Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Система управления содержимым веб-сайтов

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР _____ Д. И. Украинцев

(подпись, дата) (инициалы, фамилия)

Группа ПО-01б

Руководитель ВКР _____ В. В. Серебровский

(подпись, дата) (инициалы, фамилия)

Нормоконтроль _____ А. А. Чаплыгин

(подпись, дата) (инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой _____ А. В. Малышев

(подпись, дата) (инициалы, фамилия)

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Украинцева Д.И., шифр 20-06-0408, группа ПО-01б

1. Тема «Система управления содержимым веб-сайтов » утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1620-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) проанализировать оссобенности функционирования систем управления содержимым веб-сайта;

2) разработать концептуальную модель системы управления содержимым веб-сайта;

3) спроектировать программную программную систему управления содержимым веб-сайта;

4) сконструировать и протестировать программную систему управления содержимым веб-сайта.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: содержимое страниц и постов, текст; медиафайлы; кактегории и теги постов; методанные страниц; регистрационная информация пользователей.

2) Выходными данными для программной системы являются: сформированные веб-страницы сайта на основе шаблонов и содержимого страниц и постов.

4. Содержание работы (по разделам):

4.1. Введение

4.1. Анализ предметной области

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ

Лист 2. Цели и задачи разработки

Лист 3. Диаграмма прецедентов

Лист 4. Реляционная модель данных

Лист 5. Диаграмма компонентов

Лист 6. Диаграмма классов (часть 1)

Лист 7. Диаграмма классов (часть 2)

Лист 8. Заключение

Руководитель ВКР		В. В. Серебровский
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению		Д. И. Украинцев
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 105 страницам. Работа содержит 23 иллюстрации, 30 таблиц, 20 библиографических источников и 8 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: веб-сайт, веб-страница, система, CMS, веб-форма, Apache, веб-сервер, база данных, класс, компонент, модуль, сущность, модель, контроллер, представление, метод, редактор контента, администратор, пользователь.

Объектом разработки является программно-информационная система для управления содержимым веб-сайта.

Целью выпускной квалификационной работы является разработка системы управления содержимым веб-сайта, предназначенной для совместного создания и управления веб-сайтами.

В процессе создания системы были выделены основные сущности системы, разработана база данных. Была использована методология ООП, спроектированы классы и реализованы методы модулей, обеспечивающие работу с сущностями предметной области и корректную работу системы.

При разработке системы был использован веб-сервер Apache и языки программирования PHP, Javascript, язык разметы HTML, каскаданые таблицы стилей CSS.

ABSTRACT

The volume of work is 105 pages. The work contains 23 illustrations, 30 tables, 20 bibliographic sources and 8 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. Source code fragments are presented in Appendix B.

List of keywords: website, web page, system, CMS, web form, Apache, web server, database , class, component, module, entity, model, controller, view, method, content editor, administrator, user.

The object of development is a software information system for managing website content.

The purpose of the final qualifying work is to develop a website content management system intended for joint creation and management of websites.

In the process of creating the system, the main entities of the system were identified and a database was developed. The OOP methodology was used, classes were designed and module methods were implemented to ensure work with domain entities and correct operation of the system.

When developing the system, the Apache web server and the programming languages PHP, Javascript, HTML markup language, and CSS cascading style sheets were used.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных.

ИС – информационная система.

ИТ – информационные технологии.

ПО – программное обеспечение.

РП – рабочий проект.

СУБД – система управления базами данных.

ТЗ – техническое задание.

ТП – технический проект.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

ООП - (Объектно-ориентированное программирование) – методология программирования, основанная на представлении программы в виде совокупности объектов.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных.

HTML (HyperText Markup Language) – язык гипертекстовой разметки документов для просмотра веб-страниц в браузере.

SQL (Structured Query Language) – декларативный язык программирования для структурных. запросов к базе данных.

CMS (Content Management System) – система управления содержимым.

MVC (Model-View-Controller) – это шаблон (паттерн) программирования, разделяющий архитектуру приложения на три отдельных компонента: модель (Model), представление (View), контроллер (Controller).

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования.

WYSIWYG (What You See Is What You Get, «что видишь, то и получишь») – свойство прикладных программ или веб-интерфейсов, в которых содержание отображается в процессе редактирования и выглядит максималь-

но близко похожим на конечную продукцию, которая может быть печатным документом, веб-страницей или презентацией.

ВВЕДЕНИЕ

Любой веб-сайт состоит из набора страниц, а различия заключаются лишь в том, как они организованы. Существует два вида организации веб-сайта – статический и динамический. В первом случае специалисты, отвечающие за создание и поддержку сайта пишут в HTML-форме каждую в отдельности страницу, включая ее оформление и контент. Во втором – в основе любой веб-страницы лежит шаблон, определяющий расположение в окне веб-браузера всех компонентов страницы, и вставка конкретной информации производится с использованием стандартных средств, не требующих от участника процесса знания языка HTML и достаточно сложных для неспециалиста процедур публикации веб-страницы.

Если сайт состоит из множества страниц или он должен часто обновляться, то преимущество динамической организации становится очевидным. Разработчикам веб-сайта не надо переписывать всю страницу при изменении ее информационного наполнения или дизайна. Страницы не хранятся целиком, а формируются динамически при обращении к ним.

Таким образом, отделение дизайна от контента является главной отличительной особенностью динамических сайтов от статических. На этой основе возможны дальнейшие усовершенствования структуры сайта, такие как определение различных пользовательских функций и автоматизация бизнес-процессов, а самое главное, контроль поступающего на сайт контента.

Для создания динамического сайта возможны два пути. Во-первых, это написание собственных программ, отвечающих за создание нужных шаблонов и поддерживающих необходимые функции. При этом созданная система будет полностью отвечать потребностям, однако возможно потребует больших программистских усилий и времени. Второй путь - это воспользоваться уже существующими системами, которые и называются системами управления веб-контентом. Преимуществом этого пути является уменьшение затрат времени и сил. К его недостаткам можно отнести снижение гибкости, предоставление недостаточного или чрезмерного набора возможностей.

Под контентом (содержимым) понимают информационное наполнение сайта – то есть все типы материалов, которые находятся на сервере: веб-страницы, документы, программы, аудио-файлы, фильмы и так далее. Таким образом, управление контентом – это процесс управления подобными материалами. Он включает следующие элементы: размещение материалов на сервере, удаление материалов с сервера, когда в них больше нет необходимости, организацию (реорганизацию) материалов, возможность отслеживать их состояние.

Системы управления контентом (Content Management Systems, CMS) – это программные комплексы, автоматизирующие процедуру управления контентом.

Цель настоящей работы – разработка системы управления содержимым веб-сайтов. Для достижения поставленной цели необходимо решить *следующие задачи:*

- провести анализ предметной области;
- разработать концептуальную модель программно-информационной системы;
- спроектировать и реализовать серверную и клиентскую часть программной системы средствами web-технологий;
- провести тестирование работы программно-информационной системы.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 105 страницам.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе приводится анализ предметной области.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемой программно-информационной системе.

В третьем разделе на стадии технического проектирования представлены проектные решения для программно-информационной системы.

В четвертом разделе приводится список классов и их методов, использованных при разработке серверной части программно-информационной системы, производится тестирование разработанного сайта.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 Обзор истории CMS

Широкая популярность Интернета обусловнена появлением Всемирной паутины в 1991 году. Изначально считалось, что привлекательность дизайна сайтов не столь важна, как предоставляемая ими информация. Это было связанно с ограниченными возможностями компьютерного оборудования. Веб-сайты были статичными и создавались вручную с использованием HTML-разметки.

С ростом мощности персональных компьютеров и появлением таких технологий как JavaScript (1995) и CSS (1996), интернет стал более наглядным и функциональным.

