# DOSSIER PROJET

# ANNEXE : ECFs  Backend


# CORALIE DUBREUIL

# Sommaire

# Ecf 3 : Parc locatif

## 1. Liste des compétences du référentiel couvertes par le projet

**Activité type : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité**

**Compétences professionnelles :**

5, Créer une base de données

6. Développer les composants d'accès aux données

7. Développer la partie back-end d'une application web ou web mobile

## 2. Cahier des charges du projet

## Contexte du projet

Il a fallu réalisé le back-end d'une application web, sous une approche «web services» destiné à être appelé ultérieurement par un front-end pour le compte d'un organisme de gestion d'un parc locatif de locaux professionnels.

## Processus métier

L'application web est destinée à gérer les entités métier du processus de gestion de la location de locaux professionnels. Les règles métier de l'organisme gestionnaire sont les suivantes :

- L'organisme est propriétaire de biens immobiliers qui sont caractérisés par :
    - une surface (en mètres carrés)
    - un nombre de pièces
    - un prix de loyer mensuel en euros
    - une adresse (dans un souci de simplification, on ne retiendra que le nom de la ville pour la localisation du bien).
- Pour identifier ses biens, l'organisme attribue à chacun d'entre eux un code de gestion unique sous la forme «LOC####» où #### correspond à une série de 4 chiffres maximum.
- Chaque bien immobilier peut être loué simultanément à une seule et unique entreprise cliente.
- Chaque entreprise cliente peut louer un ou plusieurs biens simultanément.
- Chaque entreprise cliente est caractérisée par :
    - son nom
    - son numéro unique d'immatriculation officiel appelé SIREN (dans un de souci de simplification, on considérera que le SIREN est un simple code à 9 caractères).
- Pour identifier ses clients, l'organisme attribue à chacun d'entre eux un code client unique sous la forme «CLI####» où #### correspond à une série de 4 chiffres maximum.

L'organisme ne conserve pas d'historique sur les locations et souhaite seulement avoir un état des lieux à un instant donné du parc locatif : bien loué ou bien vacant.

## Spécifications fonctionnelles

Le back-end de l'application se présente sous la forme d'une API «web services» permettant de manipuler les entités du processus métier selon le modèle CRUD. L'API a donc pour but de lister, créer, modifier, supprimer et rechercher les entités métier. Techniquement, l'API est un jeu d'URL HTTP en méthode GET renvoyant des données au

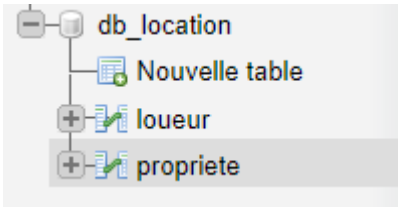
format JSON. Le format des URL à développer n'est pas standardisé mais correspond à une implémentation «propriétaire» interne à l'organisme.

## Jeu de test

Afin de tester l'application, il est demandé de saisir un jeu de données de 3 entités minimum pour chaque entité métier.

## 3. Présentation du rendu visuel du projet


*Mld pour la base de données*




*Arborescence de la base de donnée*

+ Options

| | code_client | siren | nom |
|---|---|---|---|
| Éditer Copier Supprimer | CLI0001 | a1b2c3d4e | Client1 |
| Éditer Copier Supprimer | CLI0010 | f1g2h3i4j | Client10 |
| Éditer Copier Supprimer | CLI0100 | k1l2m3n4o | Client100 |

*Table loueur*

+ Options

| | code_location | surface | nb_piece | loyer_mensuel | ville | code_client_loueur |
|---|---|---|---|---|---|---|
| Éditer Copier Supprimer | LOC0001 | 20 | 1 | 350 | Ville1 | CLI0001 |
| Éditer Copier Supprimer | LOC0010 | 60 | 3 | 768 | Ville2 | CLI0100 |
| Éditer Copier Supprimer | LOC0100 | 55 | 3 | 724 | Ville2 | *NULL* |
| Éditer Copier Supprimer | LOC1000 | 120 | 6 | 999 | Ville1 | CLI0001 |

*Table propriété*


*Liste des clients affichée dans le navigateur*

Il n'y a pas de rendu visuel à proprement parler pour ce projet, le seul résultat attendu affiché par le navigateur est la capture d'écran juste au-dessus avec la liste en JSON des codes clients provenant de la base de donnée créée pour le projet.

