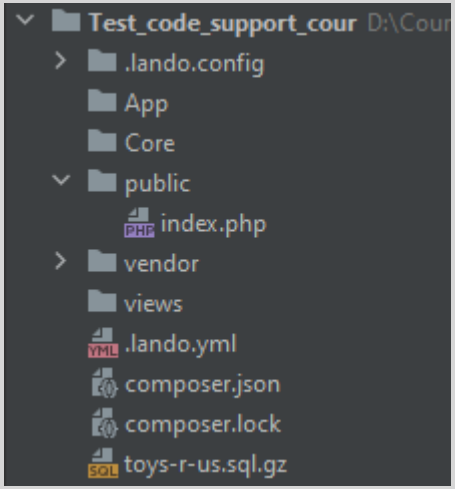
MVC

Préparation de l'architecture

On va préparer l'architecture
Dossier App
Dossier Core
Dossier public
Dossier views



On va créer un fichier index.php dans public

```php
<?php
echo 'welcome';
```

On va utiliser un router depuis une librairie
lien: [Router]

On configure composer.json

```json
{
    "require": {
        "miladrahimi/phprouter": "5.*"
    },
    "autoload": {
        "psr-4": {
            "App\\": "App/",
            "Core\\": "Core/"
        }
    }
}
```

composer dump-autoload

puis le .htaccess (dans le dossier public)

```
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews
    </IfModule>

    RewriteEngine On

    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)/$ /$1 [L,R=301]

    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

On importe la BDD lando db-import nom_de_le_bdd
Dans Core, on crée un dossier Database puis on crée une interface DatabaseConfigInterface.php pour la config de la BDD

```php
namespace Core\Database;

interface DatabaseConfigInterface
{
    public function getHost(): string;
    public function getName(): string;
    public function getUser(): string;
    public function getPass(): string;
}
```

On crée la classe Database

```php
namespace Core\Database;

use \PDO;

class Database
{
    private const PDO_OPTIONS = [
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
    ];

    private static ?PDO $pdoInstance = null;

    public static function getPDO( DatabaseConfigInterface $config ): PDO
    {
        if( is_null( self::$pdoInstance ) ) {
            $dsn = sprintf( format: 'mysql:dbname=%s;host=%s', $config->getName(), $config->getHost() );

            self::$pdoInstance = new PDO( $dsn, $config->getUser(), $config->getPass(), options: self::PDO_OPTIONS );
        }

        return self::$pdoInstance;
    }

    private function __construct() {}
    private function __clone() {}
    private function __wakeup() {}
}
```

Dans App on crée un fichier App.php
Qui implémente l'interface, on déclare nos constantes
Et on définit les méthodes « get » imposé par l'interface

```php
use Core\Database\DatabaseConfigInterface;

class App implements DatabaseConfigInterface
{
    // On déclare les constantes
    private const DB_HOST = 'database';
    private const DB_NAME = 'lamp';
    private const DB_USER = 'lamp';
    private const DB_PASS = 'lamp';

    //On déclare nos méthodes "get" imposé par l'interface
    public function getHost(): string
    {
        return self::DB_HOST;
    }

    public function getName(): string
    {
        return self::DB_NAME;
    }

    public function getUser(): string
    {
        return self::DB_USER;
    }

    public function getPass(): string
    {
        return self::DB_PASS;
    }

}
```

Dans App on configure le router

```php
class App implements DatabaseConfigInterface
{
    // On déclare les constantes
    private const DB_HOST = 'database';
    private const DB_NAME = 'lamp';
    private const DB_USER = 'lamp';
    private const DB_PASS = 'lamp';

    private static ?self $instance = null;
    //méthode appelée au démarrage de l'appli dans index.php
    public static function getApp(): self
    {
        if( is_null( self::$instance )) self::$instance = new self();

        return self::$instance;
    }

    private Router $router;
    public function getRouter(): Router { return $this->router; }

    private function __construct() {
        $this->router = Router::create();
    }
```

```php
    //On aura 3 méthodes à définir
    // 1: méthode start (activation du routeur).
    public function start():void
    {
        //enregistrement des routes
        $this->registerRoutes();
        //démarrage du routeur
        $this->startRouter();
    }
    // 2: méthodes qui enregistrent les routes
    private function registerRoutes(): void
    {
        $this->router->get( path: '/', function (){
            echo 'Utiliser le Controller pour envoyer la vue';
        } );
    }
    // 3: méthode qui démarre le routeur
    private function startRouter(): void
    {
        try {
            $this->router->dispatch();
        }
        catch( RouteNotFoundException $e ) {
            //TODO gérer erreur not found
            echo $e;
        }
        catch( InvalidCallableException $e ) {
            //TODO gérer erreur not valid
            echo $e;
        }
    }

}
```

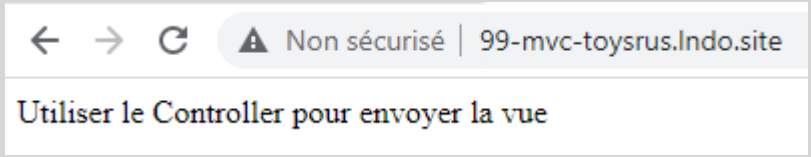
Dans index.php

```php
use App\App;

const DS = DIRECTORY_SEPARATOR;
define("PATH_ROOT", dirname( path: __DIR__) . DS);

require_once PATH_ROOT . 'vendor/autoload.php';

App::getApp()->start();
```



Maintenant, on va devoir définir nos contrôleurs (dans App/Controller)

```php
namespace App\Controller;

interface ControllerInterface
{
    static function redirect(
        string $uri,
        int    $status = 302,
        array  $headers = []
    );
}
```