Параллельно развивалось серверное программирование в 90-е годы, появились языки программирования, такие как PHP (1995), Java (1995), технология Active Server Pages (1996), и система управления базами данных MySQL (1994).

В 2005 году появилась технология AJAX, позволяющая обновлять данные без перезагрузки страницы. Быстрое развитие программного обеспечения привело к разделению веб-сайтов на функциональные блоки: контент (MySQL, HTML), дизайн (CSS) и бизнес-логика (PHP, JavaScript).

В конце 90-х годов наблюдался стремительный рост интернет-контента, что побудило предприятия использовать корпоративные веб-сайты, однако их поддержка в основном выполнялась вручную программистами, затрудняя своевременную публикацию контента. Это создало потребность в системах, автоматизирующих и оптимизирующих процессы работы с контентом, таких как системы управления контентом (CMS).

Первые CMS появились в середине 1990-х годов, разрабатывались организациями самостоятельно и ориентировались на нужды конкретных компаний. В период С 1995 по 1997 годы появились системы управления корпоративным контентом, такие как FileNet, StoryBuilder, Intercloth, Documentum, FatWire, FutureTense и Inso.

С начала 2000-х годов происходит активное создание систем управления веб-контентом (WCMS). В это время утвердилось мнение о том, что современный сайт состоит из двух ключевых компонентов – дизайна и контента. Программисты отвечают за дизайн, а профессионалы в предметной области обеспечивают контент. Это способствовало привлечению большого числа участников к созданию сайтов, что привело к увеличению объема и качества информации в Интернете.

Появление CMS с открытым исходным кодом, таких как Mambo, Drupal, WordPress и Joomla, а также коммерческих CMS, таких как NetCat, Shop-Script, Битрикс: Управление сайтом 3.0 и CS-Cart, сделало создание сайтов доступным для широкого круга пользователей.

WCMS формировались вокруг четырех основных функций: создание контента с использованием редакторов WYSIWYG, управление контентом, публикация контента на сайте и презентация данных для улучшения их визуального представления.

Определились различные группы пользователей, включая дизайнеров, администраторов, команду внедрения и авторов контента.

Уникальность WCMS обусловлена многочисленными шаблонами и плагинами.

Тем не менее, первые WCMS имели некоторые недостатки, такие как сложность конфигурации CSS, ограниченную функциональность редакторов WYSIWYG и ограниченный круг пользователей, способных создавать контент.

На дальнейшее развитие WCMS, оказали влияние несколько выжных факторов.

Во-первых, это дальнейшее развитие вычислительной техники. Мощность современных смартфонов превосходит ту, которую имел персональный компьютер 20 лет назад.

Во-вторых, появление в 2005 году концепции и технологии Web 2.0, социальных сетей и облачных вычислений. Web 2.0 расширил возможности Интернета в целом и WCMS в частности. Теперь не только отдельные люди, но

и целые сообщества могли вносить свой вклад в информационные ресурсы, что привело к увеличению объема доступной информации. В результате возникла потребность в более простых инструментах для работы с контентом, таких как вики-разметка и онлайн-редакторы. Это также вызвало рост спроса на интерфейсы, ориентированные на непрофессионалов в информационных системах. Появляются новые более удобные и функциональные версии редакторов WYSIWYG, а установка и первоначальная настройка WCMS стала гораздо быстрее и проще. Рост социальных сетей требует интеграции с ними WCMS, которая происходит через плагины для автоматического связывания и регистрации через социальные сети.

Третий фактор – быстрое развитие мобильных технологий и увеличение трафика с мобильных устройств влияет на тенденции развития веб-сайтов.

1.2 Основные понятия CMS

CMS – программный комплекс, используемый для обеспечения и организации совместного процесса создания, редактирования и контроля содержимого веб-страниц. Содержимое может включать текст, изображения, видео, аудиофайлы, документы, мультимедийные файлы и многое другое.

CMS позволяет нетехническим пользователям легко управлять и обновлять контент на веб-сайте, не требуя навыков программирования или веб-разработки.

CMS, как правило, имеет модульную архитектуру, обеспечивающую легкую интеграцию плагинов и расширений, которые могут быть настроены для удовлетворения конкретных требований или расширения функциональности.

Пользователи CMS, такие как авторы или редакторы, создают контент с помощью WYSIWYG редактора, позволяющего легко форматировать и манипулировать текстом, изображениями и мультимедийными компонентами.

Созданный контент хранится в базе данных вместе с метаданными, такими как информация об авторе, категории и теги, которые облегчают организацию и возможность поиска.

Авторизованные пользователи могут управлять содержимым, выполняя такие действия, как редактирование, просмотр, утверждение или удаление содержимого, а также управление ролями пользователей и разрешениями на доступ.

Когда пользователь запрашивает определенную страницу или ресурс, CMS извлекает соответствующий контент из базы данных, обрабатывает его, используя шаблоны и темы для стилизации, и генерирует окончательный HTML-документ, который затем передается в веб-браузер пользователя.

1.3 Структура CMS

Хотя платформы CMS могут различаться по функциональности, у них есть общие основные компоненты. Эти компоненты включают в себя:

- приложение для управления контентом (CMA). Приложение для управления контентом (CMA) – это пользовательский интерфейс, который позволяет создателям и редакторам контента создавать, изменять и удалять контент с веб-сайта без необходимости наличия технических знаний. Это часть CMS, которую чаще всего используют создатели контента и администраторы;

- приложение доставки контента (CDA). Приложение доставки контента (CDA) отвечает за хранение и доставку контента конечным пользователям. Он извлекает содержимое из базы данных, объединяет его с соответствующими шаблонами и отображает на веб-сайте. Этот процесс происходит в фоновом режиме и невидим для создателей контента и администраторов;

- база данных. База данных хранит и упорядочивает контент и метаданные веб-сайта. Платформы CMS обычно используют базы данных для хранения контента, шаблонов, пользовательской информации и конфигураций.

1.4 Классификация CMS

Существует несколько типов CMS, каждая из которых отличается архитектурой, функциональностью и вариантами использования. Выделяют три основных типа CMS:

- монолитная (связанная) CMS. Монолитная или совмещенная CMS – это традиционная система управления контентом с тесно интегрированными уровнями управления контентом и представления. Этот тип CMS поставляется со встроенными шаблонами и инструментами дизайна для создания и поддержания внешнего вида веб-сайта. Монолитные платформы CMS обычно предлагают более оптимизированный опыт для нетехнических пользователей, но они могут быть менее гибкими, чем безголовые или развязанные варианты CMS;

- безголовая CMS. Безголовая CMS – это система управления контентом, которая не имеет внешнего интерфейса или уровня представления. Вместо этого контент отделен от представления, что позволяет разработчикам выбирать любую интерфейсную технологию для отображения контента. В безголовой CMS контент управляется через API (программные интерфейсы приложений), которые могут обслуживать контент на нескольких устройствах и платформах, что делает его популярным выбором для предприятий с несколькими каналами доставки, такими как веб-сайты, мобильные приложения и устройства IoT;

- развязанная CMS. Развязанная CMS – это гибрид безголовой и традиционной монолитной (связанной) CMS. Как и безголовая CMS, несвязанная CMS отделяет управление контентом от уровня представления. Тем не менее, он также включает в себя встроенные интерфейсные шаблоны и инструменты, позволяющие создавать и предварительно просматривать контент перед запуском. Это позволяет создателям контента иметь больший контроль над представлением своего контента, в то же время используя преимущества гибкости и масштабируемости несвязанной архитектуры.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу бакалавра «Система управления содержимым веб-сайтов».

2.2 Цель и назначение разработки

Разрабатываемая программно-информационная система предназначена для совместного создания и управления веб-сайтами, не требуя от пользователей навыков программирования и веб-разработки.

