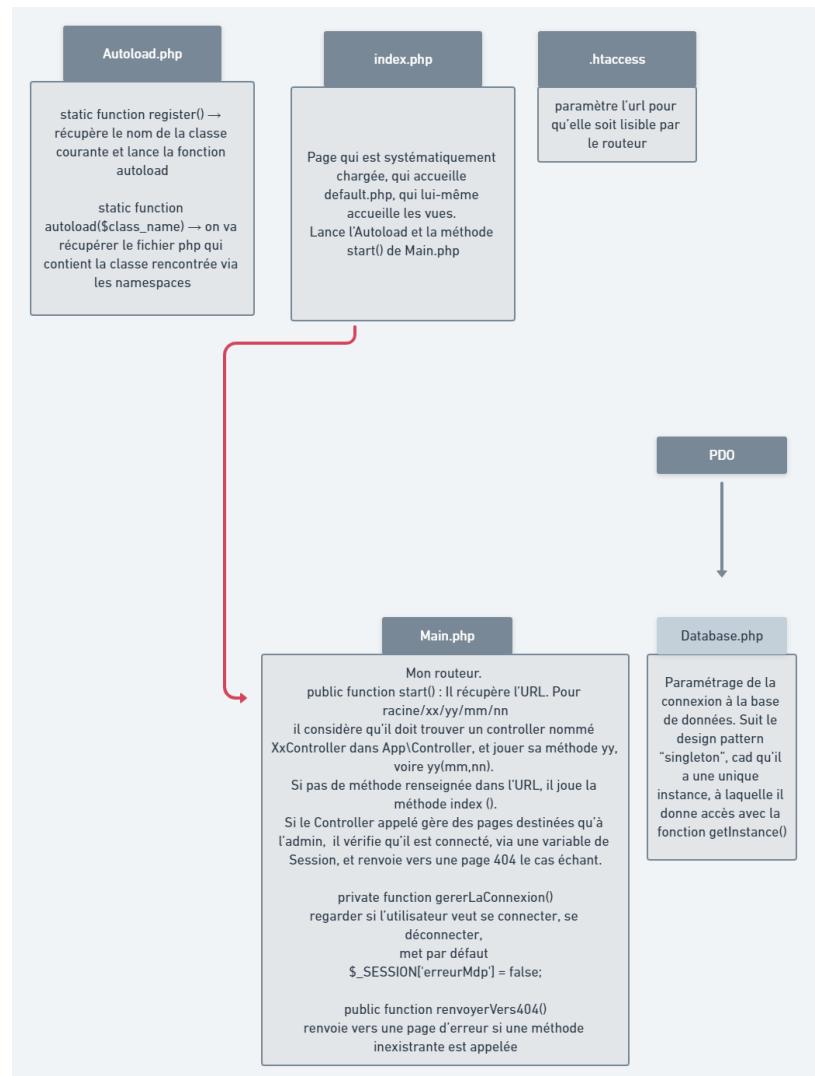
La page index.php, à la racine, est celle qui est toujours appelée. Elle charge l'Autoloader, instancie Main.php, et lance la fonction start() de Main. Main est un Routeur rustique, qui gère très peu l'erreur.

Le fichier .htaccess permet d'instaurer une réécriture de l'url pour qu'elle puisse être lue par start().

Exemple : Pour surfwave.test/gestion/modifier, start() définira qu'il faut exécuter la méthode modifier() du fichier GestionController. Pour utiliser ce controller en particulier, il est nécessaire que l'utilisateur soit identifié et possède les droit d'administration du site. C'est start() qui se charge de cette vérification avant que le Controller ne soit utilisé.

Les méthodes appelées via l'url finissent toutes, via la méthode render() de Controller.php, par "require" un fichier ou un ensemble de fichier contenus dans le dossier views, en leur envoyant si besoin de la donnée.

Il n'y a pas de fichier "routes.php", les vues à afficher sont définies au cas par cas. Toutes les vues sont "imprimées" via la variable \$content, dont un echo est fait dans le seul template de ce projet, 'default.php'.



