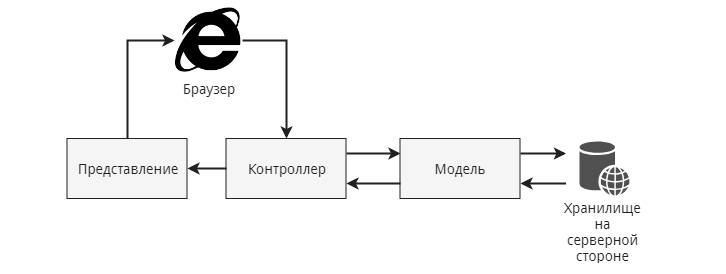
Engine websites





Паттерн MVC (Model-View-Controller) - это архитектурный шаблон, который используется для организации и структурирования кода в веб-приложениях и других программных системах. MVC разделяет приложение на три основных компонента: Model, View и Controller, каждый из которых выполняет свои уникальные функции.

Модель (Model):

Модель представляет собой бизнес-логику, данные и правила, связанные с обработкой данных. Она отвечает за взаимодействие с базой данных, получение, обработку и хранение данных. Модель не зависит от представления и контроллера, что делает её независимой и переиспользуемой. Модель также может содержать методы для выполнения операций CRUD (создание, чтение, обновление и удаление данных).

Представление (View):

Представление отвечает за отображение данных пользователю. Оно генерирует HTML, которое будет показано пользователю, основываясь на данными, предоставленными моделью. Представление может включать в себя шаблоны, стили и другие элементы, создающие пользовательский интерфейс. Однако представление не должно содержать бизнес-логику или непосредственно взаимодействовать с данными.

Контроллер (Controller):

Контроллер является посредником между моделью и представлением. Он обрабатывает пользовательские запросы, вызывает соответствующие методы модели для получения или обработки данных и выбирает соответствующий шаблон представления для отображения результатов. Контроллер также отвечает за передачу данных между моделью и представлением.

Преимущества паттерна MVC:

Разделение ответственностей: MVC разделяет код на четыре отдельных компонента, позволяя разработчикам легче поддерживать и расширять приложение.

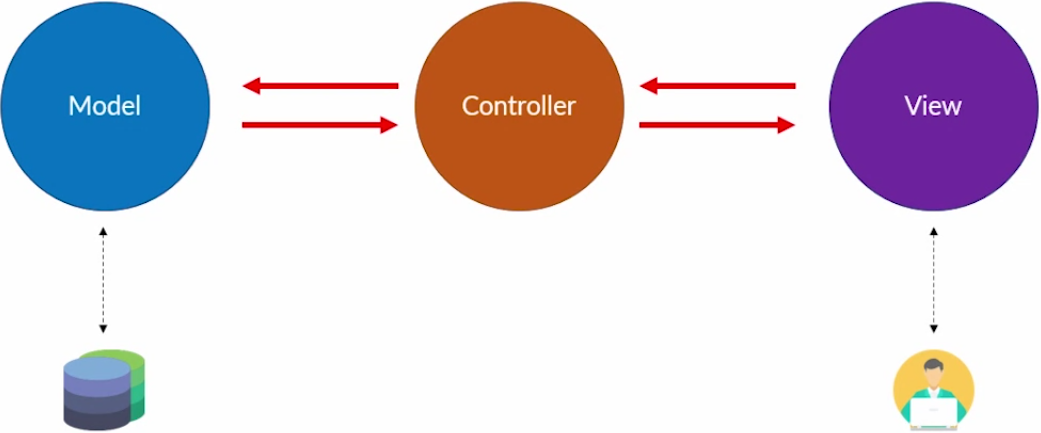
Улучшенная переиспользуемость: Модель может быть переиспользована в различных частях приложения или даже в других проектах.

Гибкость и масштабируемость: Каждый компонент может быть изменен или заменен без влияния на другие компоненты.

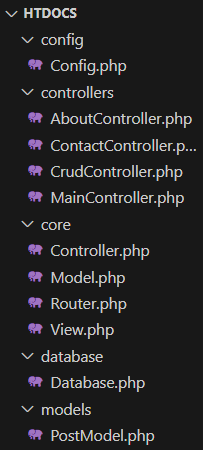
Легкость тестирования: Поскольку каждый компонент имеет четко определенную ответственность, их можно тестировать независимо друг от друга.