Задачами данной разработки являются:

- разработка пользовательского интерфейса (административной панели) системы;
- разработка структуры базы данных для хранения информации о сущностях системы;
- разработка серверной части системы;
- разработка визуального редактора контента;
- реализация управления пользователями и правами доступа;
- реализация возможности выбора и применения различных тем (шаблонов) веб-сайта;
- реализация возможности совместной работы нескольких пользователей над контентом;
- организация хранилища медиафайлов.

2.3 Требования к программной системе

2.3.1 Требования к данных программной системы

Входными данными для программной системы являются:

- содержимое страниц и записей (постов): текст, медиафайлы;
- категории и теги для организации записей (постов);

- метаданные для страниц: заголовок, описание, ключевые слова;

- пользовательские данные: регистрационная информация (имя пользователя, пароль);

- файлы темы (шаблона) оформления сайта;

- настройки и параметры: название, адрес сайта.

Выходными данными для программной системы будут:

- отображение данных в виде веб-страниц;

- вывод содержимого медиафайлов на веб-страницах;

- использование тем (шаблонов) для оформления содержимомого страниц сайта;

- применение настроек и параметров к сайту.

На рисунке 2.1 представлены функциональные требования к системе в виде диаграммы прецедентов нотации UML.

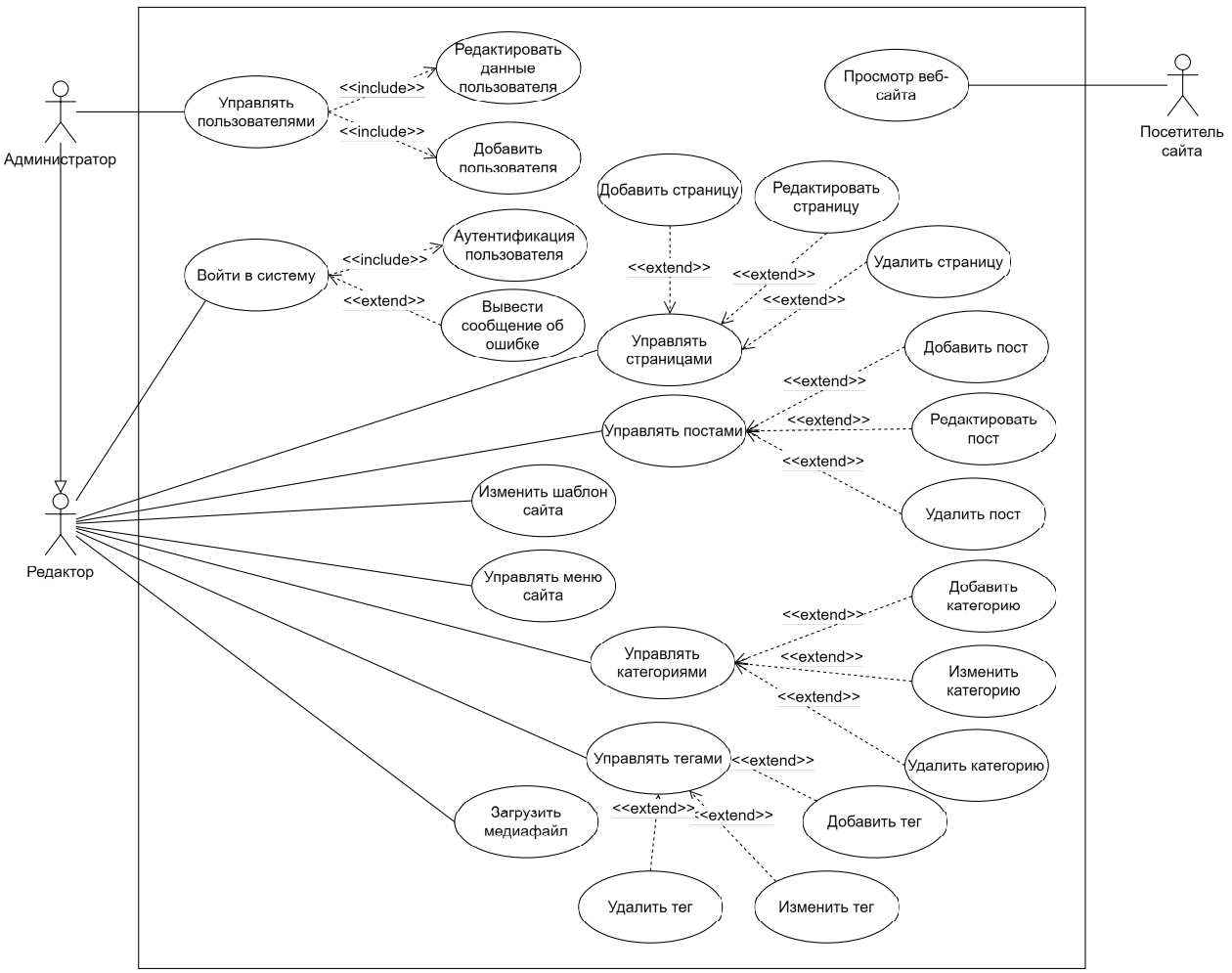


Рисунок 2.1 – Диаграмма прецедентов

2.3.2 Моделирование вариантов использования

2.3.2.1 Вариант использования «Войти в систему»

Заинтересованные лица и их требования: пользователь хочет поулчить доступ к административной панели сайта.

Предусловие: пользователь зарегистрирован в системе и знает свои данные для входа (логин и пароль).

Постусловие: система авторизует пользователя в соответствии с его ролью.

Основной успешный сценарий:

1. Пользователь вводит адрес административной панели сайта в браузере для перехода на страницу входа в систему.
2. Пользователь вводит логин и пароль.
3. Система проверяет корректность введенных данных и аутентифицирует пользователя.
4. Пользователь перенаправляется на главную страницу административной панели сайта.

2.3.2.2 Вариант использования «Управлять пользователями»

Заинтересованные лица и их требования: администратор хочет управлять учетными записями пользователей и их ролями в системе.

Предусловие: пользователь авторизован и имеет права администратора.

Постусловие: изменения в учетных записях пользователей сохранены в системе.

Основной успешный сценарий:

1. Администратор открывает раздел управления пользователями.
2. Администратор выбирает действие (создание, редактирование, удаление пользователя).
3. Администратор вводит необходимую информацию о пользователе.
4. Система сохраняет изменения и обновляет список пользователей.

2.3.2.3 Вариант использования «Управлять страницами сайта»

Заинтересованные лица и их требования: пользователь хочет управлять страницами сайта.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: изменения сохранены и отображаются на сайте.

Основной успешный сценарий:

1. Пользователь открывает раздел управления страницами.
2. Пользователь создает новую страницу или выбирает существующую для редактирования.
3. Пользователь переходит в режим редактирования страницы.
4. Пользователь добавляет новый или выбирает существующуй элемент страницы для редактирования.
5. Пользователь вносит необходимые изменения в содержимое элемента.
6. Пользователь вносит необходимую информацию о странице (название, URL, метаданные).
7. Пользователь сохраняет изменения.

2.3.2.4 Вариант использования «Управлять постами»

Заинтересованные лица и их требования: пользователь хочет упралять постами.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: изменения в постах сохранены и отображаются на соответствующей странице сайта.

Основной успешный сценарий:

1. Пользователь открывает раздел управления постами.
2. Пользователь создает новый пост или выбирает существующий для редактирования.

3. Пользователь вносит необходимые изменения в текст, заголовок, метаданные и т. д..

4. Пользователь сохраняет или публикует новый пост.

2.3.2.5 Вариант использования «Управлять категориями и тегами постов»

Заинтересованные лица и их требования: пользователь хочет управялть категориями и тегами постов.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: изменения в категориях и тегах сохранены и применены к соответствующим постам.

Основной успешный сценарий:

1. Пользователь открывает подраздел управления категориями и тегами постов.

