Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Московский политехнический университет»

Образовательная программа «Веб-технологии»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Отчет по курсовой работе

по дисциплине «Серверная веб-разработка»

Тема: «Backend-фреймворк на PHP»

**Выполнил:**

Студент группы 241-321

Иванилов А. Т.

подпись, дата

**Принял:**

Старший преподаватель кафедры ИКТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Володина О.В.

подпись, дата

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc196383607)

[Глава 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc196383608)

[1.1 Анализ предметной области 4](#_Toc196383609)

1.2 Выбор архитектурного подхода…………………………………………

[1.3 Анализ функциональности проекта 4](#_Toc196383610)

[Глава 2. ПРАКТИЧЕСКАЯ ЧАСТЬ 5](#_Toc196383612)

2.1 Постановка задачи………………………………………………………….

[2.2 Структура базы данных 5](#_Toc196383614)

[2.3 Реализация модели пользователя 5](#_Toc196383615)

[2.4. Реализация создания и редактирования статей 5](#_Toc196383616)

[2.5 Реализация загрузки изображений и выбора автора статьи 6](#_Toc196383617)

[ЗАКЛЮЧЕНИЕ 7](#_Toc196383618)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 8](#_Toc196383619)

# ВВЕДЕНИЕ

Цель проекта — разработать веб-приложение на языке PHP, реализующее CRUD-функциональность (создание, чтение, обновление, удаление) с использованием архитектурного подхода MVC.

Тематика проекта — система управления статьями, в которой зарегистрированные пользователи могут создавать, редактировать и удалять свои публикации, добавлять изображения и указывать автора статьи.

В процессе разработки особое внимание было уделено разделению логики на модель, представление и контроллеры, что способствует масштабируемости и поддерживаемости кода.

Цель проекта: разработать веб-сайт на тему «Онлайн Записная книжка»

Задачи проекта:

1. Проанализировать предметную область
2. Спроектировать структуру базы данных
3. Спроектировать и разработать административную часть
4. Спроектировать и разработать главной страницы
5. Спроектировать и разработать детальной страницы
6. Спроектировать и разработать дополнительного функционала

Ссылка на репозиторий (проверить доступность в режиме инкогнито)

# Глава 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Анализ предметной области

## В современном цифровом пространстве большое количество информации публикуется и управляется при помощи веб-приложений. Одним из распространённых типов таких приложений являются системы управления контентом (CMS), позволяющие пользователям без глубоких технических знаний создавать, редактировать и публиковать материалы в интернете.

## Ключевым функциональным ядром таких систем является реализация CRUD-операций (от англ. Create, Read, Update, Delete) — создание, просмотр, редактирование и удаление данных. Данные в рамках подобных систем чаще всего представлены в виде статей, блогов, комментариев, заметок и т.п.

## В выбранной предметной области пользователь играет активную роль: он не только потребляет информацию, но и создаёт её. Поэтому важно обеспечить удобную авторизацию, разграничение прав доступа, а также механизмы работы с мультимедийными файлами — такими как изображения, сопровождающие статьи. Возможность прикрепления изображения к статье делает контент более визуально привлекательным и информативным.1.4. Анализ функциональности проекта.

1.2 Выбор архитектурного подхода

Архитектурно такие системы часто строятся на принципе MVC (Model-View-Controller), что позволяет разделить обработку данных, логику отображения и управление пользователями. Это повышает масштабируемость, упрощает поддержку кода и позволяет в будущем расширять функциональность проекта.

Таким образом, предметная область проекта охватывает:

* Веб-разработку на стороне сервера (PHP);
* Базы данных и хранение информации (MySQL);
* Принципы аутентификации пользователей;
* Обработку файлов и загрузку изображений;
* Организацию кода по архитектуре MVC;
* Стандарты взаимодействия с пользователем через браузер.

1.3 Анализ функциональности проекта

Проект представляет собой веб-приложение с возможностью управления пользователями и статьями, реализованное на языке PHP с использованием архитектурного шаблона MVC. В качестве интерфейса используется самостоятельно разработанная административная панель, которая предоставляет доступ ко всем возможностям системы.

Классическая авторизация и регистрация пользователей в проекте не реализованы. Все действия выполняются в рамках единого административного доступа, что упрощает логику приложения и делает его более удобным для локального или внутреннего использования (например, как внутренняя CMS для одного редактора или небольшой команды).

