# Module M2105 - RT web dyna - TD/P 2

En cas de problèmes avec le démarrage d'apache :

- Lancer le bash **setup-xampp.bat** du dossier de xampp
- Démarrer **xampp-control**
- Démarrer le serveur **Apache**
- Tester la réponse du serveur en allant à l'adresse **http://127.0.0.1** dans un navigateur client

## Notions abordées

- Programmation orientée Objet
- Prise en main d'un framework
  - Contrôleurs
  - Formulaires/vues Twig
- Tableaux associatifs
- Utilisation de la Session Http (**USession**)

## Installations

- php 7.1 ou supérieur (vérifier avec php -v en invite de commande)
- composer (Téléchargement)
- PhpStorm ou Eclipse PHP

Installer **Ubiquity-devtools**

```
composer global require phpmv/ubiquity-devtools 1.0.x-dev
```

En cas de problèmes, consulter Procédures pour install de Xampp, composer, Ubiquity

### Création du projet

Créer le projet **tp2** en invite du commande à partir du dossier **htdocs** de XAMPP

```
cd htdocs
ubiquity new tp2 -q=semantic -a
```

Ouvrir/créer ce projet avec votre IDE (Eclipse ou PHPStorm)

Vérifier que les fichiers **.htaccess** et **app/config.php** font bien référence au dossier adéquat à partir du dossier **htdocs** de xampp :

```
AddDefaultCharset UTF-8
```

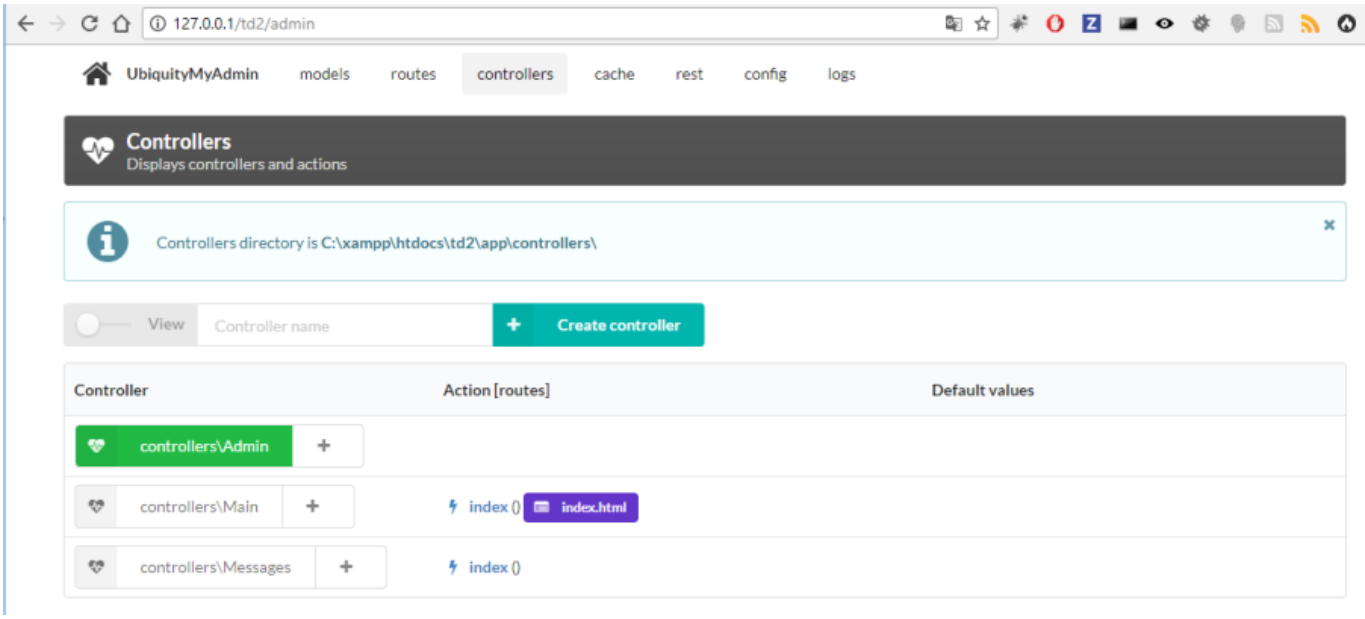
```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /tp2/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{HTTP_ACCEPT} !(.*images.*)
    RewriteRule ^(.*)$ index.php?c=$1 [L,QSA]
</IfModule>
```

```
<?php
return array(
        "siteUrl"=>"http://127.0.0.1/tp2/",
...
```

# Intro

## Contrôleur et action par défaut

- Créer le contrôleur **Messages**, à partir de la console d'administration **http://127.0.0.1/tp2/admin**.