```php
namespace App\Controller;

use App\App;
use Laminas\Diactoros\Response\RedirectResponse;

abstract class Controller implements ControllerInterface
{
    static function redirect(
        string $uri,
        int    $status = 302,
        array  $headers = []
    )
    {
        $response = new RedirectResponse($uri, $status, $headers);
        App::getApp()->getRouter()->getPublisher()->publish($response);
        die();
    }
}
```

On va se créer notre contrôleur pour la page d’accueil => PageController.php dans App/Controller

```php
namespace App\Controller;

class PageController
{
    public function  index(): string
    {
        return "Welcome";
    }
}
```

Dans App.php on va appeler notre controlleur dans registerRoutes()

```php
private function registerRoutes(): void
{
    $this->router->get( path: '/', [PageController::class, 'index'] );
}
// 3: méthode qui démarre le routeur
```

Welcome

On voit maintenant que notre route passe par le contrôleur, le but maintenant ca va être que le controlleur nous renvoi la bonne view

Dans Core, on va créer la classe View.php

```php
namespace Core;

class View
{
    //on définit le chemin absolu de notre dossier views
    // on réutilise les constantes d'index.php
    public const PATH_VIEWS = PATH_ROOT .'views'. DS;
    // on déclare une propriété titre
    public string $title = 'Titre par défaut';

    public function __construct(private readonly string $name, private $is_complete = false )
    {}

    // on crée la méthode pour récupérer le chemin de la vue
    private function getRequirePath(): string
    {
        $arr_name = explode( separator: '/', $this->name );

        // TODO: Gérer la conformité du tableau obtenu
        $category = $arr_name[0];
        $name = $arr_name[1];
        $name_prefix = $this->is_complete ? '' : '_';

        return self::PATH_VIEWS . $category . DS . $name_prefix . $name . '.html.php';
    }
    //on crée la méthode de rendu
    public function render( ?array $view_data = [] ): void
    {
        if(!empty($view_data)) extract( &array: $view_data );
        // Mise en cache du résultat
        ob_start();

        require_once $this->getRequirePath();

        // Libération du cache
        ob_end_flush();
    }
}
```

Dans PageController

```php
namespace App\Controller;


use Core\View;

class PageController
{
    public function index(): void
    {

        // Préparation des données à transmettre
        $view_data = [
            'title_tag' => 'mon site',
            'list_title' => 'Bienvenue',
            'toy_list' => [
                'jouet 1',
                'jouet 2'
            ]
        ];
        $view = new View( name: 'pages/home');
        $view->title = 'Mon site MVC';

        $view->render($view_data);
    }
}
```

On crée notre vue dans views/page/_home.html.php

```php
<main>
<h1><?php echo $list_title ?></h1>
<?php if( empty( $toy_list ) ): ?>
    <div>Aucun jouet en ce moment</div>
<?php else: ?>
    <ul>
        <?php foreach( $toy_list as $toy ): ?>
            <li><?php echo $toy ?></li>
        <?php endforeach ?>
    </ul>
<?php endif ?>

</main>
```

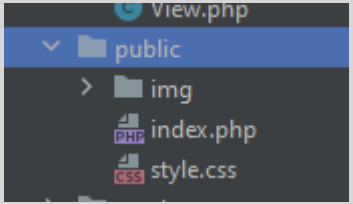
On peut se créer des templates de rendu pour nos vues, on crée un nouveau dossier dans views _templates

Dans _top.html.php

```php
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title><?php echo $title_tag ?></title>
    <!-- import bootstrap -->
    <link
        rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
        integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi"
        crossorigin="anonymous"
    >
    <link rel="stylesheet" href="/style.css">

</head>
<body>
<div id="container">
```




```css
*{
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}

#container {
    background-image: url('img/background.jpg');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: cover;
    display: flex;
    align-items: center;
    flex-direction: column;
    width: 100%;

}
main {
    width: 50%;
    margin-top: 25px;
}

h1 {
    text-align: center;
}
```

Dans _bottom.html.php

```
</div>
<footer>
   <a href="/"> ©2022 - L'IDEM </a>
</footer>

<script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.2/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-kQtW33rZJAHjgefvhyyzcGF3C5TFyBQBA13V1RKPf4uH+bwyzQxZ6CmMZHmNBEfJ"
        crossorigin="anonymous">
</script>
</body>
</html>
```

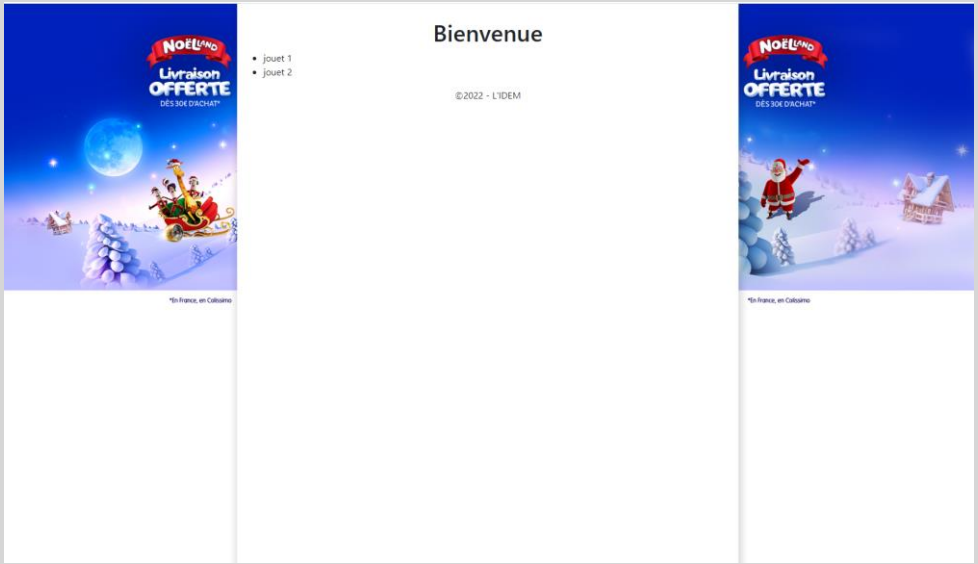
Dans View.php on définit le chemin jusqu'à notre dossier _templates