Основной реализованный функционал проекта:

* Создание пользователей — через админскую панель можно добавлять новых пользователей с заданными именами. Каждый пользователь отображается в системе как автор статей.
* Управление статьями (CRUD) — для каждой статьи можно задать заголовок, текст, прикрепить изображение и выбрать пользователя-автора.
* Вывод списка статей — реализован просмотр всех добавленных статей в виде списка с краткой информацией.
* Просмотр полной статьи — при переходе к статье отображается полный текст, изображение (если прикреплено), а также имя автора.
* Редактирование статьи — можно изменить заголовок, содержимое, изображение и автора статьи.
* Удаление статьи — реализована функция удаления статьи из базы данных.
* Загрузка изображений — статьи могут содержать прикреплённые изображения (форматы JPG, PNG). Картинки загружаются на сервер и хранятся в отдельной директории.
* MVC-архитектура — приложение построено по модели MVC, что обеспечивает структурированность кода, разделение логики и удобство масштабирования.

Особенности реализации:

* Все данные (пользователи, статьи) хранятся в базе данных.
* Вся работа с проектом осуществляется через единый интерфейс без логина и пароля.
* Пользователи добавляются не через регистрацию, а вручную через административный интерфейс.
* Используется простой роутинг на основе контроллеров и параметров URL.

Такой подход делает проект удобным в образовательных целях и демонстрирует умение студента разрабатывать функциональные веб-приложения с нуля, без использования готовых CMS или фреймворков.

# Глава 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1. Постановка задачи

В рамках данного курсового проекта необходимо реализовать веб-приложение, позволяющее администрировать статьи и пользователей. Основное внимание уделяется созданию простой CMS-системы, реализованной с нуля на языке PHP с использованием архитектуры MVC.

Цель проекта:

Разработка функционального веб-приложения для создания, редактирования и управления статьями и пользователями через административную панель.

Задачи проекта:

1. Спроектировать и создать структуру базы данных.
2. Реализовать архитектурный шаблон MVC.
3. Разработать интерфейс для:

3.1 добавления и редактирования пользователей;

3.2 создания и редактирования статей;

3.3 привязки статьи к конкретному пользователю;

1. прикрепления изображений к статьям.
2. Обеспечить корректное хранение и отображение данных.
3. Обеспечить удобную и логичную навигацию внутри административной части сайта.

Проект не предусматривает пользовательскую авторизацию — все действия выполняются через административный интерфейс.

## 2.2 Структура базы данных

База данных проекта реализована при помощи двух таблиц : users и articles. В таблице users хранится вся необходимая информация о пользователях приложения : id пользователя (для реализации привязки автора статьи к самой статье) , username , почта и пароль (для последующего развития проекта путем добавления функций авторизации и регистрации), и дата создания пользователя. В свою очередь в таблице articles хранится важная информация для самих статей : id статьи , название статьи, содержимое, id автора (связь с первой таблицей), дата написания статьи и путь до картинки (если в статье присутствует изображение). Визуальное представление базы данных можно увидеть на рисунке 1.

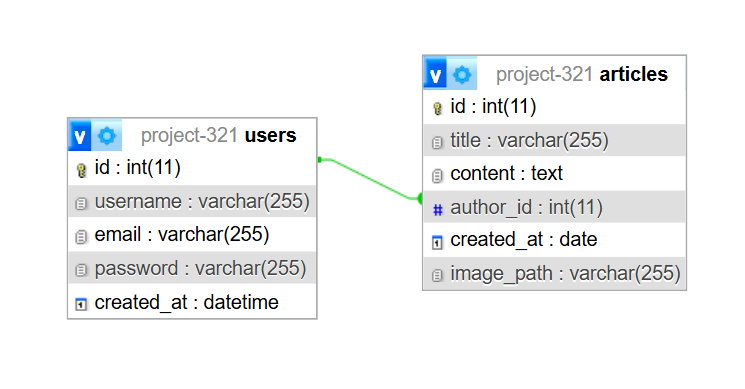


Рис. 1 – Визуальное представление базы данных

## 2.3. Реализация модели пользователя

Логика проекта реализована через файл ActiveRecordEntity и отдельные файлы моделей (Models).

Из файла Models/User.php

class User extends ActiveRecordEntity

{

    protected $username;

    protected $email;

    protected $isConfirmed;

    protected $role;

    protected $passwordHash;

    protected $authToken;

    protected $createdAt;

    protected static function getTableName(){

        return 'users';

    }

    public function setUsername(string $username){

        $this->username = $username;

    }

    public function setEmail(string $email){

        $this->email = $email;

    }

    public function getUsername(): string

    {

        return $this->username;

    }

    public function getEmail()

    {

        return $this->email;

    }

}

Для реализации модели пользователя в проекте применён паттерн Active Record. Все сущности проекта (в том числе пользователь) наследуют абстрактный класс ActiveRecordEntity, который инкапсулирует основную логику взаимодействия с базой данных: загрузка, сохранение, обновление и удаление данных. Это позволяет сократить дублирование кода и централизовать работу с данными.