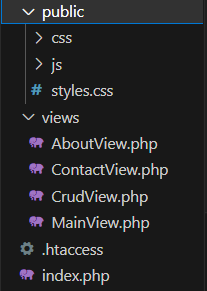
Паттерн MVC широко применяется в разработке веб-приложений, так как помогает создавать чистый, структурированный и легко поддерживаемый код. Однако, при реализации стоит учесть особенности и требования конкретного проекта.

Реализация движка сайта на базе паттерна MVC



Структура проекта:





В папке public есть 2 папки css и js эти папки являются локально скачанными от фреймворка bootstrap по это ссылке: <https://getbootstrap.com/docs/4.0/getting-started/download/>

Так же в папке public есть отдельный файл styles.css этот файл создаем мы сами и он отвечает за глобальную стилизацию для всех представлений проекта.

Для быстрого внедрения в проект и понимания работы движка сайта на базе паттерна MVC мы сначала создадим простые страницы с выводом на экран на каждой странице ее имя.

Let’s start:

Создание абстрактного движка:

core/Controller.php

<?php

abstract class Controller {

    abstract public function index();

}

?>

core/ Model.php

<?php

require\_once 'database/Database.php';

abstract class Model {

    protected $db;

    public function \_\_construct() {

        $this->db = new Database();

    }

    // Создание записи

    abstract public function create($data);

    // Получение записи по id

    abstract public function read($id);

    // Обновление записи

    abstract public function update($id, $data);

    // Удаление записи

    abstract public function delete($id);

}

?>

core/View

<?php

abstract class View {

    abstract public function render($data = []);

}

?>

core/ Router.php (Как таковым не является обязательным для реализации паттерна, но тогда роутинг прийдется реализовать вручную)

ucfirst — Преобразует первый символ строки в верхний регистр

$\_GET['route']: Здесь происходит доступ к значению параметра 'route' из суперглобального массива $\_GET. В PHP массив $\_GET содержит данные, отправленные на скрипт с использованием HTTP-метода GET.

??: Это оператор объединения с null. Он проверяет значение слева ($\_GET['route']) на наличие null. Если значение не null, то оно будет использовано. В противном случае будет использовано значение справа ('main').

Таким образом, данная строка кода устанавливает переменную $route равной значению параметра 'route' из URL, если он присутствует в запросе. Если параметр отсутствует или его значение равно null, то переменной $route будет присвоено значение 'main'.

<?php

class Router {

    public function route() {

$route = $\_GET['route'] ?? 'main';

        $controllerName = ucfirst($route) . 'Controller';

        $viewName = ucfirst($route) . 'View';

        if (file\_exists('controllers/' . $controllerName . '.php') &&

            file\_exists('views/' . $viewName . '.php')) {

            require\_once 'controllers/' . $controllerName . '.php';

            require\_once 'views/' . $viewName . '.php';

            $controller = new $controllerName();

            $controller->index();

        } else {

            echo "404 - Not Found";

        }

    }

}

?>

config/Config.php

self - это ключевое слово в PHP, которое используется внутри классов для обращения к статическим свойствам и методам того же класса, в котором оно используется.

<?php

class Config {

    private static $databaseConfig = [

        'dbname' => 'mvc\_db',

        'dsn' => 'mysql:host=localhost',

        'username' => 'root',

        'password' => ''

    ];

    public static function getDatabaseConfig() {

        return self::$databaseConfig;

    }

}

?>

В PHP ключевое слово self используется внутри классов для обращения к статическим свойствам или методам этого же класса. Это позволяет вам обращаться к свойствам и методам класса без создания экземпляра этого класса. Пример:

<?php

class Example {

    public static $property = "Hello, world!";

    public static function showMessage() {

        echo self::$property;

    }

}

Example::showMessage();  // Выведет: Hello, world!

?>

В данном примере, у нас есть класс Example со статическим свойством $property и статическим методом showMessage(). Статические элементы принадлежат самому классу, а не его экземплярам.

Ключевое слово self используется внутри метода showMessage(), чтобы обратиться к статическому свойству $property класса Example. В этом контексте self указывает на текущий класс (в данном случае, Example).

database/Database.php

В этом классе мы создаем подключение к бд, а так же проверяем ее существование и в ней же создаем сразу таблицу posts для нашей будущей модели.