```
// on réutilise les constantes d'index.php
public const PATH_VIEWS = PATH_ROOT . 'views' . DS;
public const PATH_PARTIALS = self::PATH_VIEWS .'_templates'. DS;
// on déclare une propriété titre
```

Dans la méthode render

```php
public function render(?array $view_data = []): void
{
    if (!empty($view_data)) extract( &array: $view_data);
    // Mise en cache du résultat
    ob_start();
    //on importe le template top
    if (!$this->is_complete) {
        require_once self::PATH_PARTIALS . '_top.html.php';
    }
    // on affiche le contenu
    require_once $this->getRequirePath();
    //on importe le template bottom
    if (!$this->is_complete) {
        require_once self::PATH_PARTIALS . '_bottom.html.php';
    }

    // Libération du cache
```



Disons que pour accéder à la page de la liste des jouets, il faut être connecter. On va devoir créer une nouvelle route (login) pour rediriger sur le formulaire s'il n'est pas connecter ou diriger sur la page d'accueil s'il est en session. Donc il va falloir:

- Créer la route
- Créer contrôleur d'authentification
- Créer vue du formulaire
- Créer classe User
- Créer le gestionnaire de session, etc…

Dans App.php => méthode registerRoutes

```php
$this->router->get( path: '/', [PageController::class, 'index'] );
// on enregistre la route du formulaire de connection
$this->router->get( path: '/connexion', [ AuthController::class, 'index' ] );
```

Maintenant on va créer notre controller AuthController

Dans App/Controller

```php
use App\Session;

/*
 * Users de test:
 * - Administrateur: doe@doe.com - mdp: doe
 * - Inscrit: toto@toto.com - mdp: toto
 */
class AuthController extends Controller
{

    public function index(): void
    {
        $view = new View( name: 'auth/login',  is_complete: true );

        $view_data = [
            'form_result' => Session::get( Session::FORM_RESULT )
        ];


        $view->render( $view_data );
    }
```

Puis on se crée une classe abstraite pour gérer la session « SessionManager » dans Core

```php
namespace Core;

abstract class SessionManager
{
    // pour pouvoir alimenter notre session
    public static function set(string $key, $value): void
    {
        $_SESSION[$key] = $value;
    }

    // pour pouvoir récupérer notre session
    public static function get(string $key)
    {
        if (!isset($_SESSION[$key])) return null;

        return $_SESSION[$key];
    }
    // pour pouvoir vider notre session
    public static function remove(string $key): void
    {
        if (!self::get($key)) return;

        unset($_SESSION[$key]);
    }
}
```

Dans App on crée une classe Session

```php
namespace App;

use Core\SessionManager;

class Session extends SessionManager
{
    public const FORM_RESULT = 'FORM_RESULT';
    public const USER = 'USER';
}
```

On enregistre la route en post

```php
$this->router->post( path: '/login', [ AuthController::class, 'login' ] );
```

On crée notre vue « login.html.php » dans views/auth

```php
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Connexion - Toys`R`Us</title>
</head>
<body>
    <h1>Connexion</h1>
<!--TODO à designer-->
    <form action="/login" method="post">
        <label>
            Email: <input type="email" name="email">
        </label><br>
        <label>
            Mot de passe: <input type="password" name="password">
        </label><br>
        <input type="submit" value="GO !">
    </form>
</body>
</html>
```

On crée maintenant la méthode login dans AuthController

```php
public function login( ServerRequest $request ): void
{
    $post_data = $request->getParsedBody();
    $form_result = new FormResult();
    $user = new User();

    // Si un des champs n'est pas rempli, on ajoute l'erreur au résultat
    if( empty( $post_data['email'] || empty( $post_data['password']) ) ) {
        $form_result->addError( new FormError( message: 'La connexion a échoué, veuillez vérifier les informations saisies.' ) );
    }
    // Sinon on confronte les valeurs saisies avec les données en BDD
    else {
        // Création des infos pour le repository
        $email = $post_data['email'];
        $password = self::hash( $post_data['password'] );

        // Appel du repository
        $user = AppRepoManager::getRm()->getUserRepo()->checkAuth( $email, $password );

        // Si retour négatif, on ajoute l'errer au résultat
        if( is_null( $user ) ) {
            $form_result->addError( new FormError( message: 'La connexion a échoué, veuillez vérifier les informations saisies.' ) );
        }
    }

    // S'il y a des erreurs, on renvoie vers la page de formulaire
    if( $form_result->has_error() ) {
        Session::set( Session::FORM_RESULT, $form_result );
        self::redirect( uri: '/connexion' );
    }

    // Si tout s'est bien passé on enregistre l'utilisateur dans la session (sans son mot de passe)
    $user->password = '';
    Session::set( Session::USER, $user );

    // Puis on redirige vers l'acceuil
    self::redirect( uri: '/' );
}
```

Ici on va devoir créer plusieurs choses:

- La classe FormResult
- La classe FormError
- Le modèle User
- La méthode hash() avec salt & pepper dans AuthController
- La classe AppRepoManager
- Le trait RepositoryManagerTrait

Maintenant on doit gérer le retour de nos formulaires, on crée un nouveau dossier Form dans Core puis on crée 2 nouvelles classes: FormError et FormResult

FormResult