Из ActiveRecordEntity.php

public function save(){

        $propertisDB = $this->MappedPropertiesToDB();

        if($this->id) $this->update($propertisDB);

        else $this->insert($propertisDB);

    }

    protected function update($propertisDB){

        $db = DB::getInstance();

        $columns2Params = [];

        $params2Values = [];

        foreach($propertisDB as $key=>$value){

            $param = ':'.$key;

            $column = '`'.$key.'`';

            $columns2Params[] = $column.'='.$param;

            $params2Values[$param] = $value;

        }

        $sql = 'UPDATE `'.static::getTableName().'` SET '.implode(',', $columns2Params).' WHERE `id`=:id';

        $db->query($sql, $params2Values, static::class);

    }

    protected function insert($propertisDB){

        $propertisDB = array\_filter($propertisDB);

        $db = Db::getInstance();

        $columns = [];

        $params = [];

        $params2Values = [];

        foreach($propertisDB as $key=>$value){

            $columns[] = '`'.$key.'`';

            $param = ':'.$key;

            $params[] = $param;

            $params2Values[$param] = $value;

        }

        $sql = 'INSERT INTO `'.static::getTableName().'` ('.implode(',', $columns).') VALUES ('.implode(',', $params).')';

        $db->query($sql, $params2Values, static::class);

    }

На базе абстрактного класса ActiveRecordEntity реализованы универсальные методы save(), insert() и update(), обеспечивающие автоматическое сохранение сущности в базе данных. Метод save() определяет, нужно ли выполнить вставку или обновление записи, а методы insert() и update() формируют соответствующий SQL-запрос. Благодаря этой архитектуре, каждая модель (включая модель пользователя) может использовать эту логику без дублирования кода.

## 2.4. Реализация создания и редактирования статей

В рамках данного проекта реализован функционал создания, редактирования, отображения и удаления статей. Для этого использован шаблон проектирования MVC (Model-View-Controller), что позволяет разделить бизнес-логику, представление и управление.

Контролер

Контроллер отвечает за обработку пользовательских запросов и взаимодействие с моделью и представлением. В частности, методы create(), store(), edit(), update() и delete() реализуют основные операции CRUD для статей.

public function store(){

$article = new Article;

$article->title = $\_POST['title'];

$article->content = $\_POST['content'];

$username = trim($\_POST['author']);

$article->createdAt = $\_POST['createdAt'];

// Получение пользователя-автора по имени

if (!empty($username)) {

$user = User::getByUsername($username);

}

// Если пользователь не найден, назначается пользователь по умолчанию "Гость"

if (empty($user)) {

$user = User::getByUsername('Гость');

if ($user === null) {

die('Ошибка: пользователь по умолчанию (Гость) не найден в базе данных');

}

}

$article->setAuthor($user);

// Обработка загрузки изображения

if (isset($\_FILES['image']) && $\_FILES['image']['error'] === UPLOAD\_ERR\_OK) {

// Проверка и загрузка файла

// ...

$article->imagePath = '/images/' . $newFileName;

}

// Сохранение статьи в базе данных

$article->save();

// Перенаправление на главную страницу

return header('Location:http://localhost/241-321-Ivanilov-Alexei/Project/www/index.php');

}

Данный метод принимает данные из формы, связывает статью с существующим пользователем, обрабатывает загруженное изображение, после чего сохраняет статью в базу данных.

Модель

Модель наследует абстрактный класс ActiveRecordEntity, реализующий логику работы с базой данных (CRUD). В модели определены свойства статьи: заголовок, содержание, ссылка на изображение, идентификатор автора и дата создания.

public function setAuthor(User $author): void

{

$this->authorId = $author->getId();

}

public function getAuthor(): ?User

{

if ($this->authorId === null) {

return null;

}

return User::getById($this->authorId);

}

public function getTitle()

{

return $this->title;

}

public function getContent()

{

return $this->content;

}

public function getCreatedAt()

{

return $this->createdAt;

}

public function getImagePath(): ?string

{

return $this->imagePath;

}

Метод setAuthor() связывает статью с объектом пользователя, что позволяет гибко управлять авторством. Метод getAuthor() возвращает объект пользователя, написавшего статью, что облегчает вывод информации об авторе.

Такой подход позволяет централизованно и удобно управлять статьями, гарантируя корректность связей между статьей и её автором, а также поддерживает расширение функционала (например, добавление изображений).

