elliot

Conception technique

DU CHALARD Wandrille
GOLFIER Maxime
MOUSCADET Marin
TERRIER Alphonse
TO VAN TRANG Bryan
VAVELIN Florian

Groupe G1D - 2017-2018

Table des matières

1 MVC (Modèle, vue, contrôleur)

Le modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web.

Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

Le but de toute cette partie est de retourner les éléments issus de la base de données.

- Un modèle (Model) contient les données ainsi que de la logique en rapport avec les données
- Une vue (View) contient la présentation de l'interface graphique.
- Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.

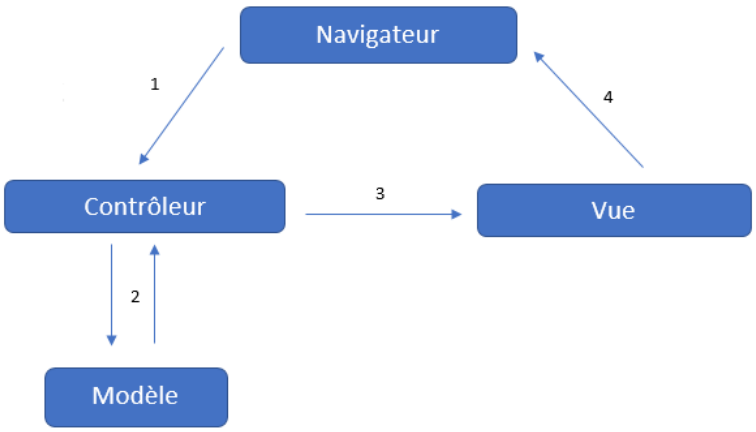

FIGURE 1 – Fonctionnement du MVC

1. Action utilisateur via une requête HTTP
2. Consultation et/ou mise à jour du modèle (facultatif)
3. Le contrôleur décide de la vue à afficher
4. La vue renvoie le HTML au navigateur

Ici, le but est de faire un MVC qui puisse être facilement reportable dans différents cas d'utilisation.

Nous l'utiliserons aussi bien pour ajouter de nouveaux capteurs que pour ajouter de nouveaux utilisateurs.

Nous avons un controller lorsque l'utilisateur est connecté et un controller lorsque celui-ci n'est pas connecté.

Chaque controller dispose d'actions qui lui sont propres.

1.1 Contrôleur

La première étape est d'ajouter des méthodes à un controller. Dans notre cas, nous souhaitons que seuls les utilisateurs connectés puissent gérer les capteurs.

```php
/** @brief Identifie l'action concernant l'authentification et appelle la méth
class ControleurAuth{
    /** @brief C'est dans le contructeur que le contrôleur fait son travail */
    function __construct($action){
        //On distingue des cas d'utilisation suivant l'action
        switch($action){
            case "home":
                require(Config::getVues()["default"]);
                break;
            case "ajoutCapteur" :
                $this->actionAjoutCapteur();
                break;
            case "afficheCapteur" :
                $this->actionAfficheCapteur();
                break;
            case "deleteCapteur" :
                $this->actionDeleteCapteur();
                break;

            default://L'action indéfinie (page par défaut, ici accueil)
                require(Config::getVues()["default"]);
                break;
        }
    }
```