```php
namespace Core\Form;

class FormResult
{
    private string $success_message;
    public function getSuccessMessage(): string { return $this->success_message; }

    private array $form_errors = [];
    public function getErrors(): array { return $this->form_errors; }

    public function __construct( string $success_message = '' )
    {
        $this->success_message = $success_message;
    }

    public function has_error(): bool
    {
        return !empty( $this->form_errors );
    }

    public function addError( FormError $error ): void
    {
        $this->form_errors[] = $error;
    }
}
```

FormError

```php
namespace Core\Form;

class FormError
{
    private string $field_name;
    public function getFieldName(): string { return $this->field_name; }

    private string $message;
    public function getMessage(): string { return  $this->message; }

    public function __construct( string $message, string $field_name = '' )
    {
        $this->field_name = $field_name;
        $this->message = $message;
    }
}
```

Dans la vue login.html.php on peut ajouter l'endroit ou il y aura le message d'erreur

```php
<body>
    <h1>Connexion</h1>
    <?php if( $form_result && $form_result->has_error() ): ?>
        <div style="...">
            <?php echo $form_result->getErrors()[0]->getMessage() ?>
        </div>
    <?php endif ?>
<!--TODO à designer-->
    <form action="/login" method="post">
```

On crée une classe abstraite Model dans Core que l'on pourra étendre aux autres modèles

```php
namespace Core;

abstract class Model
{
    public int $id;

    public function __construct( array $data_row = [] )
    {
        foreach( $data_row as $column => $value ) {
            if(  ! property_exists( $this, $column ) ) continue;

            $this->$column = $value;
        }
    }
}
```

Puis on crée notre modèle User dans App/Model

```php
namespace App\Model;

use Core\Model;

class User extends Model
{
    public const ROLE_SUBSCRIBER = 1;
    public const ROLE_ADMINISTRATOR = 2;

    public string $email;
    public string $password;
    public int $role;
}
```

On s’occupe de la méthode hash() pour les mots de passes

```php
class AuthController extends Controller
{
    public const AUTH_SALT = 'c56a7523d96942a834b9cdc249bd4e8c7aa9';
    public const AUTH_PEPPER = '8d746680fd4d7cbac57fa9f033115fc52196';
```

```php
public static function hash( string $str ): string
{
    $data = self::AUTH_SALT . $str . self::AUTH_PEPPER;

    return hash( algo: 'sha512', $data );
}
```

Puis on crée notre classe AppRepoManager dans App

```php
namespace App;

use App\Model\Repository\UserRepository;
use Core\RepositoryManagerTrait;

class AppRepoManager
{
    //import du trait
    use RepositoryManagerTrait;

    private UserRepository $userRepository;
    public function getUserRepo(): UserRepository { return $this->userRepository; }

    protected function __construct()
    {
        $config = App::getApp();

        $this->userRepository = new UserRepository( $config );
    }
}
```

Ici, on utilise un trait RepositoryManagerTrait
Et on utilise aussi un repository UserRepository.

Le trait dans Core

```php
namespace Core;

// Un trait permet de gérer une portion de code utilisable directement dans plusieurs
// classes sans notion de hiérarchie
// De sorte que self représente ici la classe qui utilise le trait
// Si on était dans un context hiérarchique,
// le mot self désignerait la classe dans laquelle il apparaît
// https://www.php.net/manual/fr/language.oop5.traits.php
trait RepositoryManagerTrait
{
    private static ?self $rm_instance = null;

    public static function getRm(): self
    {
        if( is_null( self::$rm_instance ) ) self::$rm_instance = new self();

        return self::$rm_instance;
    }

    protected function __construct() {}

    private function __clone() {}
    public function __wakeup() {}
}
```

Le repository dans App/Model/Repository qui étend la classe abstraite Repository

```php
namespace App\Model\Repository;

use App\Model\User;
use Core\Repository;

class UserRepository extends Repository
{
    public function getTableName(): string { return 'users'; }

    public function checkAuth( string $email, string $password ): ?User
    {
        $q = sprintf(
            format: 'SELECT * FROM `%s` WHERE `email`=:email AND `password`=:password',
            $this->getTableName()
        );

        $sth = $this->pdo->prepare( $q );

        if( !$sth ) return null;

        $sth->execute( [ 'email' => $email, 'password' => $password ] );

        $user_data = $sth->fetch();

        return empty( $user_data ) ? null : new User( $user_data );
    }
}
```

La classe abstraite Repository dans Core

```php
namespace Core;

use Core\Database\Database;
use Core\Database\DatabaseConfigInterface;
use PDO;

abstract class Repository
{
    protected PDO $pdo;

    abstract public function getTableName(): string;

    public function __construct( DatabaseConfigInterface $config )
    {
        $this->pdo = Database::getPDO( $config );
    }
```

Maintenant on va rediriger sur le formulaire si l'utilisateur n'est pas loggé
Dans AuthController on va créer la méthode isAuth()

```php
public static function isAuth(): bool
{
    return !is_null( Session::get( Session::USER ) );
}
```

Dans la méthode render() de View.php

```php
public function render(?array $view_data = []): void
{
    // ICI on check si l'utilisateur est en session sinon on redirige sur /connexion
    $auth = AuthController::class;
```

Dans _top.html.php

```php
<body>
<?php if( !$auth::isAuth() ) $auth::redirect('/connexion');
```

On peut designer le formulaire de connexion dans login.html.php

```
<body class="d-flex justify-content-center">
<div class="col-4 d-flex flex-column border p-3">