- Afficher le texte **Hello world** dans la méthode **index()** :

```
<?php
namespace controllers;
 /**
 * Controller Messages
 **/
class Messages extends ControllerBase{

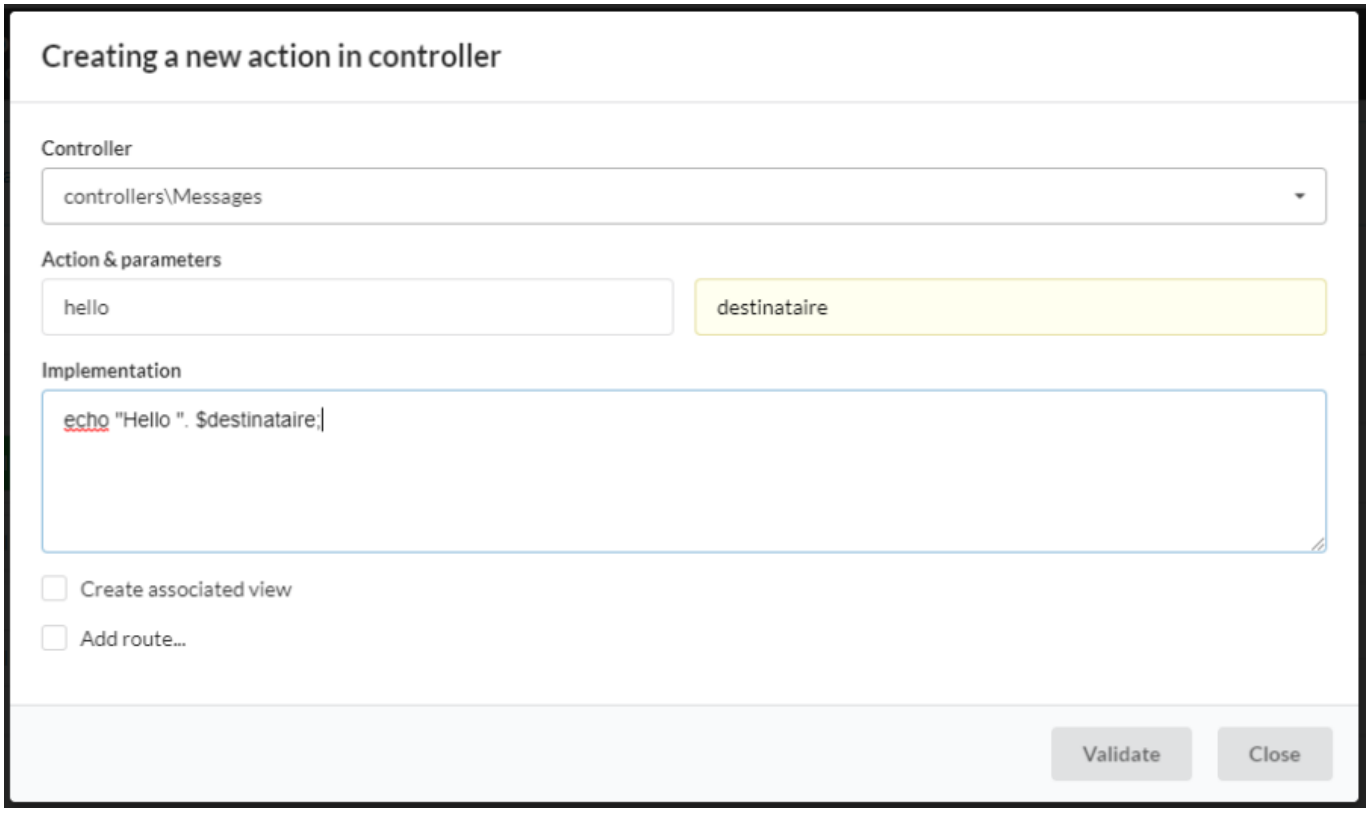
    public function index(){
        echo "Hello world";
    }
}
```

Tester en allant à l'adresse **http://127.0.0.1/tp2/Messages** : l'action par défaut d'un contrôleur est la méthode **index**.

Il est également possible de tester une action à partir de l'interface d'administration, en cliquant sur le bouton GET en face de l'action.

## Action avec paramètre

- Créer l'action **hello** dans le contrôleur **Messages** avec le paramètre **destinataire**




```
<?php
namespace controllers;
 /**
 * Controller Messages
 **/
class Messages extends ControllerBase{

    public function index(){
        echo "Hello world";
    }

    public function hello($destinataire){
        echo "Hello ". $destinataire;
    }
}
```

Tester le résultat aux adresses :

- http://127.0.0.1/tp2/Messages/hello/world
- http://127.0.0.1/tp2/Messages/hello/you !