Contenu du fichier default.php

```

<!DOCTYPE html>
<html lang="fr">
<?php include($_SERVER['DOCUMENT_ROOT'] . '/views/includes/head.php') ?>
<?php include($_SERVER['DOCUMENT_ROOT'] . '/views/includes/styles.php') ?>

<body>
    <!-- HEADER -->
    <header>
        <?php include $_SERVER['DOCUMENT_ROOT'] . '/views/includes/navbar.php' ?>
    </header>

    <?= $content ?>

    <?php include($_SERVER['DOCUMENT_ROOT'] . '/views/includes/footer.php') ?>

</body>

</html>

```

Gestion des données saisies en front-end via le Controller

Contrôle de saisie

Dans cette première version, le projet comporte très peu de contrôle de saisie. Celle-ci se fait pour l'instant par des inputs très précisément typés dans le code HTML des vues. Une liste de choix prédéfinie vient généralement remplacer un input de saisie, quand c'est possible.

Exemple avec la suppression ou création de tarifs

SUPPRESSION

Sélectionnez l'horaire et le produit pour lesquels vous voulez supprimer un tarif

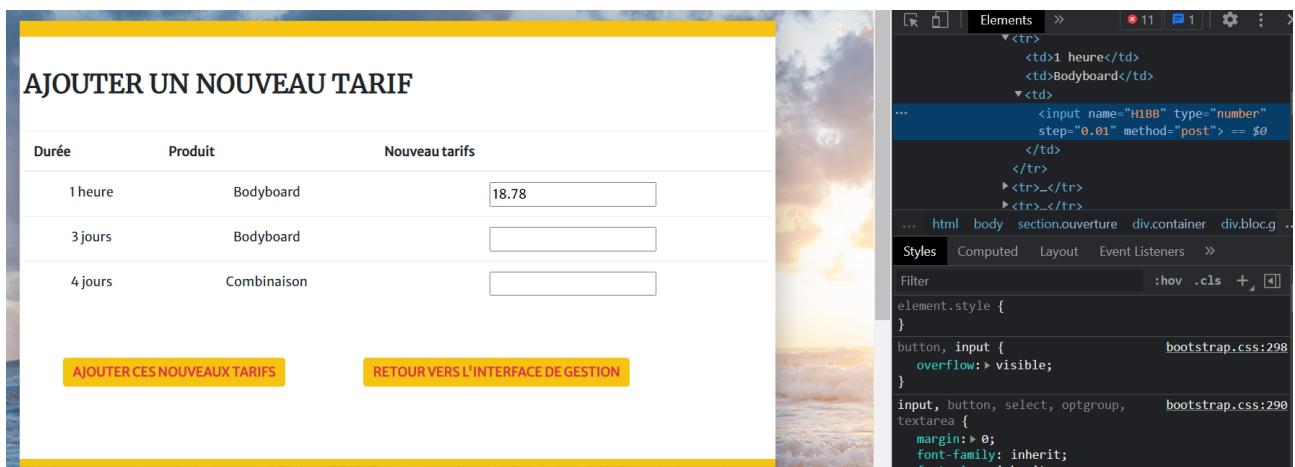
Créneau horaire concerné	<input type="text" value="1 heure"/>	<input type="button" value="▼"/>
Produit concerné	<input type="text" value="Bodyboard"/> Bodyboard Combinaison Planche de surf	<input type="button" value="▼"/>
<input type="button" value="SUPPRIMER"/>		

L'admin a une liste de choix imposés à la fois pour supprimer un tarif ou créer un nouveau tarif. Cette liste est basée sur les enregistrements des table CatProd et Duree.

Un nouveau tarif est forcément un nombre n'ayant pas plus de deux chiffres après la virgule.

L'utilisateur ne peut créer un nouveau tarif que pour les enregistrements identifiés comme étant "manquants" dans la table tarif, grâce à une méthode de Model.

Il peut en revanche supprimer un tarif qui n'existe déjà pas, si cela lui fait plaisir.



Récupération de la donnée du front vers la Database

Lorsque des formulaires sont disponibles dans les vues, les données sont transmises par la méthode post. Dans son “action”, les formulaires prévoient qu’au clic sur submit, la page se recharge elle-même.

Les données transmises par formulaire sont interceptées par la même méthode qui gère l'affichage, et redirige vers l'interface de gestion plutôt que vers la vue initialement prévue si un \$_POST contient une valeur à gérer.