<?php

require\_once 'config/Config.php';

class Database {

    private $connection;

    public function \_\_construct() {

        $config = Config::getDatabaseConfig();

        try {

            $this->connection = new PDO(

                $config['dsn'],

                $config['username'],

                $config['password']

            );

            $this->connection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

            $this->createDatabase();

            $this->useDatabase();

            $this->createTable();

        } catch (PDOException $e) {

            echo 'Database connection failed: ' . $e->getMessage();

        }

    }

    private function createDatabase() {

        $config = Config::getDatabaseConfig();

        $dbName = $config['dbname'];

        try {

            $createDbQuery = "CREATE DATABASE IF NOT EXISTS $dbName";

            $this->connection->exec($createDbQuery);

        } catch (PDOException $e) {

            echo 'Database creation failed: ' . $e->getMessage();

        }

    }

    private function useDatabase() {

        $config = Config::getDatabaseConfig();

        $dbName = $config['dbname'];

        try {

            $useDbQuery = "USE $dbName";

            $this->connection->exec($useDbQuery);

        } catch (PDOException $e) {

            echo 'Database selection failed: ' . $e->getMessage();

        }

    }

    private function createTable() {

        $createQuery = "

            CREATE TABLE IF NOT EXISTS posts (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                title VARCHAR(255) NOT NULL,

                content VARCHAR(512) NOT NULL

            )

        ";

        try {

            $this->connection->exec($createQuery);

        } catch (PDOException $e) {

            echo 'Table creation failed: ' . $e->getMessage();

        }

    }

    public function executeQuery($query, $params = []) {

        $stmt = $this->connection->prepare($query);

        $stmt->execute($params);

        return $stmt;

    }

}

?>

index.php

точка входа в проект Здесь мы создаем объект Router и сразу запускаем метод route()

<?php

require\_once 'core/Router.php';

$router = new Router();

$router->route();

?>

.htaccess

Конфиг файл для webserver за счет него мы всегда будем сначала заходить в главную страницу

Options -Indexes

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^(.\*)$ index.php?route=$1 [QSA,L]

Файл .htaccess является файлом конфигурации для сервера веб-приложений Apache. Он используется для определения различных настроек, включая перенаправления URL, правила маршрутизации и многое другое.

В нашем случае, файл .htaccess используется для настройки "человеко-понятных" URL и маршрутизации на основе паттерна MVC. Вот подробное объяснение каждой строки в файле:

Options -Indexes: Эта строка запрещает серверу Apache отображать списки файлов в каталогах, если в каталоге нет файла индекса (например, index.php или index.html).

RewriteEngine On: Эта строка включает движок перезаписи URL сервера Apache, который позволяет нам создавать пользовательские правила перенаправления.

RewriteCond %{REQUEST\_FILENAME} !-d: Эта строка задает условие для перенаправления: если запрашиваемый URL не является директорией на сервере.

RewriteCond %{REQUEST\_FILENAME} !-f: Эта строка также задает условие для перенаправления: если запрашиваемый URL не является существующим файлом на сервере.

RewriteRule ^(.\*)$ index.php?route=$1 [QSA,L]: Эта строка задает фактическое правило перенаправления. Она говорит, что все запросы, которые не соответствуют существующим директориям или файлам, должны быть перенаправлены на index.php, добавляя параметр route в виде всего запроса. [QSA,L] - это опции перенаправления. QSA (Query String Append) сохраняет существующие параметры запроса, а L (Last) указывает на то, что это последнее правило для перенаправления.

.htaccess позволяет вам использовать "человеко-понятные" URL, например, http://localhost/about, вместо более сложных запросов с параметрами, и перенаправляет все такие запросы на ваш index.php, где вам будет удобно обрабатывать их согласно паттерну MVC.

controllers/AboutController.php

Здесь создается объект будущего представления и запускается метод render() аналогично будет и в других controllers

<?php

require\_once 'core/Controller.php';

require\_once 'views/AboutView.php';

class AboutController extends Controller {

    public function index() {

        $view = new AboutView();

        $view->render();

    }

}

?>

controllers/ContactController.php

<?php

require\_once 'core/Controller.php';

require\_once 'views/ContactView.php';

class ContactController extends Controller {

    public function index() {

        $view = new ContactView();

        $view->render();

    }

}

?>

controllers/MainController.php

В этом контролере создаем модель PostModel т.к. мы будем на главной странице выводить список всех постов в метод render() передаем массив постов.