    <?php if ($auth::isAuth()) $auth::redirect('/'); ?>
    <h1>Connexion</h1>
    <?php if ($form_result && $form_result->has_error()): ?>
        <div style="...">
            <?php echo $form_result->getErrors()[0]->getMessage() ?>
        </div>
    <?php endif ?>
    <form action="/login" method="post">
        <div class="mb-3">
            <label for="email" class="form-label">Email</label>
            <input type="email" name="email" class="form-control" id="email" aria-describedby="emailHelp">
            <div id="emailHelp" class="form-text">Ne jamais partager votre email.</div>
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Mot de passe</label>
            <input type="password" name="password" class="form-control" id="password">
        </div>
        <button type="submit" class="btn btn-primary">Se connecter</button>
    </form>
</div>
<body>
</html>
```

On peut designer le formulaire de connexion dans login.html.php

```
<body class="d-flex justify-content-center">
<div class="col-4 d-flex flex-column border p-3">

    <?php if ($auth::isAuth()) $auth::redirect('/'); ?>
    <h1>Connexion</h1>
    <?php if ($form_result && $form_result->has_error()): ?>
        <div style="...">
            <?php echo $form_result->getErrors()[0]->getMessage() ?>
        </div>
    <?php endif ?>
    <form action="/login" method="post">
        <div class="mb-3">
            <label for="email" class="form-label">Email</label>
            <input type="email" name="email" class="form-control" id="email" aria-describedby="emailHelp">
            <div id="emailHelp" class="form-text">Ne jamais partager votre email.</div>
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Mot de passe</label>
            <input type="password" name="password" class="form-control" id="password">
        </div>
        <button type="submit" class="btn btn-primary">Se connecter</button>
    </form>
</div>
<body>
</html>
```

On va maintenant s’occuper d’afficher les jouets sur la page d’accueil, on commence par créer les modèles liés aux jouets (Toy.php, Brand.php, Store.php)

```php
namespace App\Model;

use Core\Model;

class Toy extends Model
{
    // Propriétés correspondant aux
    // colonnes de la table
    public string $name;
    public string $description;
    public int $brand_id;
    public float $price;
    public string $image;
    public string $slug;

    // Propriétés de modélisation
    public ?Brand $brand = null;
}
```

```php
namespace App\Model;

use Core\Model;

class Brand extends Model
{
    public string $name;
}
```

```php
namespace App\Model;

use Core\Model;

class Store extends Model
{
    public string $name;
    public int $postal_code;
    public string $city;
}
```

On crée le repository ToyRepository.php

```php
namespace App\Model\Repository;

use App\Model\Toy;
use Core\Repository;

class ToyRepository extends Repository
{
    public function getTableName(): string { return 'toys'; }

    public function findAll(): array
    {
        return $this->readAll( class_name: Toy::class );
    }

}
```

Ensuite on déclare notre classe dans AppRepoManager

```php
class AppRepoManager
{
    //import du trait
    use RepositoryManagerTrait;

    private UserRepository $userRepository;
    public function getUserRepo(): UserRepository { return $this->userRepository; }

    private ToyRepository $toyRepository;
    public function getToyRepo(): ToyRepository { return $this->toyRepository; }
```

Ensuite on s’occupe du controller

```php
namespace App\Controller;

use App\AppRepoManager;
use Core\View;

class ToyController extends Controller
{
    public function index(): void
    {
        $view_data = [
            'title_tag' => 'Mon site',
            'h1_tag' => 'Nos Jouets',
            'toys' => AppRepoManager::getRm()->getToyRepo()->findAll()
        ];

        $view = new View( name: 'toy/list' );
        $view->title = 'Tous nos jouets - Mon super site MVC';
        $view->render( $view_data );
    }

}
```

Et il nous reste à créer la vue _list.html.php dans views/toy

```php
<h1><?php echo $h1_tag ?></h1>
<?php if (empty($toys)): ?>
    <div>Aucun jouet en ce moment</div>
<?php else: ?>
    <div class="d-flex flex-row flex-wrap justify-content-center col-6">
        <?php foreach ($toys as $toy): ?>
            <div class="card m-2" style="...">
                <img src="/img/<?= $toy->image ?>" class="card-img-top img-fluid p-3" alt="...">
                <div class="card-body">
                    <h3 class="card-title"><?= $toy->name ?></h3>
                    <p class="card-text"><?= $toy->price ?> €</p>
                    <a href="/jouet/<?= $toy->id ?>" class="btn btn-primary">Voir détail</a>
                </div>
            </div>
        <?php endforeach; ?>
    </div>
<?php endif ?>
```

On va maintenant gérer la mécanique pour afficher la vue détail d'un article

- On déclare la route dans App.php

```php
// Déclaration des patterns pour tester les valeurs des arguments
$this->router->pattern( name: 'id', pattern: '[1-9]\d*' );
$this->router->pattern( name: 'slug', pattern: '(\d+-)?[a-z]+(-[a-z-\d]+)*' );

$this->router->get( path: '/', [ToyController::class, 'index'] );
//on définit la route pour accéder au détail
$this->router->get( path: '/jouet/{id}', [ ToyController::class, 'toyById' ] );
```

- On crée la requête dans ToyRepository

```php
public function findById( int $id ): ?Toy
{
    return $this->readById( class_name: Toy::class, $id );
}
```

- On indique le href de la nouvelle route dans _list.html.php

```
<h3 class="card-title"><?= $toy->name ?></h3>
<p class="card-text"><?= $toy->price ?> €</p>
<a href="/jouet/<?= $toy->id ?>" class="btn btn-pri
iv>
```

- On définit notre méthode dans le controller ToyController

```
public function toyById( int $id ): void
{
    $toy_result = AppRepoManager::getRm()->getToyRepo()->findById( $id );