## Paramètre par défaut d'une action

Ajouter une valeur par défaut au paramètre **$destinataire** de la méthode **hello**

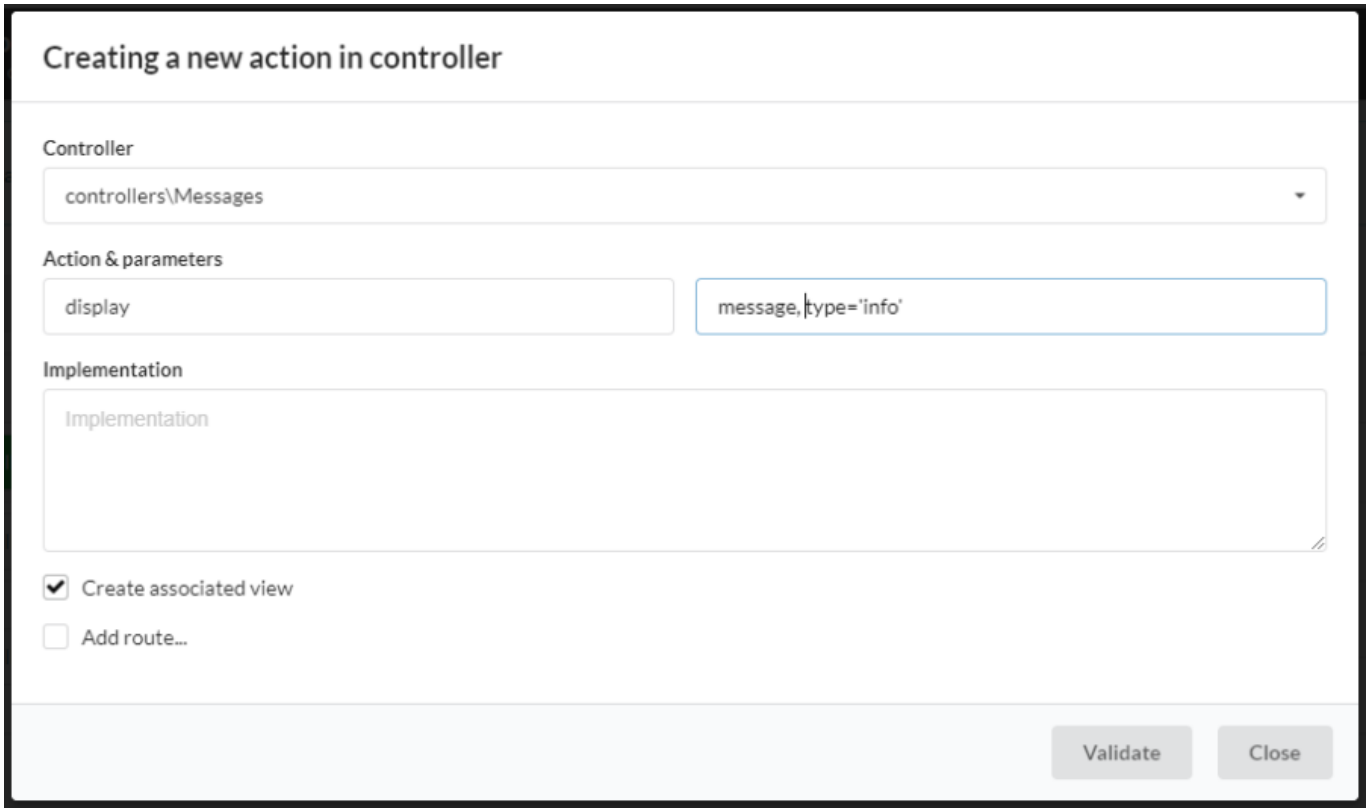
```
<?php
...
    public function hello($destinataire='all the world !'){
        echo "Hello ". $destinataire;
    }
}
```

Tester le résultat aux adresses :

- http://127.0.0.1/tp2/Messages/hello
- http://127.0.0.1/tp2/Messages/hello/world
- http://127.0.0.1/tp2/Messages/hello/you !