<?php

require\_once 'core/Controller.php';

require\_once 'views/MainView.php';

require\_once 'models/PostModel.php';

class MainController extends Controller {

    public function index() {

        $postModel = new PostModel();

        $posts = $postModel->listPosts();

        $view = new MainView();

        $view->render(['posts' => $posts]);

    }

}

?>

controllers/CrudController.php

Это отдельный контролер для модерирования постов.

<?php

require\_once 'core/Controller.php';

require\_once 'views/CrudView.php';

require\_once 'models/PostModel.php';

class CrudController extends Controller {

    public function index() {

        //create update delete

        if(isset($\_POST['btnCreate'])) $this->createPost();

        if(isset($\_POST['btnUpdate'])) $this->updatePost();

        if(isset($\_POST['btnDelete'])) $this->deletePost();

        $view = new CrudView();

        $view->render();

    }

    public function createPost() {

        if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

            $data = $\_POST;

            $postModel = new PostModel();

            $postModel->create($data);

        }

    }

    public function updatePost() {

        if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

            $data = $\_POST;

            $postModel = new PostModel();

            $postModel->update($data['id'], $data);

        }

    }

    public function deletePost() {

        if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

            $id = $\_POST['id'];

            $postModel = new PostModel();

            $postModel->delete($id);

        }

    }

}

?>

models/PostModel.php

Это простая модель в которой мы переопределяем методы crud от базового класса т.к. в абстрактном методе Model у нас нет абстрактного метода listPosts() из-за этого мы его добавим самостоятельно.

<?php

require\_once 'core/Model.php';

class PostModel extends Model {

    public function create($data) {

        $query = "INSERT INTO posts (title, content) VALUES (?, ?)";

        $this->db->executeQuery($query, [$data['title'], $data['content']]);

    }

    public function read($id) {

        $query = "SELECT \* FROM posts WHERE id = ?";

        return $this->db->executeQuery($query, [$id])->fetch(PDO::FETCH\_ASSOC);

    }

    public function update($id, $data) {

        $query = "UPDATE posts SET title = ?, content = ? WHERE id = ?";

        $this->db->executeQuery($query, [$data['title'], $data['content'], $id]);

    }

    public function delete($id) {

        $query ="DELETE FROM posts WHERE id = ?";

        $this->db->executeQuery($query, [$id]);

    }

    public function listPosts() {

        $query = "SELECT \* FROM posts";

        return $this->db->executeQuery($query)->fetchAll(PDO::FETCH\_ASSOC);

    }

}

?>

views/MainView.php

Здесь сказать нечего, это обычное представление в котором есть метод render() и его задача передать в браузер html разметку, подобным образом будут реализованы и другие представления.

<?php

require\_once 'core/View.php';

class MainView extends View {

    public function render($data = []) {

        // Выводим Bootstrap-ссылку и разметку

        echo "<!DOCTYPE html>

<html>

<head>

    <title>Main View</title>

    <link rel='stylesheet' href='./public/css/bootstrap.min.css'>

    <link rel='stylesheet' href='./public/styles.css'>

</head>

<body>

    <div class='container'>

    <a href='crud' class='btn btn-danger'>crud page</a>

    <h1>Main View</h1>";

    // Выводим список постов

    if (isset($data['posts'])) {

        echo "<ul class='list-group'>";

        foreach ($data['posts'] as $post) {

            echo "<li class='list-group-item'>

                <strong>id => {$post['id']}</strong><br>

                <strong>title => {$post['title']}</strong> <br>

                <strong>content => {$post['content']}<strong><br>

            </li>";

        }

        echo "</ul>";

    }

    echo "</div>

</body>

</html>";

    }

}

views/AboutView.php

<?php

require\_once 'core/View.php';

class AboutView extends View {

    public function render($data = []) {

        // Выводим Bootstrap-ссылку и разметку

        echo "<!DOCTYPE html>

<html>

<head>

    <title>About View</title>

    <link rel='stylesheet' href='./public/css/bootstrap.min.css'>

    <link rel='stylesheet' href='./public/styles.css'>

</head>

<body>

    <div class='container'>

        <h1>About View</h1>

        <p>This is the AboutView content.</p>

    </div>

</body>

</html>";