J'ai également ajouté les images du mld réalisé avant la création de la base de donnée qui permet de se représenter les tables de la base de donnée, leurs informations ainsi que les liens entre les tables par l'intermédiaire des clés étrangères.

J'ai également ajouté des captures d'écrans de la base de donnée après sa création dans l'interface phpMyAdmin, où on peut voir les deux tables ainsi que les jeux de données utilisés pour l'exercice et on pout donc constater que les données code_client sont bien les mêmes entre celles de la base de données et celles qui s'affiche dans le navigateur.

## 4. Présentation technique du projet

Pour créer la base de données, j'ai écris un script qui commence par supprimer une possible base de données du même nom préexistante (dû à des tests précédents par exemple). Ensuite je crée la base de donnée puis les tables avec leurs différentes entrées de données. En d'autres termes je crées les tableaux avec les titres de colonnes mais pas les contenus. Ceux-ci seront ajoutés directement dans PHPmyAdmin, l'interface pour visualiser la base de données. Les tables sont créees dans un ordre précis, celles sans

« foreign keys » (ou clés étrangères) avant celle qui en ont. Les clés étrangères servent à relier les tables entre elles.

```sql
-- Suppression de la DB
▶ Execute
DROP DATABASE IF EXISTS db_location;

-- Création de la DB
▶ Execute
CREATE DATABASE db_location;

▶ Execute
USE db_location;
-- Création de la table Loueur
▶ Execute
CREATE TABLE loueur(
    code_client VARCHAR(7) NOT NULL,
    siren VARCHAR(9) NOT NULL UNIQUE,
    nom VARCHAR(56) NOT NULL,

    PRIMARY KEY(code_client)
);

-- Création de la table location
▶ Execute
CREATE TABLE propriete(
    code_location VARCHAR(7) NOT NULL,
    surface DECIMAL NOT NULL,
    nb_piece INT NOT NULL,
    loyer_mensuel DECIMAL NOT NULL,
    ville VARCHAR(56) NOT NULL,
    code_client_loueur VARCHAR(7),


    PRIMARY KEY(code_location),
    FOREIGN KEY (code_client_loueur) REFERENCES loueur(code_client)
);
```

*Script pour la création de la base de données*

Je crée ensuite un MVC pour « Model View Controller », c'est une structure qui permet de séparer clairement les principaux composants de l'application ou du site. Il est constitué des éléments suivant (visible dans les captures d'écran suivantes) :

- le(s) modèle(s) qui se chargent des échanges avec la base de données grâce à la méthode CRUD (pour « Create, Read, Update, Delete ») auquel on peut éventuellement ajouter un S pour « Search » (SCRUD). Ce sont les 4 ou 5 opérations de bases qu'on peut

réaliser avec une base de données. En français, cela donne créer, lire, mettre à jour, supprimer et éventuellement rechercher. Un modèle ne sert qu'à faire des requêtes à la base de données, il n'y a pas de traitement de ces données dedans.

- La(es) vue(s) contient le code HTML destiné à structurer les pages. En général, elles sont séparées entre les grandes parties du codes telles que « header », « footer »…

- Le(s) contrôleur(s) contient le traitement des données demandées par le modèle en vue de leur affichage dans la vue par exemple.