Exemple avec la méthode supprimer() de GestionController

```
public function supprimer()
{
    //vérifier si il y a déjà eu un essai de suppression de prix
    if (isset($_POST['duree']) && isset($_POST['categoprod'])) {
        $instanceModelTarif = new ModelTarifs;
        $instanceModelTarif->delete2FK($_POST['categoprod'],
$_POST['duree']]);
        header('Location: /gestion');
        exit;
    }
    //Lancement de la page si il n'y a pas eu de suppression

    $instanceModelCatProd = new ModelCatprod;
    $categoprod = $instanceModelCatProd->findAll();

    $instanceDuree = new ModelDuree;
    $duree = $instanceDuree->findAll();

    $tableau_vues_donnees[] = ['gestion/supprimer', ['categoprod' =>
$categoprod]];
    $tableau_vues_donnees[] = ['gestion/supprimer', ['duree' => $duree]];

    $this->render($tableau_vues_donnees);
}
```

Parcours de la data

Si une donnée doit être sélectionnée, créée, mise à jour ou supprimée en database, le parcours est le suivant :

-
1. La donnée est reçue par le controller via un formulaire sur l'interface
 2. Le controller instancie l'instance de Model appropriée pour traiter cette donnée
 3. L'instance va utiliser une méthode de Model.php pour envoyer sa requête en database. Soit l'instance doit mettre en forme la donnée avant, soit elle utilise la méthode convenue directement.
 4. Le Model assemble la requête, la fait exécuter, et renvoie généralement un tableau associatif.

Organisation des Models

En bref

Model est la seule classe qui communique directement avec la base de données, via la classe Database qui lui fournit une instance unique de connexion. Models n'utilise que la méthode executeRequete pour faire réaliser les requêtes par la base de données.

Ses autres méthodes concernent toutes les préparations de requête.

Héritage

Model.php contient des attributs "protected" vides qui sont écrites dans les constructeurs des instances de Model.

```
class Model extends Database
{
    protected $table;
    protected $first_id;
    protected $second_id;
    protected $columnToGetPrinted;
    protected $id_values;
    // Instance de connexion
    private $db;
```

```
class ModelTarifs extends Model
{

    public function __construct()
    {
        $this->table = 'tarifs';
        $this->first_id = 'categoProd';
```

```

    $this->second_id = 'codeDuree';
    $this->columnToGetPrinted = 'prixLocation';
}

```

Cela permet de pré-écrire, dans Model, des requêtes très générales.

Par exemple, cette méthode qui vise à récupérer des données sur trois tables, alors que les clés primaires des deux premières composent la clé primaire de la troisième (celle qui appellera la méthode).

J'envoie donc simplement des instances des classes héritées de Models qui sont concernées, et la méthode peut composer elle-même la requête à partir de ses classes filles.

extrait :

```

public function findBy2FK(model $tableAnnexe1, model $tableAnnexe2, string
$condition = null, bool $toutesColonnes)
{
    $table1 = $tableAnnexe1->get_tableName();
    $cle1 = $tableAnnexe1->get_first_id();
    $colonneAPublier1 = $tableAnnexe1->get_columnToGetPrinted();

    $table2 = $tableAnnexe2->get_tableName();
    $cle2 = $tableAnnexe2->get_first_id();
    $colonneAPublier2 = $tableAnnexe2->get_columnToGetPrinted();

    // $this se réfère au model appelant
    $tablePrincipale = $this->get_tableName();
    $colonnePrincipale = $this->get_columnToGetPrinted();

[.....autres traitements d'assemblage.....]

    // J'assemble le tout
    $requete = ($listeSelect . $from . $premiereJointure .
    $secondeJointure . $condition);
    $request = $this->executerRequete($requete);
    return $request->fetchAll();
}

```

Tous les noms de tables et champs à aller chercher sont accessibles avec les mêmes méthodes.

Le rôle des classes héritées de Model est donc de formater les données à envoyer en base de données de manière à ce qu'elles soient correctement manipulées par les méthodes de Models.php.

Cas particuliers des classes Database et Controller

La chaîne de connexion pour l'accès à la Database se crée à partir de constantes de la Classe, où sont enregistrés les codes d'accès, etc. Database.php a une méthode statique qui permet de générer une seule instance de cette classe. Cette méthode est utilisée dans une seule fonction de Model.

Controller est une classe abstraite, il n'est donc jamais instancié.