    }

}

?>

view/ContactView.php

<?php

require\_once 'core/View.php';

class ContactView extends View {

    public function render($data = []) {

        // Выводим Bootstrap-ссылку и разметку

        echo "<!DOCTYPE html>

<html>

<head>

    <title>Contact View</title>

    <link rel='stylesheet' href='./public/css/bootstrap.min.css'>

    <link rel='stylesheet' href='./public/styles.css'>

</head>

<body>

    <div class='container'>

        <h1>Contact View</h1>

        <p>This is the ContactView content.</p>

    </div>

</body>

</html>";

    }

}

?>

CrudView.php

<?php

require\_once 'core/View.php';

class CRUDView extends View {

    public function render($data = []) {

        // Выводим Bootstrap-ссылку и разметку

        echo "<!DOCTYPE html>

<html>

<head>

    <title>CRUD Page</title>

    <link rel='stylesheet' href='./public/css/bootstrap.min.css'>

    <link rel='stylesheet' href='./public/styles.css'>

</head>

<body>

    <div class='container'>

        <h1>CRUD Page</h1>

        <a href='/' class='btn btn-primary'>main page</a>";

        // Форма для создания поста

        echo "<h2>Create Post</h2>

        <form class='form-group' action='crud' method='post'>

            <input class='form-control' type='text' name='title' placeholder='Title'>

            <textarea class='form-control' name='content' placeholder='Content'></textarea>

            <button class='btn btn-success' type='submit' name='btnCreate'>Create</button>

        </form>";

        // Форма для обновления поста

        echo "<h2>Update Post</h2>

        <form class='form-group' action='crud' method='post'>

            <input class='form-control' type='text' name='id' placeholder='id'>

            <input class='form-control' type='text' name='title' placeholder='Title'>

            <textarea class='form-control' name='content' placeholder='Content'></textarea>

            <button class='btn btn-warning' type='submit' name='btnUpdate'>Update</button>

        </form>";

        // Форма для удаления поста

        echo "<h2>Delete Post</h2>

        <form class='form-group' action='crud' method='post'>

            <input class='form-control' type='text' name='id' placeholder='id'>

            <button class='btn btn-danger' type='submit' name='btnDelete'>Delete</button>

        </form>";

        echo "</div>

</body>

</html>";

    }

}

?>

public/styles.css

Глобальные стили для всего проекта.

/\* styles.css \*/

/\* Общие стили \*/

body {

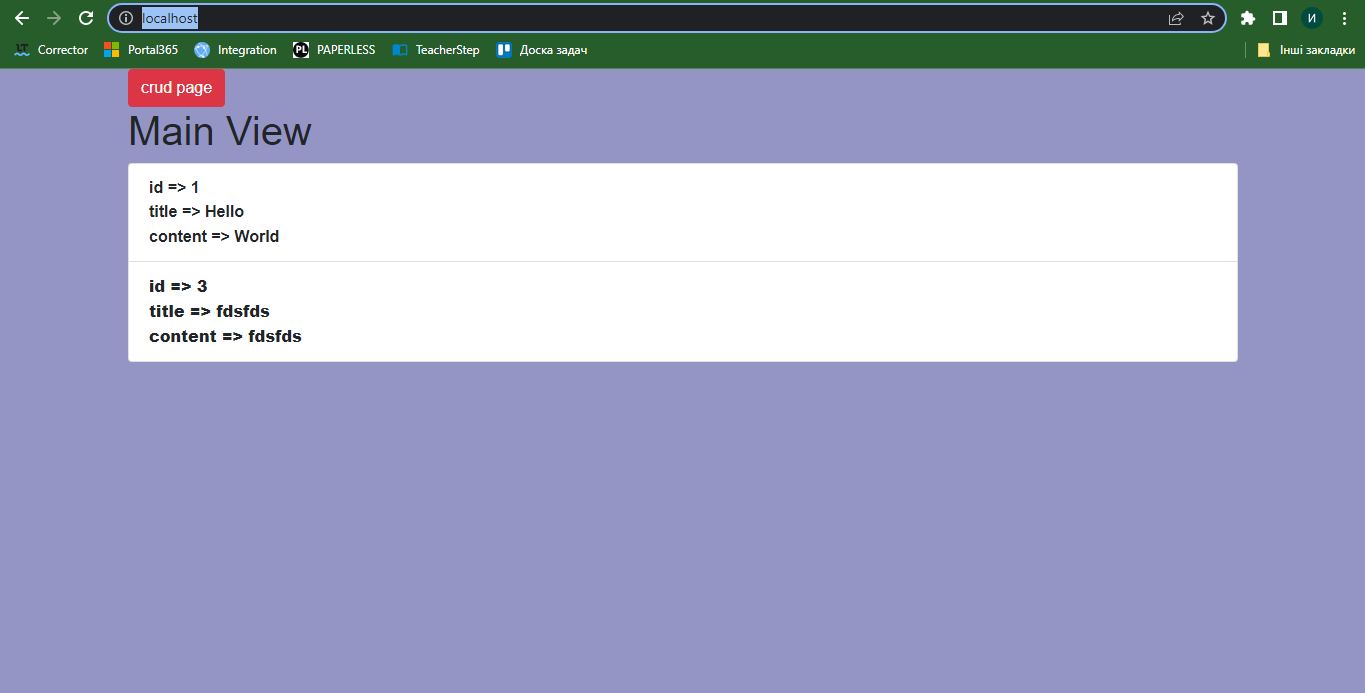
    font-family: sans-serif;