- Le routeur est toujour unique dans un MVC, c'est le point d'entrée de l'application. Il envoie la demande au bon contrôleur pour afficher la bonne page, issue du bon chemin, demandée par l'utilisateur.

```php
//Récupération de PDOUtils
require_once "./utils/PDOUtils.php";

//Création de la classe Client
class ModelClient
{
    // Identifiant unique client
    private $code_client;

    // Siren du client
    private $siren;

    // Nom du client
    private $nom;

    // Fonctions de récupération des données
    //Récupération du code client
    public function getCode()
    {
        return $this->code_client;
    }

    //Récupération du numéro Siren du client
    public function getSiren()
    {
        return $this->siren;
    }

    //Récupération du nom du client
    public function getNom()
    {
        return $this->nom;
    }
```

*Création de la classe Client*

```php
<?php

// Récupération du modèle
require_once "./model/ModelClient.php";

class ControllerClient
{

    private ModelClient $model;

    public function __construct()
    {
        // Instantiation du modèle
        $this->model = new ModelClient();
    }

    // Lister les clients
    public function list()
    {
        $clients = $this->model->getAllClients();

        require_once "./view/list.php";
    }

    // Ajouter un client
    public function add()
    {
        //a faire
        echo "action ajouter";
    }

    // Suppimer un client
```

*Controller*

```php
// Fonctions d'attribution des données
//Attribution du code client
public function setCode($code_client)
{
    $this->code_client = $code_client;
}

//Attribution du numéro de Siren du client
public function setSiren($siren)
{
    $this->siren = $siren;
}

//Attribution du nom du client
public function setNom($nom)
{
    $this->nom = $nom;
}

// Méthodes du CRUD

public function getAllClients()
{
    $prepared_sql = "SELECT code_client, siren, nom FROM loueur";
    $array_values = [];

    return PDOUtils::query($prepared_sql, $array_values, true);
}

public function add()
{
   //a faire
}
public function update()
```

*Créations des fonctions*

```php
<?php

//Récupération du Contrôleur
require_once "./control/ControllerClient.php";

//Récupération du header :
  require_once "./view/header.php";

//Création du contrôleur
$control = new ControllerClient();

// Vérification de l'action demandée, si elle est demandée
if (isset($_GET["action"])) {

    switch ($_GET["action"]) {

        //Action ajouter
        case "add":
            $control->add();
            break;

        //Action Editer
        case "edit":
            $control->edit();
            break;

        //Action supprimer
        case "remove":
            $control->remove();
            break;

        //Action Lister
        case "list":
            $control->list();
            break;
```

*Router*

```php
<h2>Liste des Clients</h2>
<?php

//Création d'un tableau avec les valeurs des codes clients
$array_code_client = array();
foreach ($clients as $client){
    $array_code_client[] = $client["code_client"];
};

//Création du tableau
$array_client = [
    'action' => "list",
    'entity' => "client",
    'data' => $array_code_client
];

//Conversion du tableau en json
$json_client = json_encode($array_client);

echo $json_client;
?>
```

*Création du résultat final dans la view*

# Ecf 4 : Jeu de la chance

## 1. Liste des compétences du référentiel couvertes par le projet

**Activité type : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité**

**Compétences professionnelles :**

8. Élaborer et mettre en œuvre des composants dans une application de gestion de contenu.

## 2. Cahier des charges du projet

**Contexte du projet**

Il a fallu réaliser un plugin Wordpress destiné à proposer un mini-jeu de hasard « numéro chance » lors de la consultation de la home page d'un site web réalisé avec Wordpress.

**Déroulement du jeu**

Le numéro chance est un nombre entier compris entre 1 et 9.

À l'installation du plugin, un numéro chance aléatoire est défini. Par la suite, l'administrateur du site va pouvoir modifier le numéro.

Lors de la consultation de la page d'accueil du site web, le jeu est proposé systématiquement à l'utilisateur. Celui-ci va pouvoir alors décider de tenter sa chance en jouant ou de ne pas jouer et d'annuler le jeu.

S’il décide de jouer, il va pouvoir choisir un nombre entier entre 1 et 9. Si son choix correspond au numéro chance, un message lui indiquera « C’est votre jour de chance ! », sinon le message lui indiquera « Dommage, la chance n’était pas loin... ». Après avoir pris connaissance du message, l’utilisateur peut continuer normalement la consultation du site web.