Le but du controller est de faire des appels au model. En fonction du résultat issu de ce modèle, nous afficherons des vues d'erreurs ou de succès.

```
private function actionAjoutCapteur(){
  $model = ModelCapteur::getModelCapteurCreate($_POST);
    if ($model->getError ( ) === false ) {
        require(Config::getVues()["ajoutCapteur"]);
    } else {
        if (!empty($model->getError()['persistance'])){
            // Erreur d'accès à la base de donnée
            require(Config::getVuesErreur()["default"]);
        } else {
            // Erreur de saisie
            require(Config::getVuesErreur()["default"]);
        }
    }
  }

  private function actionAfficheCapteur(){
    $model = ModelCapteur::getModelCapteurDisplay($_POST);
      if ($model->getError ( ) === false ) {
          require(Config::getVues()["afficheCapteur"]);
      } else {
          if (!empty($model->getError()['persistance'])){
              // Erreur d'accès à la base de donnée
              require(Config::getVuesErreur()["default"]);
          } else {
              // Erreur de saisie
              require(Config::getVuesErreur()["default"]);
          }
      }
}

private function actionDeleteCapteur(){
  $model = ModelCapteur::getModelCapteurDelete($_POST);
    if ($model->getError ( ) === false ) {
        require(Config::getVues()["deleteCapteur"]);
    } else {
        if (!empty($model->getError()['persistance'])){
            // Erreur d'accès à la base de donnée
            require(Config::getVuesErreur()["default"]);
        } else {
            // Erreur de saisie
            require(Config::getVuesErreur()["default"]);
        }
    }
  }
```

1.2 Modèle

Le model va interagir avec la base de données. Le résultat est renvoyé par la suite au controller.

```
private function actionAjoutCapteur(){
  $model = ModelCapteur::getModelCapteurCreate($_POST);
    if ($model->getError ( ) === false ) {
        require(Config::getVues()["ajoutCapteur"]);
    } else {
        if (!empty($model->getError()['persistance'])){
            // Erreur d'accès à la base de donnée
            require(Config::getVuesErreur()["default"]);
        } else {
            // Erreur de saisie
            require(Config::getVuesErreur()["default"]);
        }
    }
  }

  private function actionAfficheCapteur(){
    $model = ModelCapteur::getModelCapteurDisplay($_POST);
      if ($model->getError ( ) === false ) {
          require(Config::getVues()["afficheCapteur"]);
      } else {
          if (!empty($model->getError()['persistance'])){
              // Erreur d'accès à la base de donnée
              require(Config::getVuesErreur()["default"]);
          } else {
              // Erreur de saisie
              require(Config::getVuesErreur()["default"]);
          }
      }
}

private function actionDeleteCapteur(){
  $model = ModelCapteur::getModelCapteurDelete($_POST);
    if ($model->getError ( ) === false ) {
        require(Config::getVues()["deleteCapteur"]);
    } else {
        if (!empty($model->getError()['persistance'])){
            // Erreur d'accès à la base de donnée
            require(Config::getVuesErreur()["default"]);
        } else {
            // Erreur de saisie
            require(Config::getVuesErreur()["default"]);
        }
    }
  }
```

1.3 Vues

1.3.1 Configuration

Nous devons configurer les vues que nous allons renvoyer à l'utilisateur dans /mvc/Config/Config.php

Config.php

```
        $db_name="dbname=elliot_db";
        $db_user ="root";
        $db_password="";*/


    }

    /** @brief retourne le tableau des chemins vers les vues */
    public static function getVues()
    {
        global $rootDirectory; //racine du site
        $vueDirectory = $rootDirectory."Vue/"; //répertoire contenant les vues
        return array("default" => $vueDirectory."vueAccueil.php",
                     "connexion" => $vueDirectory."vueConnexion.php",
                    "inscription" => $vueDirectory."vueInscription.php",
                    "index" => $vueDirectory."vueIndex.php",
                    "ajoutCapteur" => $vueDirectory."vueAjoutCapteur.php",
                    "afficheCapteur" => $vueDirectory."vueAfficheCapteur.php",
                    "deleteCapteur" => $vueDirectory."vueDeleteCapteur.php");
    }
```

1.3.2 Création

Nous pouvons maintenant réaliser ces vues.

vueAjoutCapteur.php | vueDeleteCapteur.php | vueAfficheCapteur.php

```
Votre capteur a été supprimé de la base de données
```