    background-color: rgb(149, 149, 197);

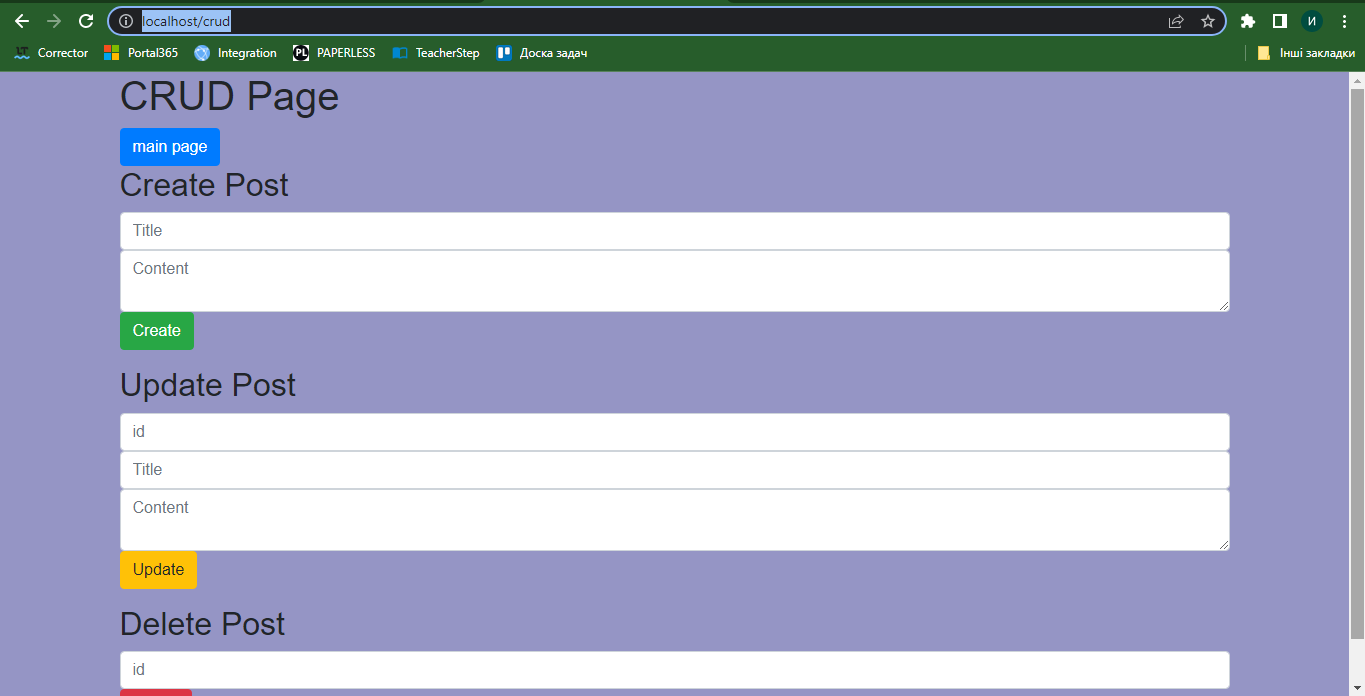
}

Result:

<http://localhost/>



<http://localhost/crud>



1. Laravel:



Описание: Современный и мощный фреймворк с широким спектром функций, включая элегантный синтаксис, ORM, миграции, аутентификацию и многое другое.