**Spécifications fonctionnelles**

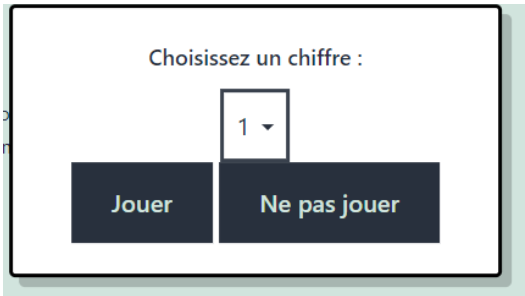
La zone de jeu devant s’afficher sur la page d’accueil sera constituée d’une fenêtre popu intitulée « Jour de chance ? » incluant une liste déroulante permettant de choisir un nombre entier entre 1 et 9. Deux boutons « Jouer » et « Ne pas jouer » permettront respectivement de vérifier le numéro chance et d’annuler le jeu.

Un clic sur le bouton « Jouer » Vérifiera si le numéro choisi correspond au numéro chance . Le contenu de la fenêtre popup sera alors remplacé par le message adéquat défini dans le cahier des charges, ainsi qu’un unique bouton « OK » permettant de fermer la fenêtre popup.

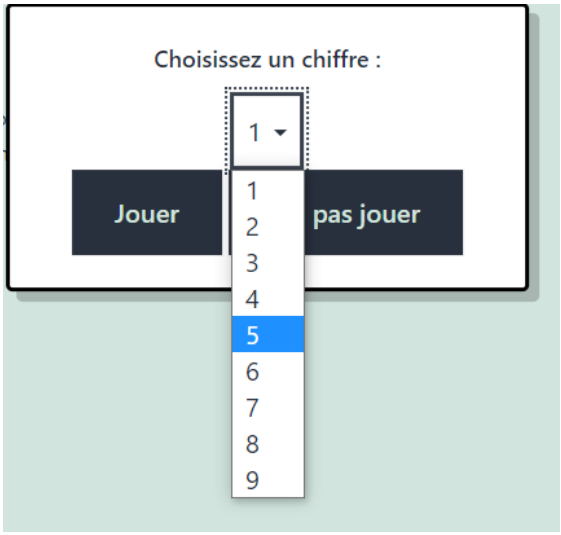
Un clic sur le bouton « Ne pas jouer » fermera simplement la fenêtre popup.

Afin de définir le numéro chance, une zone de saisie spécifique sera intégrée à la partie admin (« dashboard ») du site web.

## 3. Présentation du rendu visuel du projet

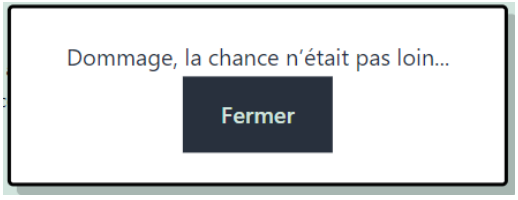



*Présentation du jeu*




*Choix du chiffre*


*Gagné*




*Perdu*

Le but de cet exercice était de créer une extension Wordpress en PHP qui permet à l'administrateur de sélectionner un chiffre entre 1 et 9 qui sera le nombre chanceux du jour (voir l'interface côté administrateur ci-dessous). Du côté de l'utilisateur si il le souhaite, il peut choisir un chiffre entre 1 et 9 dans le menu déroulant pour voir si il correspond au numéro sélectionner par l'administrateur. Un message s'affiche ensuite en fonction du résultat. Il peut également choisir de ne pas jouer, à ce moment là la fenêtre pop-up s'efface et l'utilisateur à accès au site derrière.