    // Si le jouet n'existe pas on lance l'erreur 404
    if( is_null( $toy_result ) ){
        //TODO gerer l'erreur
        die;
    }

    $view_data = [
        'title_tag' => $toy_result->name,
        'toy' => $toy_result
    ];

    $view = new View( name: 'toy/details' );
    $view->title = $toy_result->name . ' - Mon Super site';

    $view->render( $view_data );
}
```

- On crée la vue dans le dossier views/toy

```php
<div class="d-flex flex-row flex-wrap justify-content-center col-6 m-3">
    <h1><?php echo $title_tag ?> </h1>
    <div class="d-flex mt-3">
        <div class="col-4">
            <img src="/img/<?php echo $toy->image ?>" alt="<?php echo $toy->name ?>">
        </div>
        <div class="col-8">
            <p><?php echo $toy->description ?></p>
        </div>
    </div>

</div>
```

Maintenant on souhaite récupérer le nom de la marque du jouet, dans ToyRepository on va devoir créer une nouvelle requête, mais avant on va devoir créer le repository de Brand et le déclarer dans AppRepoManager.php

On crée le repository BrandRepository.php dans Repository

```php
namespace App\Model\Repository;

use Core\Repository;

class BrandRepository extends Repository
{
    public function getTableName(): string { return 'brands'; }

}
```

Ensuite on l’enregistre dans AppRepoManager.php

```php
private BrandRepository $brandRepository;
public function getBrandRepo(): BrandRepository { return $this->brandRepository; }

protected function __construct()
{
    $config = App::getApp();

    $this->userRepository = new UserRepository( $config );
    $this->toyRepository = new ToyRepository( $config );
    $this->brandRepository = new BrandRepository( $config );
}
```

Dans ToyRepository, on créer la fonction findByIdWithBrand()

```php
public function findByIdWithBrand(int $id): ?Toy
{
    // Je voudrais obtenir un tableau avec toutes les
    // infos à propos d'un Toy, y compris la marque
    // toys.*, brands.name
    $q = sprintf(
    //On crée la requête avec sprintf()
    // on passe les variables avec %1$s et %2$s
        format: 'SELECT
                    `%1$s`.*,
                    `%2$s`.name as brand_name
                FROM `%1$s`
                JOIN `%2$s` ON `%2$s`.id = `%1$s`.brand_id
                WHERE `%1$s`.id=:id',
        // %1$s représente la table toy
        $this->getTableName(),
        // %2$s représente la table brand
        AppRepoManager::getRm()->getBrandRepo()->getTableName()
    );
    $sth = $this->pdo->prepare($q);
    if (!$sth) return null;
    $sth->execute(['id' => $id]);
    $row_data = $sth->fetch();
```

```php
if ( empty( $row_data )  ) return null;
// L'hydrator ne remplit que les données qu'il reconnaît
$toy = new Toy( $row_data );

// On reconstitue un tableau de
// données pour l'hydrator de Brand
$brand_data = [
    'id' => $toy->brand_id,
    'name' => $row_data[ 'brand_name' ]
];

// On crée un objet Brand
$brand = new Brand( $brand_data );

// On ajoute cet objet dans Toy
$toy->brand = $brand;

return $toy;
```

Dans ToyController, on appelle la méthode findByIdWithBrand()

```php
public function toyById( int $id ): void
{
    $toy_result = AppRepoManager::getRm()->getToyRepo()->findByIdWithBrand( $id );

    // Si le jouet n'existe pas on lance l'erreur 404
```

Dans la vue détail on peut afficher la marque

```php
<div class="d-flex flex-row flex-wrap justify-content-center col-6 m-3 ">
    <h1 class="text-success"><?php echo $title_tag ?> </h1>
    <div class="d-flex mt-3">
        <div class=" d-flex flex-column align-items-center col-4">
            <img src="/img/<?php echo $toy->image ?>" alt="<?php echo $toy->name ?>">
            <!-- On ajoute la marque du jouet -->
            <h3 class="text-success mt-3"><?php echo $toy->brand->name ?></h3>
        </div>
        <div class="col-8">
```

Nous allons créer la navbar, dans BrandRepository nous allons créer la méthode getBrandByName()

```php
class BrandRepository extends Repository
{
    public function getTableName(): string
    {
        return 'brands';
    }

    public function getBrandByName(): ?array
    {
        $arr_result = [];
        $q_brand = sprintf(
        //On crée la requête avec sprintf()
        // on passe les variables avec %1$s et %2$s
            format: 'SELECT %1$s.name , %1$s.id ,  COUNT(%1$s.`id`) AS total_brand
                    FROM %1$s
                    INNER JOIN %2$s
                    ON %1$s.id = %2$s.brand_id
                    GROUP BY %1$s.id',
            // %1$s représente la table brand
            $this->getTableName(),
            // %2$s représente la table toy
            AppRepoManager::getRm()->getToyRepo()->getTableName()
        );
        $sth_brand = $this->pdo->query($q_brand);
        if (!$sth_brand) return null;
        while ($row_data_brand = $sth_brand->fetch()) $r_brand[] = new Brand($row_data_brand);
        return $r_brand;
    }
```

Et dans le template _top.html.php on va construire notre navbar

```
    <!--    import cdn icons-->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.2/font/bootstrap-icons.css">