## Création d'une action et affichage dans une vue

Créer l'action **display** dans le contrôleur **Messages**, et lui passer les paramètres **message, type='info', icon='info circle'**,
cocher la case **create associated view** à la création :




Modification de l'action :

L'action doit passer les paramètres à la vue (sous forme d'un tableau associatif) :

```
...
    public function display($message,$type='info',$icon='info'){
$this->loadView('Messages/display.html',["message"=>$message,"type"=>$type,"icon"=>
$icon]);
    }
```

Modifier la vue pour qu'elle affiche un message produit par Semantic-ui

```
<div class="ui {{type}} icon message">
    <i class="{{icon}} icon"></i>
    <div class="content">
        {{message}}
    </div>
</div>
```

Tester la page en allant aux adresses : Tester le résultat aux adresses :

- http://127.0.0.1/tp2/Messages/display/Hello
- http://127.0.0.1/tp2/Messages/display/Ceci est un message d'avertissement/warning/warning circle

Il est également possible de passer les variables en utilisant la fonction compact :

```
...
    public function display($message,$type='info',$icon='info'){
        $this->loadView('Messages/display.html',\compact("message","type","icon"));
    }
```

# Application

Il s'agit de créer un jeu permettant de deviner un nombre généré aléatoirement.

## Partie 1

### Template de base

1. Créer dans un seul fichier **app/views/base.html** un template structuré en 4 blocks de la façon suivante :

#### header et menu

Le block **menu** affichera la liste des opérations possibles :

```
<div class="ui secondary  menu">
```

```
    <a href="RandomNumberGame/index" class="active item">
    Home
    </a>
    {% block menu %}
    {% endblock %}
</div>
{% block header %}
    <h1 class="ui header">
        <i class="search icon"></i>
        Random Number Game
    </h1>
    <div class="ui message">
        <div class="header">
            Random number game
        </div>
        <p>Trouvez le nombre aléatoire.</p>
    </div>
{% endblock %}
...
```

#### body

Le block **body** affichera l'opération en cours

```
...
{% block body %}
{% endblock %}
...
```

#### footer

```
...
<div class="ui inverted segment">
{% block footer%}
    <a href="RandomNumberGame/termine" class="ui inverted red button">
        <i class="icon stop"></i>
        Arrêter le jeu
    </a>
{% endblock %}
</div>
```

## Contrôleur/actions

| Contrôleur | Action | Vue | Comportement |
|---|---|---|---|
| **RandomNumberGame** | index | index.html | Appelle la méthode **propose** si un nombre est déjà généré en Session ou affiche le bouton nouvelle partie |
| | propose | propose.html | Affiche le formulaire pour effectuer une proposition |
| | genere | | Génère un nombre aléatoire, le sauvegarde en session, puis appelle la méthode **index** |
| | soumet | soumet.html | Analyse la réponse envoyée par l'utilisateur et affiche le message de réponse dans la vue |
| | termine | | Détruit la variable de session et appelle la méthode **index** |

Le contrôleur définit la constante **SESSION_KEY**, clé qui permettra de sauvegarder le nombre aléatoire en session :

```
 /**
 * Controller RandomNumberGame
 **/
class RandomNumberGame extends ControllerBase{
    const SESSION_KEY="random";
```

**Notions**

Pour générer un nombre aléatoire entre 1 et 10 :

```
$number=\mt_rand(1,10);
```

Pour sauvegarder le nombre en session :

```
USession::set(self::SESSION_KEY, $number);
```

Pour récupérer la variable de session :

```
$number=USession::get(self::SESSION_KEY);
```

Pour vérifier que la variable existe en session :

```
if(USession::exists(self::SESSION_KEY)){
    //Faire quelque chose si la variable existe
}
```

Pour récupérer la variable **number** postée :

```
$number=URequest::post("number");
```

Pour plus d'informations :

