

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий

Кафедра «Инфокогнитивные технологии»

Направление подготовки: «Информатика и вычислительная техника»

КУРСОВОЙ ПРОЕКТ на тему:

«Разработка веб фреймворка»

Выполнил:

Студент

Цейгалов Андрей Владимирович

Группа

243-321

Дата

18 ноября 2025

1. Введение

Данный курсовой проект посвящён разработке и практической реализации мини-веб-фреймворка на языке **PHP**, использующего архитектурный шаблон **MVC (Model–View–Controller)**.

Целью работы является создание понятного и расширяемого каркаса веб-приложения, который обеспечивает:

- **маршрутизацию запросов** (преобразование URL в вызовы контроллеров);
- **работу с базой данных** через слой моделей и сервис доступа к данным;
- **шаблонизацию представлений** с использованием отдельных PHP-шаблонов;
- **разделение ответственности** между слоями приложения.

Реализуемое приложение представляет собой простой блог: есть список статей, просмотр отдельной статьи, добавление, редактирование и удаление статей и комментариев (для администратора), а также несколько демонстрационных страниц.

2. Цель и задачи проекта

Цель проекта – спроектировать и реализовать учебный веб-фреймворк на PHP, демонстрирующий ключевые принципы построения серверных приложений.

Для достижения цели решаются следующие **задачи**:

- **Спроектировать архитектуру** приложения на основе MVC.
- **Реализовать механизм автозагрузки** классов и центральную точку входа.
- **Реализовать маршрутизацию** HTTP-запросов на основе регулярных выражений.
- **Организовать доступ к базе данных** с использованием шаблона **Singleton** и собственного слоя Active Record.
- **Разработать подсистему представлений (View)** с поддержкой шаблонов и передачи переменных.
- **Реализовать модуль статей и комментариев** (создание, редактирование, удаление, отображение).
- **Добавить базовую модель пользователя** и примитивную проверку прав администратора.

3. Общая архитектура проекта

Структура проекта:

- **index.php** – центральная точка входа в приложение, инициализирует автозагрузку и маршрутизацию.
- **src/settings.php** – файл настроек (в данном проекте – параметры подключения к БД MySQL).
- **src/routes.php** – таблица маршрутов (соответствие URL-шаблонов контроллерам и методам).
- **src/MyProject/Controllers** – контроллеры:
 - MainController.php – главные и демонстрационные страницы;
 - ArticlesController.php – управление статьями;
 - CommentsController.php – управление комментариями.
- **src/MyProject/Models** – модели и базовый класс Active Record:
 - ActiveRecordEntity.php – общий класс для всех сущностей БД;
 - Articles/Article.php – модель статьи;
 - Comments/Comment.php – модель комментария;
 - Users/User.php – модель пользователя.
- **src/MyProject/Services/Db.php** – сервис доступа к базе данных, инкапсулирующий работу с PDO.
- **src/MyProject/View/View.php** – класс, отвечающий за рендеринг HTML-шаблонов.
- **templates** – PHP-шаблоны представлений:
 - общие шаблоны: header.php, footer.php;
 - шаблоны основных страниц: main/main.php, main/hello.php, main/bye.php, main/about.php;
 - шаблоны статей: articles/view.php, articles/edit.php;
 - шаблоны комментариев: comments/edit.php;
 - страницы ошибок: errors/403.php, errors/404.php.
- **style.css** – таблица стилей для оформления HTML-страниц.

Архитектура построена таким образом, чтобы каждый слой (Controller, Model, View) был слабо связан с другими и мог развиваться независимо.

4. Описание ключевых компонентов

4.1. Точка входа и автозагрузка (index.php)

Файл index.php выполняет следующие функции:

- регистрирует **автозагрузчик классов**, преобразующий полное имя класса с пространством имён (например, MyProject\Controllers\MainController) в путь к файлу внутри каталога src;
- получает текущий маршрут (параметр route либо часть REQUEST_URI);

- подставляет этот маршрут в таблицу маршрутов из `src/routes.php`;
- ищет совпадение по регулярным выражениям и при успешном совпадении создаёт экземпляр нужного контроллера и вызывает соответствующий метод, передавая параметры маршрута.