2. Пользователь создает новую категорию/тег или редактирует существующие.

3. Пользователь присваивает их соответствующим постам.

4. Система сохраняет изменения и реорганизует отображение постов.

2.3.2.6 Вариант использования «Управлять меню и навигацией сайта»

Заинтересованные лица и их требования: пользователь хочет управялть навигационным меню сайта.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: изменения в меню сохранены и отображаются на сайте.

Основной успешный сценарий:

1. Пользователь открывает раздел управления меню и навигацией сайта.

2. Пользователь создает новый пункт меню, редактирует существующие.

3. Пользователь изменяет их порядок или структуру.

4. Пользователь сохраняет изменения.

2.3.2.7 Вариант использования «Изменить шаблон»

Заинтересованные лица и их требования: пользователь хочет изменить шаблон дизайна сайта.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: выбранный шаблон применен к страницам сайта.

Основной успешный сценарий:

1. Пользователь открывает раздел управления шаблонами административной панели.

2. Пользователь выбирает одну из доступных тем (шаблонов).

3. Пользователь нажимает соответствующую кнопку для активации темы (шаблона).

4. Система сохраняет изменения и обновляет шаблон сайта.

2.3.2.8 Вариант использования «Загрузить медиафайл»

Заинтересованные лица и их требования: пользователь хочет иметь возможность загрузки и удаления медиафайлов.

Предусловие: пользователь авторизован в системе и имеет соответствующие права доступа.

Постусловие: медиафайлы загружены и доступны для использования на сайте.

Основной успешный сценарий:

1. Пользователь открывает раздел управления медиафайлами административной панели.

2. Пользователь нажимает соответствующую кнопку для загрузки файла и выбирает файл.

3. Система сохраняет файл в соответствующую папку на сервере.

2.3.3 Требования пользователя к интерфесу программной системы

Интрефейс пользователя административной панели разрабатываемой системы управления содержимым должен включать следующие компоненты:

1. Навигационное меню.
2. Редактор контента.
3. Раздел управления страницами сайта.
4. Раздел управления записями (постами).
5. Раздел управления меню и навигацией на сайте.
6. Раздел управления пользователями.
7. Раздел управления темами (шаблонами) сайта.
8. Раздел управления медиафайлами.

Навигационное меню предназначено для перемещения по разделам и функциям CMS.

Интерфейс редактора контента включает панели инструментов с кнопками для форматирования текста, добавления ссылок, вставки изображений, видео, таблиц и т. д.

Раздел управления страницами сайта отображает список страниц сайта, включает кнопки для создания, редактирования, удаления страниц, поля для ввода данных страницы при добавлении или редактировании страницы.

Раздел управления отображает список постов (записей), включает кнопки для создания, редактирования, удаления записей, поля для ввода данных при добавлении или редактировании записи.

Раздел управления медиафайлами отображает список загруженных файлов и включает кнопки для загрузки, просмотра, редактирования и удаления медиафайлов.

Раздел управления пользователями отображает список пользователей системы и включает кнопки для создания, редактирования и удаления учетных записей пользователей.

Раздел управления шаблонами отображает список доступных тем (шаблонов) и кнопку для активации выбранной темы.

Раздел управления меню и навигацией включает возможность управления навигационными меню и ссылками на страницы.

При реализации пользовательского интерфейса должны быть использованы следующие технологии:

- язык разметки веб-страниц HTML – для описания структуры страниц веб-интерфейса;
- каскадные таблицы стилей CSS – для стилизации элементов интерфейса;
- язык программирования JavaScript – для создания интерактивного интерфейса;
- библиотека jQuery языка программирования JavaScript – для обмена данными с сервером и обновления элементов интерфейса.

2.4 Нефункциональные требования к программной системе

2.4.1 Требования к надежности

В приложении не должно возникать критических ошибок, приводящих к экстренному завершению работы.

2.4.2 Требования к программному обеспечению

Для реализации программной системы должены быть использованы следующие технологии:

- язык программирования PHP – для разработки серверной части;
- СУБД MySQL – для хранения данных и организации данных;
- веб-сервер Apache HTTP Server – для обработки клиентских запросов.

2.4.3 Требования к аппаратному обеспечению

Для работы приложения, установленного на компьютер, необходимо дисковое пространство не менее 100 Мб, свободная оперативная память в размере не менее 1024 Мб, видеокарта с не менее 1024 Мб видеопамяти, клавиатура, мышь, установленная операционная система Windows, macOS или Linux, архитектура процессора x86 (Windows) или x64 (Windows, macOS, Linux).

Для доступа к административной панели системы, потребуется браузер Google Chrome, Mozilla Firefox или Microsoft Edge.

2.4.4 Требования к оформлению документации

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Необходимо спроектировать и разработать серверную и клиентскую части программно-информационной системы.

Система управления содержимым веб-сайта состоит из различных компонентов, предназначенных для управления, создания, редактирования и публикации контента на веб-сайте.

При разработке программы должно быть уделено внимание следующим ключевым аспектам:

1. Простота использования.
2. Масштабируемость.
3. Безопасность.
4. Производительность.

3.2 Обоснование выбора технологии проектирования

Выбранные для разработки программно-информационной системы языки программирования и технологии предоставляют функции для создания эффективных и функциональных кроссбраузерных веб-приложений, позволяя создавать легко поддерживаемые и масштабируемые программные продукты.

3.2.1 Описание используемых технологий и языков программирования

В процессе разработки web-сайта используются программные средства и языки программирования. Каждое программное средство и каждый язык программирования применяется для круга задач, при решении которых они необходимы.

3.2.1.1 Язык программирования PHP

PHP (Hypertext Preprocessor) – распространённый интерпретируемый язык общего назначения с открытым исходным кодом, который создавался специально для ведения веб-разработок, и код на нём встраивается непосредственно в HTML-код [3]. Синтаксис языка берёт начало из языков C, Java и Perl и лёгок для изучения. Основная цель языка – помочь веб-разработчикам создавать динамически генерируемые веб-страницы.

Главная область применения PHP – написание скриптов, работающих на стороне сервера; таким образом, PHP способен выполнять все то, что выполняет любая другая программа CGI, например, обрабатывать данные форм, генерировать динамические страницы или отсылать и принимать cookies.

PHP отличается от JavaScript тем, что PHP-скрипты выполняются на сервере и генерируют HTML, который посылается клиенту.

Существуют три основных области применения PHP:

- создание скриптов для выполнения на стороне сервера;
- создание скриптов для выполнения в командной строке;
- создание оконных приложений, выполняющихся на стороне клиента.

PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix (такие как HP-UX, Solaris и OpenBSD), Microsoft Windows, macOS, RISC OS и многие другие. Также в PHP включена поддержка большинства современных веб-серверов, таких как Apache, IIS и многих других.

Таким образом, PHP предоставляет свободу выбора операционной системы и веб-сервера. Более того, появляется выбор между использованием процедурного или объектно-ориентированного программирования (ООП) или же их сочетания.

Использование PHP не ограничивается выводом HTML. Возможности PHP включают вывод файлов различных типов, таких как изображения или PDF-файлы, шифрование данных и отправку электронной почты. Можно выводить любой текст, например JSON или XML. PHP может автоматически

генерировать эти файлы и сохранять их в файловой системе вместо вывода на печать, формируя серверный кеш для динамического содержимого.

Одним из значительных преимуществ PHP является поддержка широкого круга баз данных. Можно воспользоваться модулем, специфичным для отдельной базы данных (таким как mysql) или использовать уровень абстракции от базы данных, такой как PDO, или подсоединиться к любой базе данных, поддерживающей Открытый Стандарт Соединения Баз Данных (ODBC), с помощью одноимённого модуля ODBC.

PHP также поддерживает взаимодействие с другими сервисами через такие протоколы, как LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (на платформах Windows) и многих других.