2 Modélisation des données

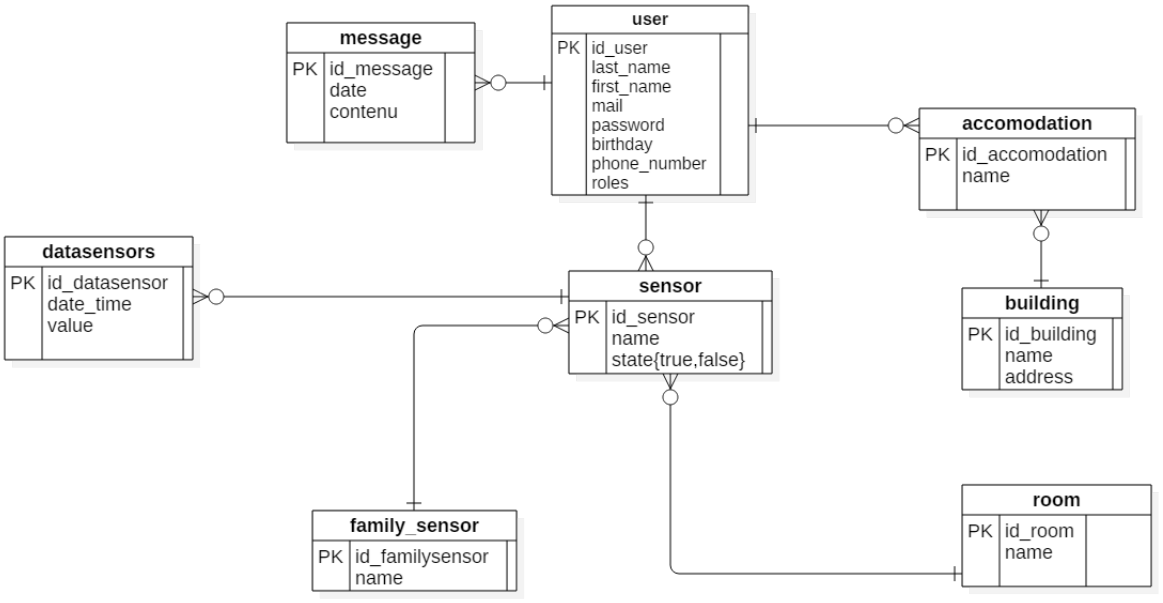

FIGURE 2 – Diagramme des relations d'entités

— **Utilisateur** :
  — Nom = chaine de caractères, obligatoire
  — Prénom = chaine de caractères, obligatoire
  — Mail = mail, obligatoire, identifiant de connexion, envoyer & recevoir un message du support
  — Mot de passe = password, obligatoire, sécurisation du compte utilisateur, crypté en base de données
  — Date de naissance = date, facultatif
  — Numéro de téléphone = entier, facultatif, notifications
  — Rôle = entier, l'administrateur n'a pas le même droit qu'un utilisateur de l'application (cf Modélisation des fonctionnalités)
— **Famille de capteurs** :
  — Nom = chaine de caractères, définit le type de capteurs (température, gyroscope, etc.)
— **Données des capteures** :
  — Datetime = Date, faire un historique des données pour afficher sur le tableau de bord
  — Valeur = entier ou tuple d'entiers, stocke les valeurs recueillies des capteurs

- **Capteurs** :
  - Nom du capteur = chaine de caractères, permettre d'identifier chaque capteur
  - État = booléen, savoir si le capteur est en fonctionnement ou pas
- **Pièce** :
  - Nom = chaine de caractères, permettre d'identifier chaque pièce
- **Habitation** :
  - Nom = chaine de caractères, permettre d'identifier chaque habitation
- **Bâtiment** :
  - Nom = chaine de caractères, permettre d'identifier chaque bâtiment
  - Adresse = chaine de caractères, permettre de localiser chaque bâtiment
- **Message** :
  - Datetime = Date, connaître l'heure d'émission de la notification
  - Contenu = chaine de caractères, message envoyé à l'utilisateur