Таким образом реализуется простая, но гибкая система маршрутизации.

4.2. Маршрутизация (*src/routes.php*)

Файл `src/routes.php` возвращает ассоциативный массив вида:

- **ключ** – регулярное выражение для сопоставления с URL;
- **значение** – массив [ИмяКлассаКонтроллера, 'метод'].

Примеры маршрутов:

- `~^$~` → `MainController::main` – главная страница со списком статей;
- `~^articles/(\d+)$~` → `ArticlesController::view` – просмотр статьи по ID;
- `~^articles/(\d+)/comments$~` → `CommentsController::add` – добавление комментария;
- `~^article/(\d+)/edit$~` → `ArticlesController::edit` – редактирование статьи;
- `~^hello/(.*)$~, ~^bye/(.*)$~, ~^about-me$~` – демонстрационные страницы.

Использование регулярок позволяет легко добавлять новые маршруты и параметры.

4.3. Сервис работы с БД (*Db.php*)

Класс `MyProject\Services\Db`:

- реализует **паттерн Singleton** (единственный экземпляр на всё приложение);
- внутри конструктора создаёт подключение к **MySQL** через PDO, используя параметры из `src/settings.php`;
- задаёт кодировку UTF8;
- предоставляет метод `query(string $sql, array $params = [], ?string $className = null): ?array`, который:
 - подготавливает и выполняет SQL-запрос;
 - по умолчанию возвращает массив ассоциативных массивов;
 - при передаче имени класса создаёт объекты указанного класса и заполняет их данными из БД;
- содержит метод `getLastInsertId()`, возвращающий идентификатор последней вставленной записи.

4.4. Слой моделей и Active Record (*ActiveRecordEntity.php*)

Абстрактный класс `MyProject\Models\ActiveRecordEntity` реализует базовый функционал шаблона **Active Record**:

- хранит первичный ключ `id` и геттер `getId()`;
- определяет магический метод `__set()` для преобразования имён полей в `camelCase`-свойства;
- реализует статические методы:
 - `findAll()` – получение всех записей таблицы;
 - `getById(int $id)` – получение одной записи по идентификатору;
- реализует методы экземпляра:
 - `save()` – определяет, нужно ли делать `INSERT` или `UPDATE`;
 - `delete()` – удаление записи из БД;
- содержит приватные методы:
 - `mapPropertiesToDbFormat()` – преобразование свойств объекта в массив `имя_поля => значение`;
 - `camelCaseToUnderscore()` – преобразование `camelCase` в `snake_case`;
 - `update()` и `insert()` – построение SQL-запросов на обновление и вставку.

Каждая конкретная модель (статья, комментарий, пользователь) должна унаследоваться от `ActiveRecordEntity` и реализовать метод `protected static function getTableName(): string`, возвращающий имя таблицы.

4.5. Контроллеры

- **MainController**
 - `main()` – вывод главной страницы со списком всех статей (`Article::findAll()`), а также признак, является ли текущий пользователь администратором.
 - `sayHello($name)`, `sayBye($name)` – демонстрационные страницы с приветствием/прощанием.
 - `aboutMe()` – статическая страница «Обо мне».
- **ArticlesController**
 - `view(int $articleId)` – просмотр статьи, отображение автора, текста, комментариев. В случае отсутствия статьи выводит шаблон ошибки **404**.
 - `add()` – создание новой статьи (доступно администратору). Обработывает POST-данные, валидирует поля и сохраняет статью в БД.
 - `edit(int $articleId)` – редактирование существующей статьи, доступно только администратору.
 - `delete(int $articleId)` – удаление статьи и связанных с ней комментариев, затем перенаправление на главную.
- **CommentsController**
 - `add(int $articleId)` – добавление комментария к статье (в проекте привязано к пользователю с `id = 1` как к автору).