- USession

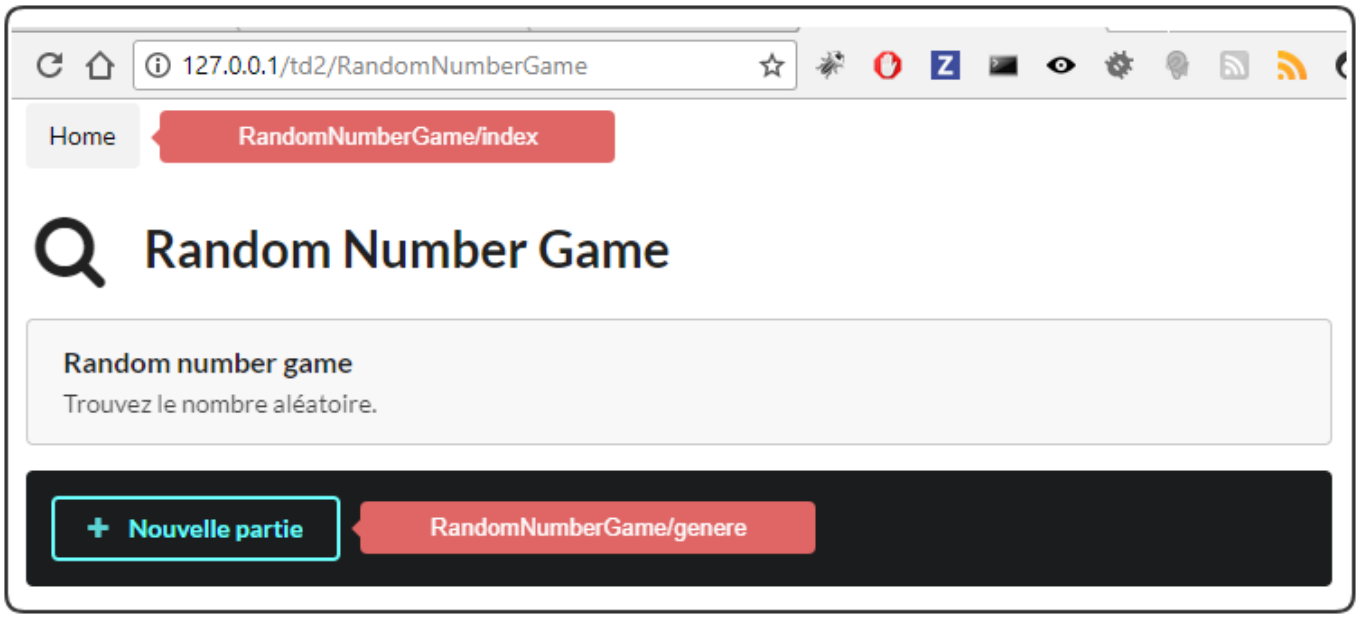
- URequest

## Ecrans

### RandomNumberGame/index

| Contrôleur | Action | Vue | Comportement |
|---|---|---|---|
| **RandomNumberGame** | index | index.html | Appelle la méthode **propose** si un nombre est déjà généré en Session ou affiche le bouton nouvelle partie |

Si aucun nombre n'est généré en session :




La vue **index.html** hérite de **base.html** et redéfinit certains blocks :

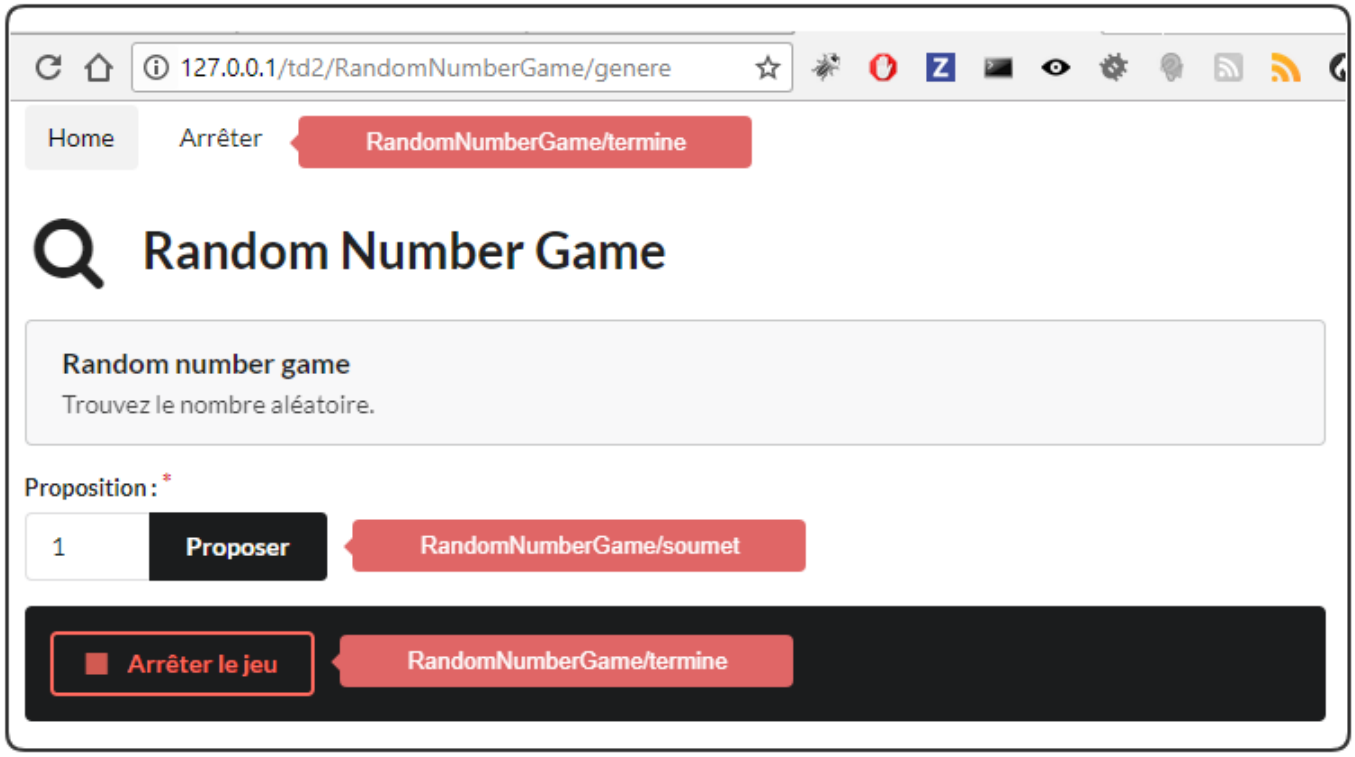
```
{% extends "base.html" %}
{% block body %}

{% endblock %}

{% block footer%}
    <a href="RandomNumberGame/genere" class="ui inverted teal button">
        <i class="ui plus icon"></i>
        Nouvelle partie
    </a>
{% endblock %}
```