## 2.5 Реализация загрузки изображений и выбора автора статьи

**Загрузка изображений**

Загрузка изображений реализована в методе store() контроллера ArticleController. При отправке формы с новой статьёй сервер проверяет, был ли загружен файл и соответствует ли он требованиям по типу (поддерживаются форматы JPEG, PNG, GIF)

if (isset($\_FILES['image']) && $\_FILES['image']['error'] === UPLOAD\_ERR\_OK) {

$fileTmpPath = $\_FILES['image']['tmp\_name'];

$fileName = $\_FILES['image']['name'];

$fileType = $\_FILES['image']['type'];

$allowedTypes = ['image/jpeg', 'image/png', 'image/gif'];

if (!in\_array($fileType, $allowedTypes)) {

die('Ошибка: можно загружать только изображения JPG, PNG или GIF');

}

$fileNameCmps = explode(".", $fileName);

$fileExtension = strtolower(end($fileNameCmps));

$newFileName = md5(time() . $fileName) . '.' . $fileExtension;

$uploadFileDir = \_\_DIR\_\_ . '/../../www/images/';

$destPath = $uploadFileDir . $newFileName;

if (move\_uploaded\_file($fileTmpPath, $destPath)) {

$article->imagePath = '/images/' . $newFileName;

} else {

die('Ошибка при загрузке файла');

}

}

Этот код обеспечивает проверку типа файла, уникальное именование загружаемого изображения и перемещение его в директорию для хранения изображений. Путь к файлу сохраняется в свойстве модели статьи $imagePath для последующего отображения.

Выбор автора статьи

# Выбор автора осуществляется путём передачи имени пользователя через форму. В контроллере происходит попытка получить объект пользователя из базы по введённому имени с помощью метода User::getByUsername().

$username = trim($\_POST['author']);

if (!empty($username)) {

$user = User::getByUsername($username);

}

if (empty($user)) {

$user = User::getByUsername('Гость');

if ($user === null) {

die('Ошибка: пользователь по умолчанию (Гость) не найден в базе данных');

}

}

$article->setAuthor($user);

Если пользователь с указанным именем не найден, то автоматически назначается пользователь с именем «Гость», что предотвращает ошибки и гарантирует наличие автора у каждой статьи. Метод setAuthor() модели Article устанавливает связь статьи с выбранным пользователем.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана функциональная веб-система с использованием архитектурного паттерна MVC, обеспечивающая создание, чтение, обновление и удаление записей пользователей и статей. Особое внимание уделялось реализации механизма выбора автора статьи и загрузки изображений, что расширяет возможности взаимодействия с контентом и улучшает пользовательский опыт.

Реализованная система демонстрирует основные принципы объектно-ориентированного программирования и работы с базой данных через собственную ORM-реализацию на основе паттерна Active Record. Использование разделения ответственности между слоями модели, контроллера и представления позволило достичь чистой и удобной архитектуры, упрощающей поддержку и дальнейшее развитие проекта.

В будущем проект может быть дополнен функционалом аутентификации, разграничения прав пользователей, улучшенными средствами валидации и загрузки файлов, а также расширен за счёт дополнительных сущностей и возможностей. Работа над данным проектом позволила закрепить знания в области PHP-разработки, MVC, работы с базами данных и веб-технологиями в целом.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Фраерман, Д. В. PHP 8: основы программирования. — Москва: Питер, 2022. — 560 с.
2. Шилдт, Г. PHP. Полное руководство. — Санкт-Петербург: БХВ-Петербург, 2021. — 720 с.
3. МакБейн, М. Изучаем PHP и MySQL. — Москва: Вильямс, 2020. — 480 с.
4. Мартин, Р. С. Чистый код. Создание, анализ и рефакторинг. — Москва: ДМК Пресс, 2018. — 464 с.
5. Пауэлл, М. PHP и MySQL для начинающих. — Санкт-Петербург: Питер, 2019. — 350 с.
6. Бак, Д. Современный PHP: руководство по эффективной разработке. — Москва: ЛитРес, 2021. — 400 с.

**Требования к оформлению текстов**

Общие требования:

1) Параметры страницы: все поля (слева, справа, сверху, снизу) – 2 см. Для всего текста следует использовать шрифт Times New Roman, размер 14 пт, межстрочный интервал − полуторный, выравнивание по ширине. Отступ абзаца (красная строка) – 1,25 см, все отступы (слева, справа, сверху, снизу) – 0. Не допускаются: подчёркивание, выделение цветом, орфографические и пунктуационные ошибки.

2) Таблицы создаются стандартными средствами редактора Microsoft Word. Не допускается вставка таблиц из других программ.

3) В виде рисунков оформляются фотографии, схемы, графики, диаграммы и др.

4) Все рисунки и таблицы должны иметь названия и отдельную нумерацию, а также ссылки на них в тексте. Подрисуночные и надтабличные подписи приводятся в тексте в месте расположения рисунка или таблицы.