- `edit(int $commentId)` – редактирование текста комментария.
- `delete(int $commentId)` – удаление комментария (только для администратора).

Во всех контроллерах для работы с представлениями используется класс `View`, для получения текущего пользователя – модель `User` (в учебной реализации текущий пользователь жёстко задан как пользователь с `id = 1`).

4.6. Подсистема представлений (*View.php* и папка *templates*)

Класс `MyProject\View\View`:

- получает путь к каталогу шаблонов при создании;
- метод `renderHtml(string $templateName, array $vars = [], int $code = 200)`:
 - устанавливает HTTP-код ответа;
 - извлекает переменные из массива `$vars`;
 - подключает нужный шаблон из директории `templates`;
 - использует буферизацию вывода, чтобы собрать HTML и вывести его целиком.

Шаблоны в папке `templates` реализуют простую систему представлений:

- общий **заголовок** (`header.php`) и **подвал** (`footer.php`);
- шаблоны для страниц списка статей, просмотра статьи, форм редактирования;
- шаблоны для страниц ошибок 403/404.

5. База данных

В файле `src/settings.php` задаются параметры подключения к MySQL:

- **host:** localhost;
- **dbname:** bd-blog;
- **user:** root;
- **password:** (пустой).

Предполагается наличие следующих таблиц (логика моделей указывает на типичную структуру блога):

- **users** – пользователи (в том числе администратор);
- **articles** – статьи (название, текст, автор, дата создания);
- **comments** – комментарии к статьям (автор, статья, текст, дата создания).

Связи:

- один пользователь может иметь несколько статей и комментариев;
- одна статья может иметь много комментариев.

6. Установка и запуск

- **Требования к окружению:**
 - установленный веб-сервер (Apache, Nginx и т.п.);
 - PHP версии **7.4+** (рекомендуется 8.x);
 - модуль PDO для работы с MySQL;
 - установленная и настроенная СУБД **MySQL**.
- **Шаги по развёртыванию:**
 1. Скопировать проект в корневую директорию веб-сервера (например, в каталог myproject).
 2. Создать в MySQL базу данных bd-blog и таблицы users, articles, comments согласно структуре моделей.
 3. При необходимости отредактировать параметры подключения в файле src/settings.php.
 4. Настроить виртуальный хост и .htaccess (если используется Apache), чтобы все запросы направлялись на index.php.
 5. Открыть в браузере адрес вида: `http://localhost/myproject/`.

7. Пользовательские сценарии

- **Просмотр главной страницы.**

Пользователь заходит на /myproject/ и видит список всех статей. Администратор дополнительно видит ссылку «Создать статью».
- **Просмотр статьи и комментариев.**

При переходе по адресу /myproject/articles/{id} отображается полная статья, информация об авторе и список комментариев. При наличии прав администратора отображаются ссылки на редактирование и удаление.
- **Добавление статьи (администратор).**

Администратор открывает /myproject/articles/add, заполняет форму и сохраняет статью. После сохранения происходит перенаправление на страницу созданной статьи.
- **Редактирование и удаление статей/комментариев.**

Администратор может редактировать и удалять статьи и комментарии через соответствующие ссылки в интерфейсе.
- **Демонстрационные страницы.**

Страницы /myproject/hello/{name}, /myproject/bye/{name}, /myproject/about-me демонстрируют работу маршрутизации и передачи параметров в шаблоны.

8. Выводы

В ходе выполнения курсового проекта был реализован учебный веб-фреймворк на РНР, отражающий основные принципы построения серверной архитектуры:

- реализованы автозагрузка классов, маршрутизация, слой доступа к данным и подсистема представлений;
- использован шаблон **MVC**, обеспечивающий разделение логики, данных и представления;
- реализован базовый **Active Record** для работы с сущностями БД;
- разработан функционал блога с поддержкой статей и комментариев, а также простая система разграничения прав администратора.

Полученный результат может служить основой для дальнейшего развития: добавления полноценной аутентификации, системы ролей, REST-API, расширенной системы шаблонов и других возможностей, характерных для промышленных веб-фреймворков.