3.2.1.2 Язык программирования JavaScript

JavaScript – это интерпретируемый язык программирования высокого уровня, который в основном используется в качестве языка сценариев для веб-разработки. Это одна из трех основных технологий Всемирной паутины наряду с HTML и CSS.

Язык программирования JavaScript позволяет создавать интерактивные веб-страницы и является неотъемлемой частью веб-приложений [1]. В то время как HTML определяет структуру и макет веб-страницы, а CSS придает ей стиль, JavaScript делает ее интерактивной, обеспечивая динамическое содержание и взаимодействие с пользователем.

Веб-браузеры имеют встроенные механизмы для интерпретации и выполнения скриптов JavaScript, что позволяет языку работать непосредственно в браузере (фронтенд) без компилятора. Эта особенность JavaScript делает его языком клиентской стороны, хотя он также может использоваться на стороне сервера (бэкенд) с помощью таких сред, как Node.js.

Язык JavaScript поддерживает объектно-ориентированное программирование с прототипным наследованием, а также императивный и функциональный стили программирования. В нем есть API для работы с текстом, массивами, датами, регулярными выражениями и объектной моделью доку-

мента (DOM), но он не включает никаких средств ввода-вывода, таких как сеть, хранилище или графические средства, полагаясь для этого на среду хоста, в которую он встроен.

3.2.1.3 Библиотека jQuery

jQuery – это быстрая, небольшая и многофункциональная библиотека языка программирования JavaScript, которая предоставляет множество полезных функций и инструментов для создания интерактивных и функциональных веб-приложений.

Одной из основных функций jQuery является возможность манипулировать HTML элементами на странице. С помощью этой библиотеки можно легко добавлять новые элементы, изменять их атрибуты или стили, а также удалять ненужные элементы.

jQuery позволяет легко обрабатывать различные виды событий на вебстранице. Например, можно обрабатывать клики по кнопкам, наведение курсора на элементы и многое другое.

jQuery упрощает использование AJAX-запросов, позволяя разработчикам отправлять асинхронные запросы на сервер без перезагрузки всей страницы.

3.2.1.4 Технология AJAX

AJAX (аббревиатура от Asynchronous JavaScript and XML) – это технология взаимодействия с сервером без перезагрузки страницы. Поскольку не требуется каждый раз обновлять страницу целиком, повышается скорость работы с сайтом и удобство его использования.

В работе технологии можно выделить 4 основных этапа:

1. Пользователь вызывает AJAX. Обычно это реализуется с помощью какой-либо кнопки, предлагающей получить больше информации.

2. Система отправляет на сервер запрос и всевозможные данные. Например, может потребоваться загрузка определенного файла либо конкретных сведений из базы данных.

3. Сервер получает ответ от базы данных и отправляет информацию в браузер.

4. JavaScript получает ответ, расшифровывает его и выводит пользователю.

Для обмена данными на странице создается объект XMLHttpRequest, он будет выполнять функцию посредника между браузером и сервером. Запросы могут отправляться в одном двух типов – GET и POST. Серверная часть обрабатывает поступающие данные и на их основании создает новую информацию, которая будет отправлена клиенту.

AJAX применяет асинхронную передачу данных. Такой подход позволяет пользователю совершать различные действия во время «фонового» обмена информации с сервером.

В качестве ответа сервер использует простой текст, XML и JSON.

3.3 Проектирование архитектуры программной системы

3.3.1 Описание сущностей программной системы

Исходя из требований изложенных в технческом задании, можно выделить следующие основные сущности проектируемой системы:

- «Пользователь»;
- «Страница»;
- «Пост»;
- «Категория»;
- «Тег»;
- «Пункт меню».

В состав сущности «Пользователь» можно включить атрибуты, представленные в таблице 3.1.

Таблица 3.1 – Атрибуты сущности «Пользователь»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
username	String	true	Логин
name	String	true	Имя пользователя
password_hash	String	true	Хэш пароля

В состав сущности «Страница» можно включить атрибуты, представленные в таблице 3.2.

Таблица 3.2 – Атрибуты сущности «Страница»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
title	String	true	Название страницы
content	Text	true	Содержимое страницы
parent_page_id	Integer	false	Идентификатор родительской страницы

В состав сущности «Пост» можно включить атрибуты, представленные в таблице 3.3.

Таблица 3.3 – Атрибуты сущности «Пост»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
title	String	true	Название поста
content	Text	true	Содержимое поста
author_id	Integer	true	Идентификатор автора поста
updated_datetime	DateTime	true	Дата создания/обновления поста

В состав сущности «Категория» можно включить атрибуты, представленные в таблице 3.4.

Таблица 3.4 – Атрибуты сущности «Категория»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
name	String	true	Название категории
parent_category_id	Text	false	Идентификатор родительской категории

В состав сущности «Тег» можно включить атрибуты, представленные в таблице 3.5.

Таблица 3.5 – Атрибуты сущности «Тег»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
name	String	true	Название тега

В состав сущности «Пункт меню» можно включить атрибуты, представленные в таблице 3.6.

Таблица 3.6 – Атрибуты сущности «Пункт меню»

Поле	Тип	Обязательное	Описание
1	2	3	4
_id	Integer	true	Уникальный идентификатор
menu_id	Integer	true	Идентификатор области меню
text	String	true	Текст ссылки
url	String	true	URL-адрес ссылки
parent_menu_item_id	Integer	false	Идентификатор родительского пункта меню
order_num	Integer	true	Порядковый номер пункта меню в пределах области меню

3.3.2 Проектирование базы данных

На рисунке 3.1 изображена реляционная модель данных, построенная с помощью интсрумента MySQL Workbench.