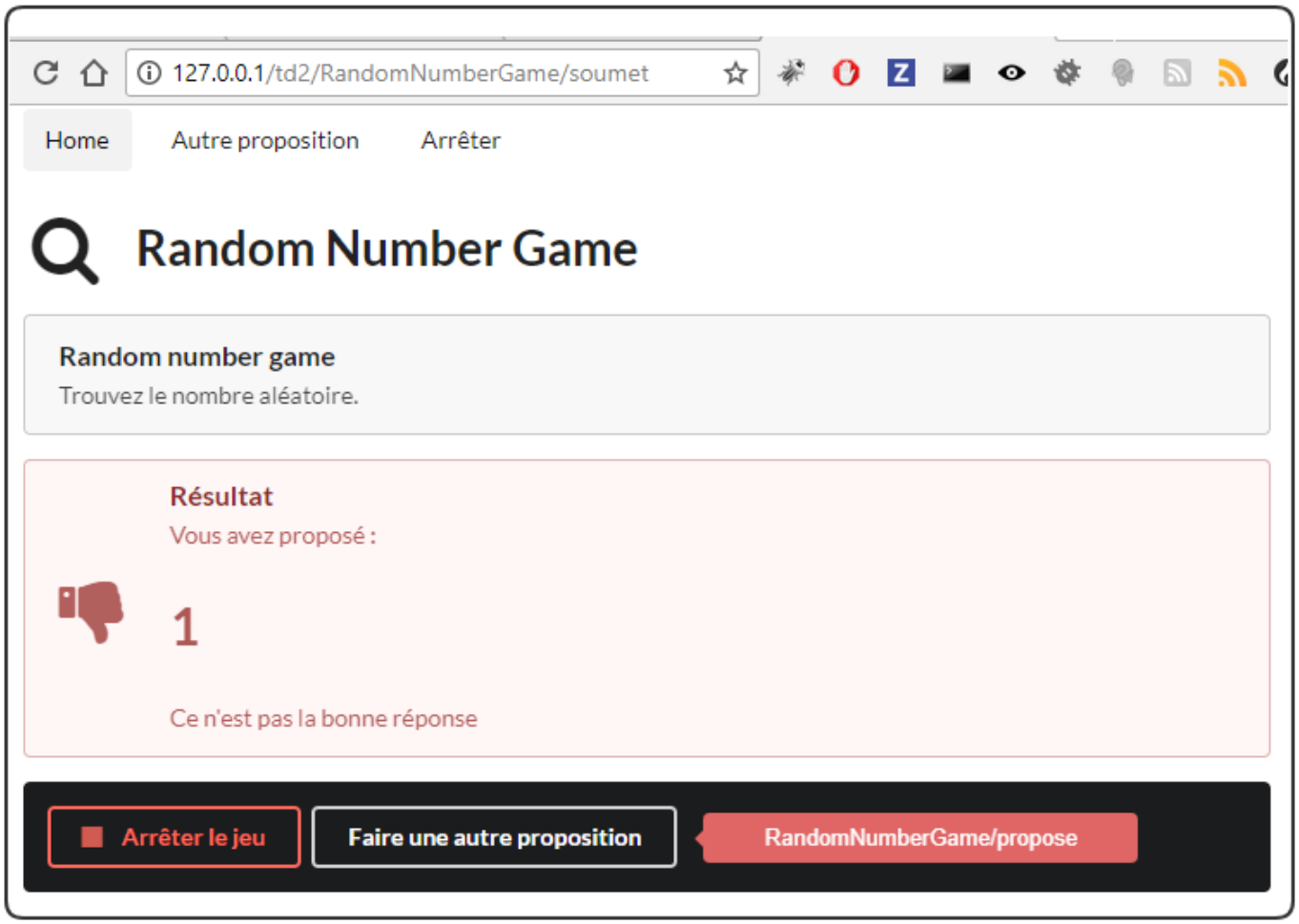
### RandomNumberGame/propose

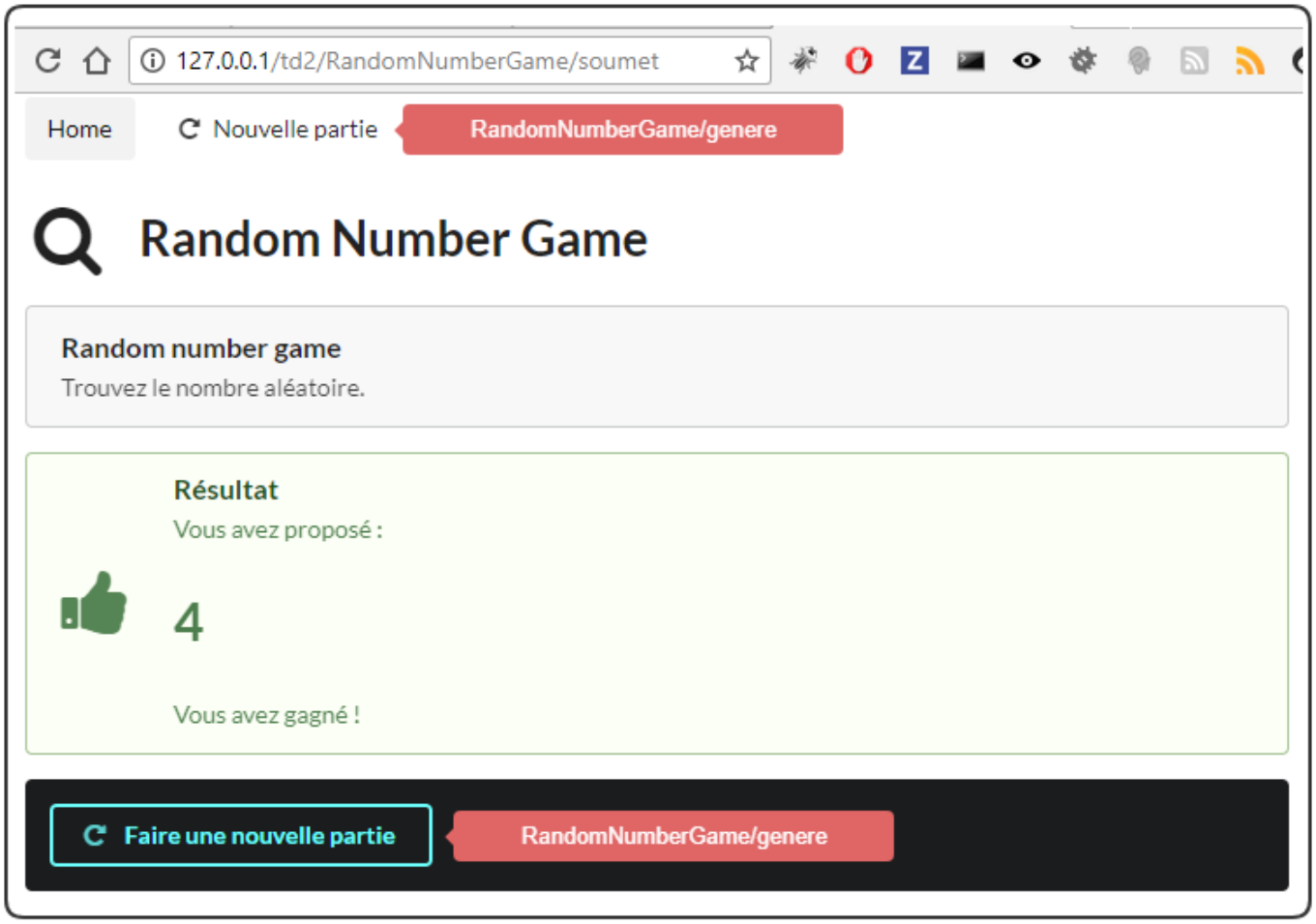
| Contrôleur | Action | Vue | Comportement |
|---|---|---|---|
| **RandomNumberGame** | propose | propose.html | Affiche le formulaire pour effectuer une proposition |




**RandomNumberGame/soumet**

| Contrôleur | Action | Vue | Comportement |
|---|---|---|---|
| **RandomNumberGame** | soumet | soumet.html | Analyse la réponse envoyée par l'utilisateur et affiche le message de réponse dans la vue |






# Partie 2

On souhaite ajouter de nouvelles fonctionnalités au jeu :