    <link rel="stylesheet" href="/style.css">

</head>
<body>
<?php if( !$auth::isAuth() ) $auth::redirect('/connexion');?>
<div id="container">
    <header>
        <div class="logo"><a href="/"><img src="/img/logo.jpg"></div>
    </header>
    <div id="top-bar">
        <nav id="main-menu" class="menu-hidden">
            <ul class="menu-root">
                <li><a href="/">Tous les jouets</li></a>
                <li><a href="#">Par marque<i class="bi bi-chevron-down"></i></a>
                    <ul>
                        <?php
                        foreach (AppRepoManager::getRm()->getBrandRepo()->getBrandByName() as $r_brand) { ?>
                            <li><a href='brands/<?php echo $r_brand->id ?>'>
                                    <?php echo $r_brand->name ?> ( <?php echo $r_brand->total_brand ?> )</a>
                            </li>
                        <?php } ?>
                    </ul>
            </ul>
        </nav>
    </div>
```

Nous reste à créer la route /brand/{id} dans App.php

```php
$this->router->get( path: '/jouet/{id}', [ ToyController::class, 'toyById' ] );
//on définit la route pour accéder aux jouets par marque
$this->router->get( path: '/brand/{id}', [ ToyController::class, 'toyByBrandId' ] );
// on enregistre la route du formulaire de connection
$this->router->get( path: '/connexion', [ AuthController::class, 'index' ] );
```

Dans ToyController créer la méthode toyBrandById

```php
public function toyByBrandId($id): void
{
    $view_data = [
        'title_tag' => 'Mon site',
        'h1_tag' => 'Nos Jouets par marque',
        'toys' => AppRepoManager::getRm()->getToyRepo()->findAllByBrand($id)
    ];

    $view = new View( name: 'toy/list' );
    $view->title = 'Tous nos jouets - Mon super site MVC';
    $view->render( $view_data );
}
```

Dans ToyRepository créer la méthode findAllByBrand

```php
public function findAllByBrand(int $id): array
{
    return $this->readByBrandId( class_name: Toy::class, $id);
}
```

Dans Repository on crée la méthode readByBrandId

```php
protected function readByBrandId( string $class_name, int $id ): ?array
{
    $arr_result = [];
    $q = sprintf( format: 'SELECT * FROM `%s` WHERE brand_id=:id', $this->getTableName() );

    $sth = $this->pdo->prepare( $q );

    if( !$sth ) return null;

    $sth->execute( [ 'id' => $id ] );

    while( $row_data = $sth->fetch() ) $arr_result[] = new $class_name( $row_data );

    return $arr_result;
}
```

On ajoute un bouton de déconnexion dans la navbar

```
</header>
<div id="top-bar" class="d-flex flex-row justify-content-between">

        <div id="main-menu" class="">
            <ul class="menu-root d-flex flex-row justify-content-between">
                <li>
                    <a href="/">Tous les jouets</a>
                </li>
                <li>
                    <a href="#">Par marque
                        <i class="bi bi-chevron-down"></i>
                    </a>
                    <ul>
                        <?php
                        foreach (AppRepoManager::getRm()->getBrandRepo()->getBrandByName() as $r_brand) { ?>
                            <li>
                                <a href='/brands/<?php echo $r_brand->id ?>'>
                                    <?php echo $r_brand->name ?> ( <?php echo $r_brand->total_brand ?> )</a>
                            </li>
                        <?php } ?>
                    </ul>
                </li>
            </ul>
        </div>

    <div class="d-flex align-items-center m-2">
        <li><a href="/logout" class="logout"><i class="bi bi-box-arrow-left"></i></a></li>
    </div>
</div>
```

On crée la route dans App

```php
$this->router->post( path: '/login', [ AuthController::class, 'login' ] );
// on enregistre la route du logout
$this->router->get( path: '/logout', [ AuthController::class, 'logout' ] );
```

On crée la méthode logout dans AuthController

```php
public function logout(): void
{
    Session::remove( key: Session::USER );
    self::redirect( uri: '/' );
}
```

On va maintenant gérer le rôle de l’utlisateur dans AuthController

```php
private static function hasRole(int $role): bool
{
    $user = Session::get(Session::USER);
      var_dump($user->role);
    if (!($user instanceof User)) return false;

    return $user->role === $role;
}

public static function isSubscriber(): bool
{
    return self::hasRole( role: User::ROLE_SUBSCRIBER);
}

public static function isAdmin(): bool
{
    return self::hasRole( role: User::ROLE_ADMINISTRATOR);
}
```

Dans la navbar, on va créer un menu pour les admins

```php
<div id="main-menu">
    <ul class="menu-root d-flex flex-row justify-content-between">
        <?php if($auth::isAdmin())
        { ?>
            <li>
                <a href="/admin">Accès admin</a>
            </li>
        <?php } ?>
        <li>
            <a href="/">Tous les jouets</a>
        </li>
```

Puis on va créer la route dans App.php

```php
$auth = AuthController::class;
//on enregistre la route admin
if ($auth::isAdmin()) {
    $this->router->get( path: '/admin', function () {
        echo "welcome to admin";
    });
}
```

Puis on crée le fichier AdminController

On modifie la route

```php
//on enregistre la route admin
if ($auth::isAdmin()) {
    $this->router->get( path: '/admin', [AdminController::class, 'index']);
}
```

Dans AdminController, on crée la classe et on ajoute la méthode index (liste des utilisateurs)

```php
namespace App\Controller;

use App\Controller\Controller;

class AdminController extends Controller
{
    public function index()
    {
        return 'Hello';
    }
}
```

Dans UserRepository, on crée la méthode findAll

```php
public function findAll(): array
{
    return $this->readAll( class_name: User::class);
}
```

Dans AdminController, on fait évoluer le code

```php
class AdminController extends Controller
{
    public function index()
    {
        $view_data = [
            'title_tag' => 'Dashboard',
            'h1_tag' => 'Liste des utilisateurs',
            'users' => AppRepoManager::getRm()->getUserRepo()->findAll()
        ];

