Le routeur (page = index) appelle le bon contrôleur (controlleur), qui récupère des informations depuis le modèle (model) qu'il passe ensuite à la vue (ca depend de ce qu’on lui a demandé)

INDEX.PHP = récupère les données et appel l’affichage

HOMEVIEW = affiche la vue, grâce au require, il charge les variables transmises par index.php, il sert juste à l’affichage

MODEL.PHP = il sert de contrôleur, il sert d’intérmédiaire entre le modèle et la vue

Architecture en MV afin de bien organiser son code.

MODEL

* On charge le fichier du model
* On appel la fonction ce qui exécute le code à l’intérieur de modèle php. On récupère la liste des billets dans la variable $req
* On charge le fichier de la vue (l’affichage) qui va présenter les informations dans une page HTML
* On ajoute nos classes (db connect = privé)

CONTROLEUR

Il fait un test, un *contrôle*: il vérifie qu'on a bien reçu en paramètre un id dans l'url ( $\_GET['id'] ).  
Si c'est le cas, il appelle les 2 fonctions du modèle dont on va avoir besoin : getPost et getComments . On récupère ça dans des variables, qui seront réutilisées dans la vue : postView.php .

VUE

Dans cette vue, on affiche le billet (récupéré avec $post ) et les commentaires (récupérés dans$comments ). Testez, ça doit fonctionner !

* model.php : le modèle, qui contient différentes fonctions pour récupérer des informations dans la base.
* index.php : le contrôleur de la page d'accueil. Il fait le lien entre le modèle et la vue.
* indexView.php : la vue de la page d'accueil. Elle affiche la page.
* post.php : le contrôleur d'un billet et ses commentaires. Il fait le lien entre le modèle et la vue.
* postView.php : la vue d'un billet et ses commentaires. Elle affiche la page.

TEMPLATE

1. Il définit le titre de la page dans $title. Celui-ci sera intégré dans la balise <title> dans le template.
2. Il définit le contenu de la page dans $content. Il sera intégré dans la balise <body> du template.  
   Comme ce contenu est un peu gros, on utilise une astuce pour le mettre dans une variable. On appelle la fonction ob\_start() (ligne 3) qui "mémorise" toute la sortie HTML qui suit, puis, à la fin, on récupère le contenu généré avec ob\_get\_clean()  (ligne 28) et on met le tout dans $content .
3. Enfin, il appelle le template avec un require. Celui-ci va récupérer les variables $title et $content qu'on vient de créer... pour afficher la page !

INDEX.PHP

ce sera le nom de notre routeur. Le routeur étant le premier fichier qu'on appelle en général sur un site, c'est normal de le faire dans index.php. Il va se charger d'appeler le bon contrôleur.

CONTROLLER.PHP

il contiendra nos contrôleurs dans des fonctions. **On va y regrouper nos anciens index.php et post.php**.

**4étapes pour ajouter des nouveaux commentaires :**

MODEL>

AJOUTE un commentaire à la base de donnée

CONTROLLER>

Appel model pour ajouter le commentaires et transmet à view

INDEX>  teste si on a bien un ID de billet, mais aussi si un nom d'auteur et un message ont bien été envoyés. Si c'est le cas, j'appelle le contrôleur  addComment  , qui appelle le modèle pour enregistrer les informations en base.

LES CLASSES

On créer les classes dans le modèle > on les charges dans le controleur (require once permet de ne pas les charger 2 fois)

CLASS MANAGER : la fonction  dbConnect()  n'est ni  private  ni  public  , elle est...  protected . Pourquoi ?  
Si je l'avais laissée  private  , les fonctions filles dans   PostManager  et  CommentManager  n'auraient pas pu l'appeler. Le type  protected  est identique à  private  , mais il autorise quand même les classes filles à appeler la fonction. Juste ce qu'il nous faut !

ON SUPP le db -> POO puisque les class extends la Manager qui factorise.

NAMESPACE