- Ajout d'une limite en nombre d'essais pour deviner le nombre
- Création de niveaux de jeu :
    - Facile : génération entre 1 et 10 en 5 essais
    - Intermédiaire : génération entre 1 et 15 en 6 essais
    - Difficile : génération entre 1 et 20 en 7 essais
- Mémorisation et affichage des parties réalisées

## Classe métier

On dispose maintenant d'une classe permettant de gérer le jeu, à intégrer dans le dossier **app/services** du projet :

```
<?php

namespace services;

class Game {
    const TROUVE="trouvé";
    const ABANDON="abandon";
    const PERDU="perdu";
```

```
    const ICONS=[self::TROUVE=>"thumbs green up",self::PERDU=>"thumbs red
down",self::ABANDON=>"window orange close"];
    private $number;
    private $min;
    private $max;
    private $essais;
    private $maxEssais;
    private $statut;
    private $messages;
    public function __construct($min,$max,$maxEssais){
        $this->min=$min;
        $this->max=$max;
        $this->maxEssais=$maxEssais;
        $this->messages=[];
        $this->essais=[];
    }

    protected function addMessage($content,$type="error",$icon=""){
        $this->messages[]=\compact("type","content","icon");
    }

    /**
     * Génére un nouveau nombre aléatoire
     */
    public function generate(){
        $this->number=\mt_rand($this->min,$this->max);
        $this->addMessage("Un nombre aléatoire a été généré","info","info circle");
    }

    /**
     * Permet de proposer un nombre, et de vérifier la proposition
     * @param int $number
     * @return boolean retourne vrai si la partie est gagnée
     */
    public function propose($number){
        if($this->reste()>0){
            if(\array_search($number, $this->essais)!==false){
                $this->addMessage("Vous avez déjà proposé le nombre
{$number}.","info","info circle");
            }else{
                $this->essais[]=$number;
                if($this->number==$number){
                    $this->statut=self::TROUVE;
                    $this->addMessage("Vous avez gagné la partie
!","success","star");
                    return true;
                }else{
                    $this->addMessage("Ce n'est pas la bonne réponse
!","error","warning circle");
                }
            }
        }
        if($this->reste()==0){
            $this->statut=self::PERDU;
            $this->addMessage("Il ne vous reste plus aucun essai, vous avez perdu
!","info","info circle");
```

```
        }else{
            $this->addMessage("Il vous reste {$this->reste()} essai(s) sur
{$this->maxEssais}.","info","info circle");
        }
        return false;
    }

    /**
     * Retourne le nombre aléatoire généré
     * @return number
     */
    public function getNumber() {
        return $this->number;
    }

    /**
     * Retourne la liste au format HTML des propositions effectuées
     * @return string
     */
    public function getEssais() {
        $result=[];
        foreach ($this->essais as $essai){
            if($essai==$this->number){
                $result[]="<u>{$essai}</u>";
            }else{
                $result[]=$essai;
            }
        }
        return \implode(" . ", $result);
    }

    /**
     * Retourne le nombre d'essais restant
     * @return number
     */
    public function reste(){
        return $this->maxEssais-\sizeof($this->essais);
    }

    /**
     * Termine la partie par abandon
     */
    public function terminate(){
        $this->statut=self::ABANDON;
        $this->addMessage("Vous avez abandonné.","","talk");
    }

    /**
     * Retourne la liste des messages émis
     * @return array
     */
    public function getMessages():array{
        return $this->messages;
    }

    /**
```

```
     * Retoune les messages et les marque comme étant lus
     * @return array
     */
    public function readMessages():array{
        $result=[];
        foreach ($this->messages as &$msg){
            if(!isset($msg["read"])){
                $msg["read"]=true;
                $result[]=$msg;
            }
        }
        return $result;
    }

    /**
     * Retourne le statut de la partie (gagné, perdu, abandon)
     * @return string
     */
    public function getStatut():string{
        if(isset($this->statut))
            return $this->statut;
        return "?";
    }

    public function getMaxEssais():int{
        return $this->maxEssais;
    }

    /**
     * Retourne le nombre d'essais réalisés
     * @return int
     */
    public function getNbEssais():int{
        return \sizeof($this->essais);
    }
    /**
     * @return int
     */
    public function getMin():int {
        return $this->min;
    }

    /**
     * @return int
     */
    public function getMax():int {
        return $this->max;
    }

    public function perdu(){
        return $this->statut===self::PERDU;
    }

    public function getStatutIcon(){
        return self::ICONS[$this->statut];
    }
```

```
}
```

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
**http://slamwiki2.kobject.net/php-rt/tp2?rev=1522673606**

Last update: **2019/08/31 14:26**