Преимущества: Легкость в изучении, обширное сообщество, встроенные инструменты для быстрой разработки.

Сравнение: Считается одним из самых популярных PHP-фреймворков, подходит для разработки разнообразных приложений.

1. Symfony:

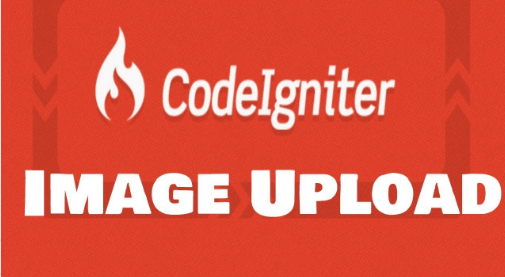


Описание: Высокопроизводительный фреймворк, использующий множество компонентов для гибкости и модульности.

Преимущества: Подходит как для маленьких, так и для крупных проектов, качественная документация, множество готовых компонентов.

Сравнение: Symfony предоставляет больше контроля разработчику, но может быть сложным для начинающих.

1. CodeIgniter:



Описание: Легковесный и быстрый фреймворк с минимальными требованиями к серверу.

Преимущества: Простота использования, быстрый запуск, хорошо подходит для небольших и средних проектов.

Сравнение: Предоставляет базовый набор инструментов, но может быть ограничен для более сложных задач.

1. Yii:



Описание: Высокопроизводительный фреймворк с уклоном на быструю разработку.

Преимущества: Быстрая разработка благодаря готовым компонентам, хорошо подходит для веб-приложений с высокой нагрузкой.

Сравнение: Yii предоставляет множество встроенных решений, но может иметь небольшой порог входа для новичков.

1. Phalcon:



Описание: Высокопроизводительный фреймворк, написанный на C и предоставляемый как расширение PHP.

Преимущества: Очень быстрый, использует преимущества низкоуровневой оптимизации.

Сравнение: Подходит для проектов, требующих высокой производительности, но требует настройки.

1. Slim:



Описание: Микрофреймворк с минимальным набором инструментов для быстрой разработки маленьких приложений и API.

Преимущества: Легковесность, простота использования, хорошо подходит для RESTful API.

Сравнение: Отличный выбор для небольших проектов, но может быть ограничен для сложных приложений.

1. Laminas (ранее Zend Framework):



Описание: Компонентно-ориентированный фреймворк с различными модулями для создания масштабируемых приложений.

Преимущества: Высокая гибкость, модульность, большое сообщество, поддержка различных стандартов.

Сравнение: Подходит для профессиональных разработчиков и проектов с большими требованиями.

1. CakePHP:



Описание: Фреймворк с акцентом на быструю разработку и с поддержкой конвенции над конфигурацией.

Преимущества: Быстрая разработка благодаря автоматическому генерированию кода, встроенные инструменты.

Сравнение: Подходит для быстрого создания веб-приложений с меньшей сложностью.

1. Flight:



Описание: Ультра-легкий микрофреймворк для создания минимальных веб-приложений.

Преимущества: Простота, минимализм, подходит для создания самых маленьких приложений.

Сравнение: Идеальный выбор для простых задач, но может быть ограничен для более крупных проектов.

1. Aura:



Описание: Микрофреймворк с фокусом на независимые компоненты, которые можно использовать по отдельности.

Преимущества: Гибкость, модульность, возможность подключать только необходимые компоненты.

Сравнение: Подходит для разработки приложений, где требуется гранулярный контроль над компонентами.

Выбор фреймворка зависит от требований проекта, уровня опыта разработчика и предпочтений. Каждый из этих фреймворков имеет свои преимущества и недостатки, и выбор должен основываться на конкретных потребностях.

Материалы к уроку:

1. <https://phpabstract.ru/phpmvc>
2. <https://anton-pribora.ru/articles/php/mvc/>
3. <https://maxsite.org/page/model-view-controller-php>

Домашнее задание:

1. Добавьте к вашему сайту регистрацию и авторизацию, нужные controllers, views, models
2. Добавьте в таблицу пользователей роли
3. Если роль позволяет, то пользователь сможет попасть на страницу crud и ее модерировать