        $view = new View( name: 'user/list' );
        $view->title = 'Tous les utilisateurs - Mon super site MVC';
        $view->render( $view_data );
    }
}
```

Dans views, on crée un nouveau dossier « user » ainsi qu'un nouveau fichier _list.html.php

```php
<h1><?php echo $h1_tag ?></h1>
<?php if (empty($users)): ?>
    <div>Aucun utilisateur enregistré</div>
<?php else: ?>
    <div class="d-flex flex-row flex-wrap justify-content-center col-6">
        <?php foreach ($users as $user):
            $role = $user->role == 1 ? 'Utilisateur' : 'Administrateur';
            ?>
            <div class="card w-75 m-2">
                <div class="card-body">
                    <h5 class="card-title"><?= $user->email ?></h5>
                    <p class="card-text"><?= $role ?></p>
                    <a href="#" class="btn btn-success">Modifier</a>
                    <a href="#" class="btn btn-danger">Supprimer</a>
                </div>
            </div>
        <?php endforeach; ?>
    </div>
<?php endif ?>
```

On s’occupe de la modification d’un user
on crée la route

```php
if ($auth::isAdmin()) {
    $this->router->get( path: '/admin', [AdminController::class, 'index']);
    $this->router->get( path: '/admin/update/{id}', [AdminController::class, 'update']);
```

On crée la méthode update dans AdminController ou on récupère les infos de l’utilisateur

```php
public function update(int $id)
{
    $view = new View( name: 'user/update');

    $view_data = [
        'form_result' => Session::get(Session::FORM_RESULT),
        'users' => AppRepoManager::getRm()->getUserRepo()->findById($id)
    ];
    $view->render($view_data);
}
```

On crée la vue dans views/user

on crée la route

```
if ($auth::isAdmin()) {
    $this->router->get( path: '/admin', [AdminController::class, 'index']);
    $this->router->get( path: '/admin/update/{id}', [AdminController::class, 'update']);
```

On crée la méthode update dans AdminController ou on récupère les infos de l'utilisateur

```
public function update(int $id)
{
    $view = new View( name: 'user/update');

    $view_data = [
        'form_result' => Session::get(Session::FORM_RESULT),
        'users' => AppRepoManager::getRm()->getUserRepo()->findById($id)
    ];
    $view->render($view_data);
}
```

Et on va sur la vue _update.html.php dans views/user

```php
<div class="col-4 d-flex flex-column border p-3">

    <h1>Modifier l'utilisateur</h1>
    <?php if ($form_result && $form_result->has_error()): ?>
        <div style="...">
            <?php echo $form_result->getErrors()[0]->getMessage() ?>
        </div>
    <?php endif ?>
    <form action="/update" method="post">
        <div class="mb-3">
            <input type="hidden" value="<?= $users->id ?>" name="id" class="form-control" id="email"
                   aria-describedby="emailHelp">
            <input type="hidden" value="<?= $users->password ?>" name="password" class="form-control" id="password">
            <label for="email" class="form-label">Email</label>
            <input type="text" value="<?= $users->email ?>" name="email" class="form-control" id="email"
                   aria-describedby="emailHelp">
        </div>
        <div class=" d-flex flex-column mb-3">
            <div>
                <label for="role" class="form-label">Son rôle:</label>
            </div>
            <div>
                <input type="radio" name="role" value="1" <?=  $users->role == 1 ? 'checked' : ''?>>
                <label for="role">Utilisateur</label>
            </div>
            <div>
                <input type="radio" name="role" value="2" <?=  $users->role == 2 ? 'checked' : ''?>>
                <label for="role">Administrateur</label>
            </div>
        </div>
        <button type="submit" class="btn btn-success">Enregistrer</button>
    </form>
</div>
```

- On crée maintenant la route /update (qui correspond à action)

```php
//on enregistre la route admin
if ($auth::isAdmin()) {
    $this->router->get( path: '/admin', [AdminController::class, 'index']);
    $this->router->get( path: '/admin/update/{id}', [AdminController::class, 'update']);

    $this->router->post( path: '/update', [AdminController::class, 'updateUser']);
}
// Déclaration des patterns pour tester les valeurs des arguments
```

On crée la méthode updateUser dans UserRepository

```php
public function updateById(string $email, int $role, int $id): ?User
{
    $q = sprintf(
        format: 'UPDATE `%s` SET `email`=:email, `role`=:role WHERE id=:id',
        $this->getTableName()
    );
    $sth = $this->pdo->prepare( $q );

    if( !$sth ) return null;

    $sth->execute( [ 'email' => $email, 'role' => $role, 'id' => $id ] );

    $user_data = $sth->fetch();

    return empty( $user_data ) ? null : new User( $user_data );
}
```

- On crée la méthode updateUser dans AdminController

```php
public function updateUser(ServerRequest $request): void
{
    $post_data = $request->getParsedBody();
    $form_result = new FormResult();
    // Si un des champs n'est pas rempli, on ajoute l'erreur au résultat
    if (empty($post_data['email'] || empty($post_data['role']))) {
        $form_result->addError(new FormError( message: 'La modification a échoué, veuillez saisir tous les champs.'));
    } // Sinon on confronte les valeurs saisies avec les données en BDD
    else {
        // Création des infos pour le repository
        $email = $post_data['email'];
        $role = $post_data['role'];
        $id = $post_data['id'];
        // Appel du repository
        $user = AppRepoManager::getRm()->getUserRepo()->updateById($email, $role, $id);
        // S'il y a des erreurs, on renvoie vers la page de formulaire
        if ($form_result->has_error()) {
            Session::set(Session::FORM_RESULT, $form_result);
            self::redirect( uri: '/admin/update/' . $id);
        }
        // Sinon on redirige vers l'accueil
        self::redirect( uri: '/admin');
    }
}
```