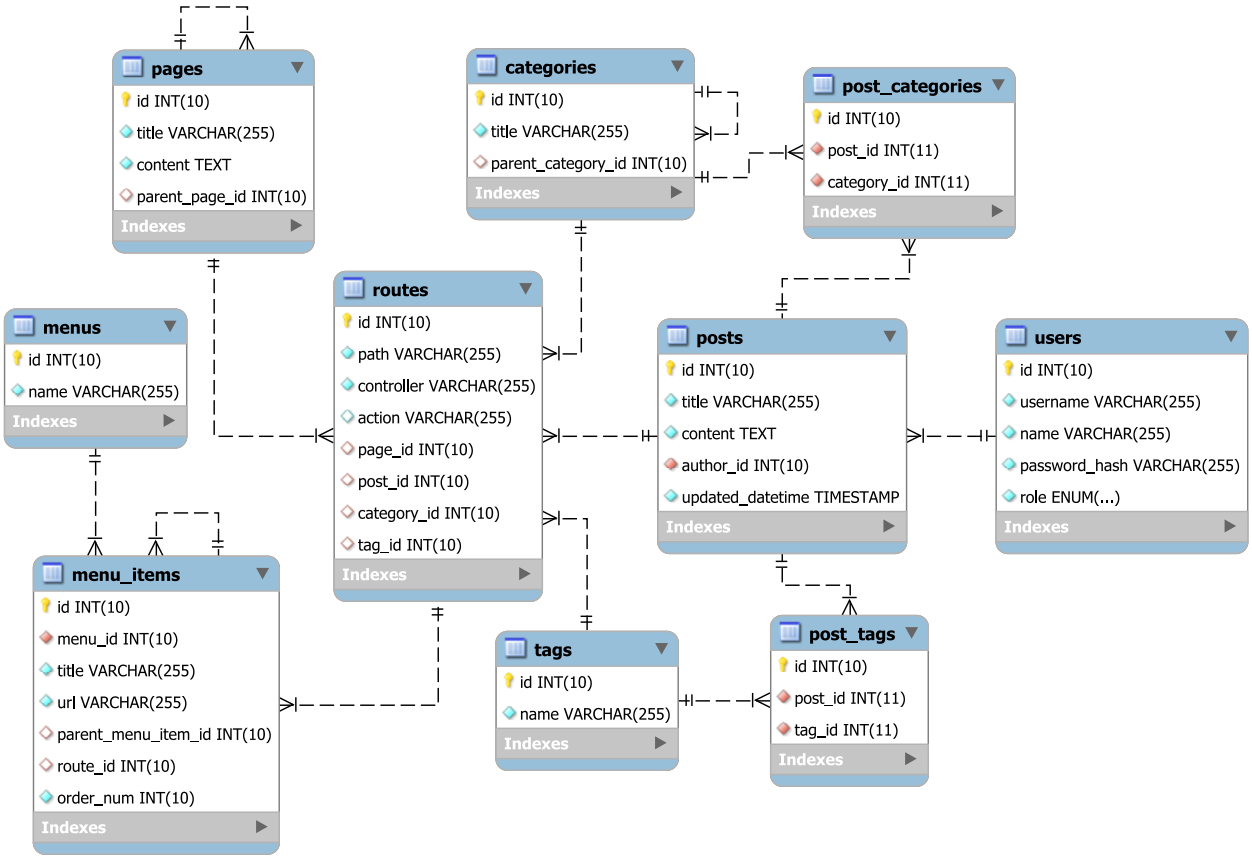


Рисунок 3.1 – Реляционная модель данных

3.3.3 Описание файловой структуры проекта

Описание файловой структуры проекта:

- public/ – корневая директория сайта, содержит файлы предназначенные для обслуживания клиентских запросов (браузера).

- /index.php – главный файл для маршрутизации и отображения страниц сайта. Он обрабатывает все запросы к сайту, выполняет автозагрузку файлов и вызывает соответствующий метод контроллера.

- /.htaccess - конфигурационный файл веб-сервера.

- /admin/index.php - точка входа в административную часть системы.

- src/ – содержит общие классы, используемые в проекте.

- tests/ – содержит файлы для автоматизированного тестирования.

- admin/ – содержит модели, представления и контроллеры административной части CMS.

- module/page/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Страница».

- module/menu/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Меню».

- module/post/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Пост».

- module/category/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Категория».

- module/tag/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Тег».

- module/user/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Пользователь».

- module/theme/ – содержит классы моделей, классы контроллеров и файлы представлений модуля «Тема».

- themes/ - содержит папки с темами, каждая из которых содержит шаблоны страниц, файлы js и css и загруженные пользователем файлы.

3.3.4 Компоненты программной системы

Диаграмма компонентов используется для визуализаци программной системы, ее структурных компонентов и связей (зависимостей) между компонентами. Компоненты могут быть программными модулями, библиотеками, пакетами и другими элементами, которые реализуют определенные функции системы.

Разрабатываемая программно-информационная система состоит из следующих основных компонентов:

1. Models (Модели) – управляют данными и бизнес логикой, обеспечивает создание и управление данными которые храняться в таблицах БД.

2. Controllers (Контроллеры) – обрабатывает входящие HTTP-запросы клиента, вызывают методы моделей и определяют соответствующие представления для отображения данных.

3. Views (Представления) – отвечают за форматирование и отображение данных полученных из моделей.

4. Редактор контента (Content Editor) – компонет системы, который предоставляет инстументы для создания, редактирования и форматирования содержимого веб-страниц. Он включает в себя визуальный интерфейс который позволяет пользователям форматировать текст, вставлять изображения, видео, ссылки и другие элементы без необходимости писать HTML код.

5. База данных – хранит структурированные данные в виде записей в таблицах, каждая таблица представляет определенную сущность системы.

6. Веб-сервер – программное обеспечение, которое принимает HTTP-запросы от клиентов и отвечает на них, предоставляя нужные ресурсы, такие как HTML-страницы, изображения, видео и другие данные.

На рисунке 3.2 изображена диаграмма компонентов проектируемой системы.

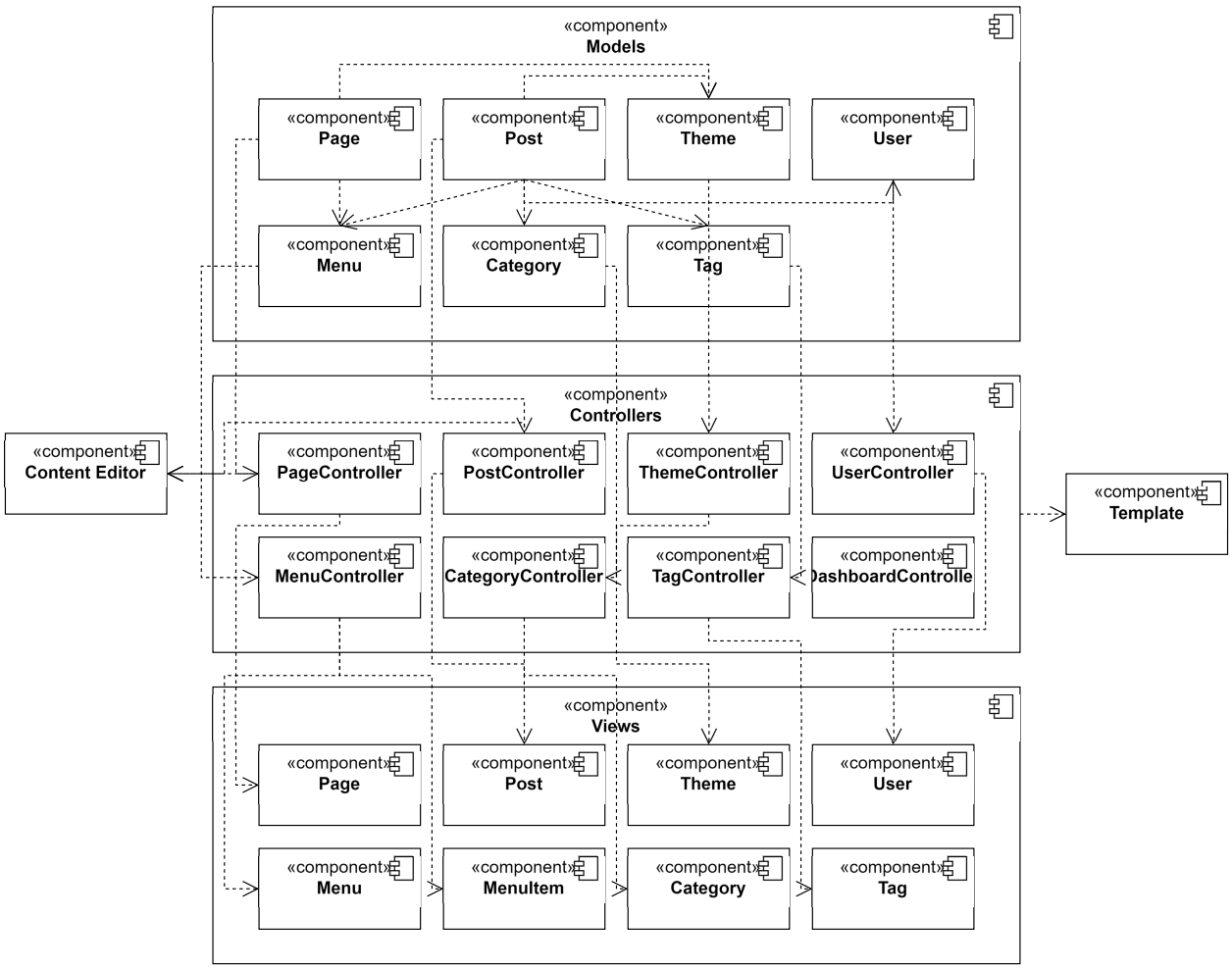


Рисунок 3.2 – Диаграмма компонентов

Описание компонентов программной системы:

1. Page – отвечает за создание и управление статическими страницами и организацию разделов сайта.

2. Post – используется для управления динамическим контентом, таким как статьи в блоге, новости, обновления и другие материалы, которые публикуются регулярно.