*Extension dans la partie administration*

## 4. Présentation technique du projet

DWWM > chance > JS chance.js > ...

```javascript
// Fonction pour fermer la fenêtre modale

function fermer() {
  // Suppression de la fenêtre modale
  document.getElementById("divchance").remove();
  document.querySelector("style[title='chance_css']").remove();
}

// Fonction pour le jeu

function jouer() {
  let essai = document.getElementById("numero").value;
  let numeroChance = document.getElementById("vrai_numero").value;
  let selection = document.getElementById("divchance");

  if (essai == numeroChance) {
    // Le jeu est gagné
    selection.innerHTML =
      "<p>C'est votre jour de chance !</p><button class='btn' onclick='fermer()'>Fermer</button>";
  } else {
    // Le jeu est perdu
    selection.innerHTML =
      "<p>Dommage, la chance n'était pas loin...</p><button class='btn' onclick='fermer()'>Fermer</button>";
  }
}

```

*Code Javascript pour le jeu*

En JavaScript, je crée 2 fonctions, la première pour la suppression de la fenêtre du jeu si elle est demandée par l'utilisateur. La deuxième organise le jeu en récupérant la valeur du chiffre choisi par l'utilisateur, celle choisie par l'administrateur et en les comparant. En fonction du résultat la fonction sélectionne le message a affiché.

```php
// Check admin mode
if (is_admin()) {

  //Enregistrement du callback sur le hook 'ajout du menu d'admin'

  add_action('admin_menu', 'chance_add_menu');

  // Fonction de création de menu du plugin

  function chance_add_menu()
  {
    // Ajout d'une page d'admin dans le dashboard

    add_menu_page(
      'Configuration du jeu de la chance', // Nom de la page
      'Jeu de la chance', // Nom du menu
      'edit_plugins', // WP capabilites
      'chance_admin',  // Menu URL slug
      'chance_show_panel', // Callback de gestion de la page
    );
  }
```

*Code PHP pour l'ajout de l'extension dans le menu administration*

```php
/**
 * Plugin Name: numero_chance
 * Description: Choisir un numéro entre 1 et 9 pour tester sa chance.
 * Author: Coralie Dubreuil
 * Version: 0.1
 */


// Enregistrement des fonctions sur les hooks

register_activation_hook(__FILE__, 'chance_enable');

register_deactivation_hook(__FILE__, 'chance_disable');

register_uninstall_hook(__FILE__, 'chance_uninstall');

// Fonction d'activation du plugin

function chance_enable()
{
  add_option('numero_chance', 8);
}

// Fonction de désactivation du plugin

function chance_disable(){}

// Fonction de désintallation du plugin

function chance_uninstall()
{
  //  Suppression inconditionnelle de la 'WP option' de la base de données

  delete_option('numero_chance');
}
```

*Code PHP de placement sur les hook*

Pour permettre au plugin de s'afficher dans WordPress, il faut le relier au thème utiliser grâce aux « Hooks » ou crochet. Ce sont des fonctions déclarées par le développeur du thème qui permettent d'interagir avec WordPress. Il existe deux types de Hook :

- Les actions qui permettent de lancer ma propre fonction à un moment clées pour ajouter des choses supplémentaires dans WordPress.

_ Les filtres qui permettent d'intercepter une valeur afin de la modifier.

```php
// Fonction d'affichage de la page d'admin du plugin

function chance_show_panel()
{
  // Flag de conexte de mise à jour
  $updated = false;

  // Controller de la page d'admin

  if ($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_POST['submit'])) {

    // Vérification du nonce

    if (isset($_POST['jeton']) && wp_verify_nonce($_POST['jeton'], 'chance_admin_nonce')) {

      // Vérification du la présence de l'attribut de la requête
      if (isset($_POST['numero']) && !empty($_POST['numero'])) {

        // Récupération du message d'avertissement
        $numero = $_POST['numero'];

        // Mise à jour du message d'avertissement dans la table de données

        $updated = update_option('numero_chance', $numero);
      }
    }
  } else {

    // Récupération du message d'avertissement depuis la table de données

    $warning = get_option('numero_chance');
  }
```

*Code PHP pour le contenu de l'extension*

Dans le cas de ma création de plug-in, j'utilise des hooks d'action.