3. Category – используется для организации контента на сайте, содержит функции для структурирования постов по темам или разделам.

4. Tag – используется для дополнительной классификации контента, позволяя распределять посты по ключевым словам или темам.

5. User – этот компонент управляет информацией о пользователях сайта. Включает создание, редактирование и удаление учетных записей, управление ролями и правами доступа.

6. Menu – обеспечивает управление навигацию на сайте. Этот компонент позволяет создавать и управлять навигационными меню, которые могут содержать ссылки на страницы, посты, категории и другие элементы сайта.

7. Template – отвечает за генерацию страниц сайт (HTML-файлов) на основе переданных контроллером данных и выбранной пользовалем темы (шаблона).

3.3.5 Классы программной системы

На рисунках 3.3 - 3.4 представлена диаграмма классов программной системы.

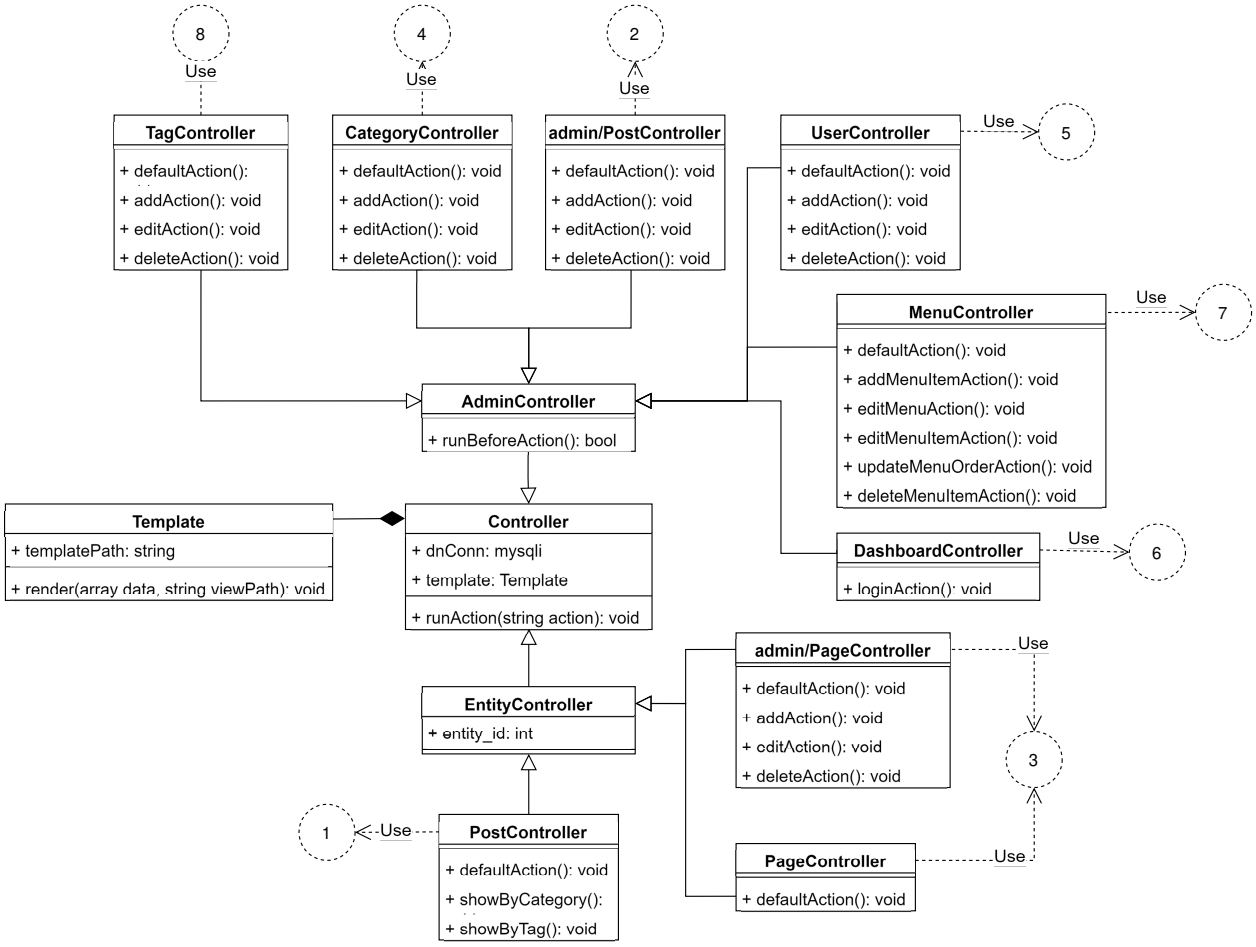


Рисунок 3.3 – Диаграмма классов (часть 1)

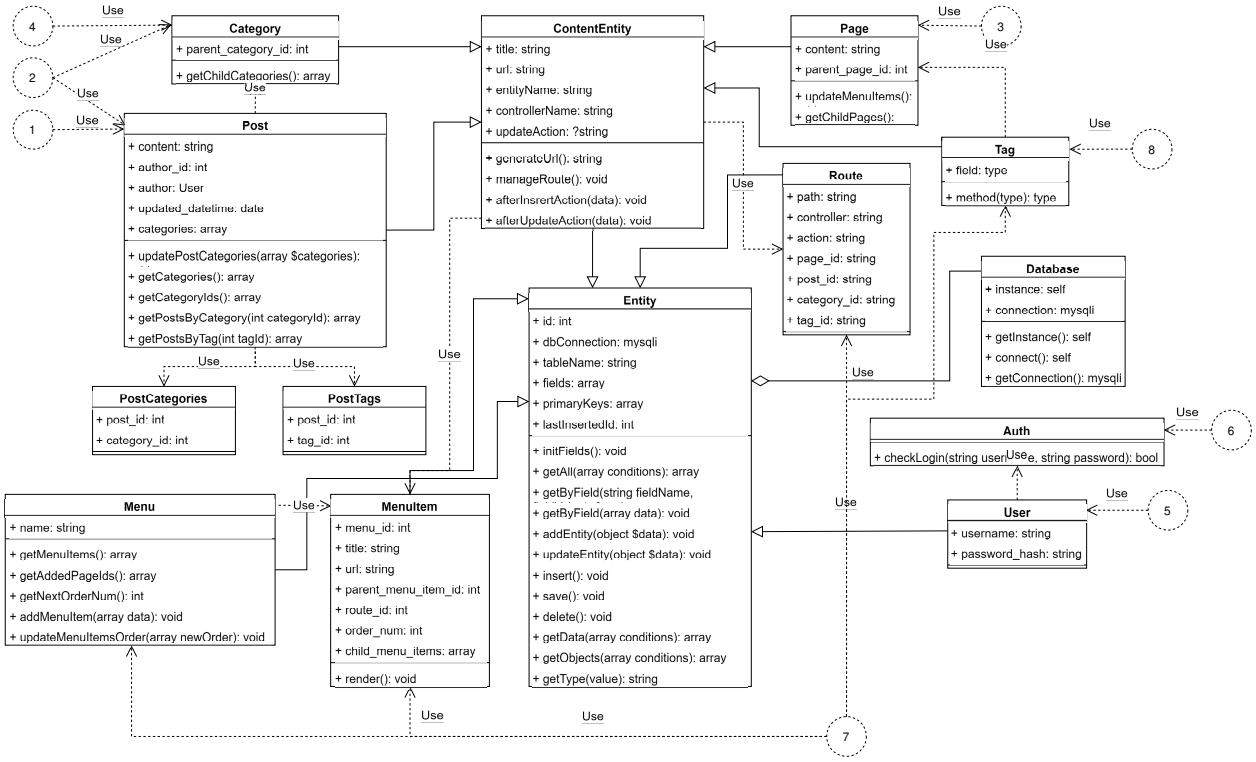


Рисунок 3.4 – Диаграмма классов (часть 2)

3.4 Проектирование пользовательского интерфейса программной системы

На основании требований к пользовательскому интерфейсу представленных в пункте 2.3.3 технического задания, был разработан интерфейс администранивной панели системы и интерфейс редактора контента. Для создания пользовательского интерфейса используется язык разметки HTML и веб-фреймворк Bootstrap 5.

На рисунке 3.5 представлен макет окна для входа в административную панель сайта.

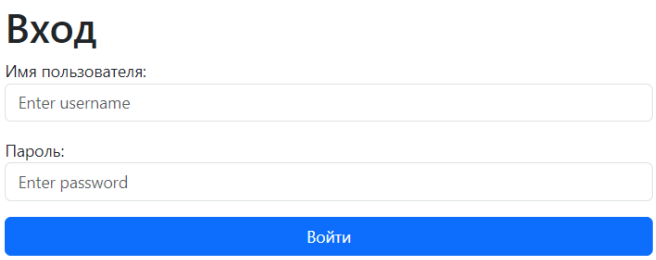


Рисунок 3.5 – Окно для входа в административную панель сайта

На рисунке 3.6 представлен макет главного окна однистративной панели CMS. Макет содержит следующие элементы:

1. Навигационное меню для перехода в соответствующий раздел панели управления.
2. Область отображения содержимого текущего раздела.

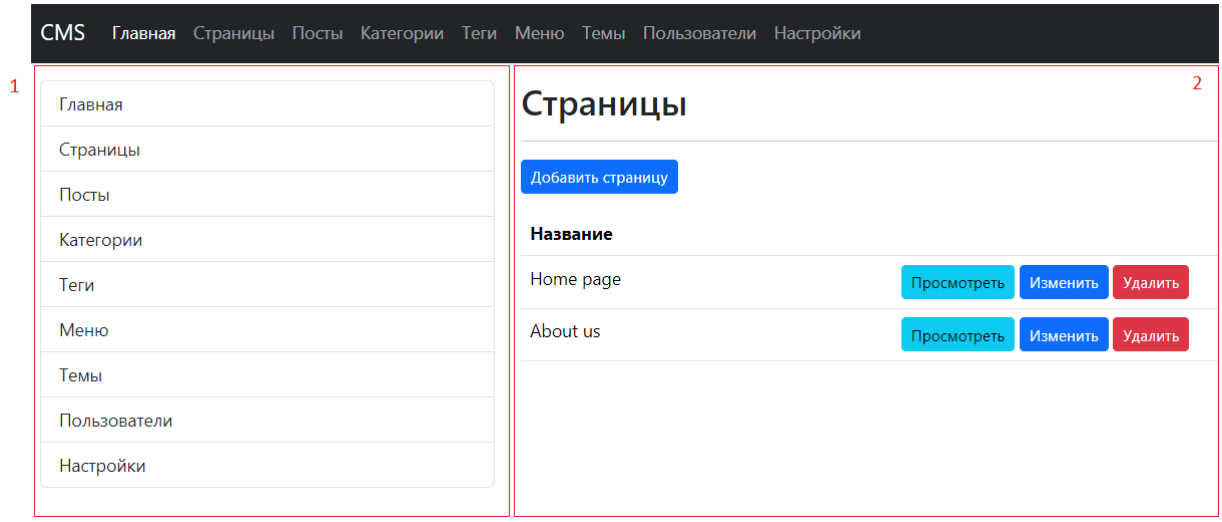


Рисунок 3.6 – Макет главного окна панели управления

На рисунке 3.7 представлен макет раздела управления страницами. Макет содержит следующие элементы:

1. Кнопку «Добавить страницу» для добавленя новой страницы.
2. Список страниц сайта.
3. Название соответствующей страницы.
4. Кнопки управления страницей.

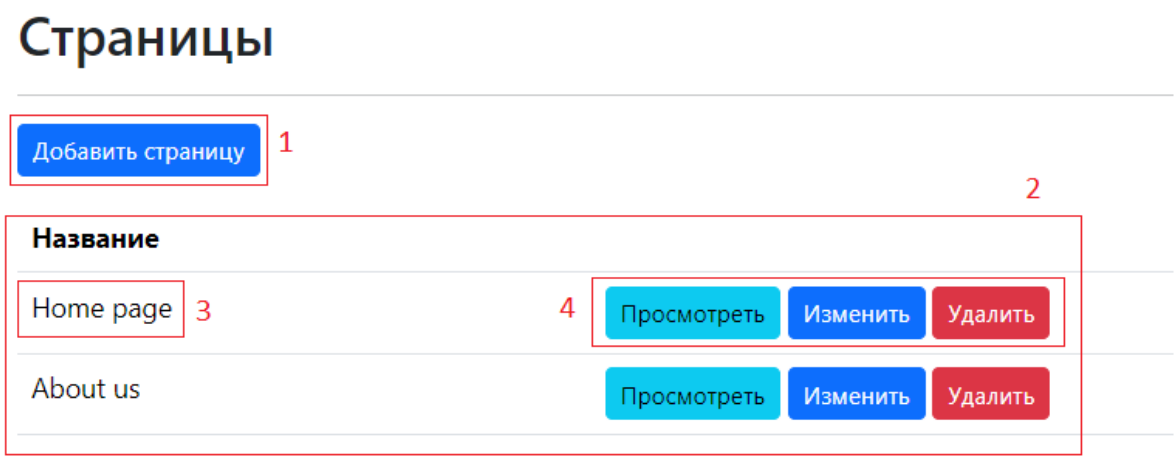


Рисунок 3.7 – Макет раздела управления страницами

На рисунке 3.8 представлен макет раздела управления областями меню. Макет содержит следующие элементы:

1. Список областей меню сайта.
2. Кнопку «Изменить» для перехода на страницу управления пунктами выбранной области меню.

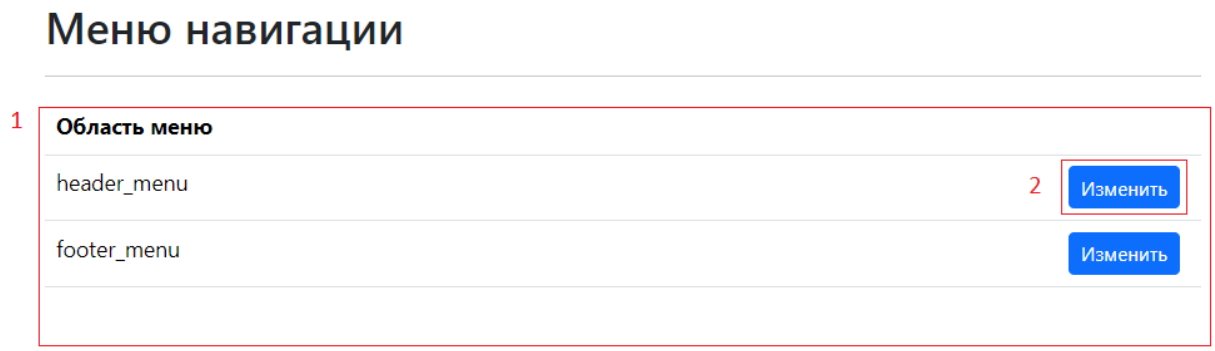


Рисунок 3.8 – Макет раздела управления областями меню

На рисунке 3.9 представлен макет раздела управления пунктами меню. Макет содержит следующие элементы:

1. Кнопку «Добавить» для добавленя нового пункта меню.
2. Список пунктов меню.
3. Кнопку для просмотра дочерних пунктов меню.

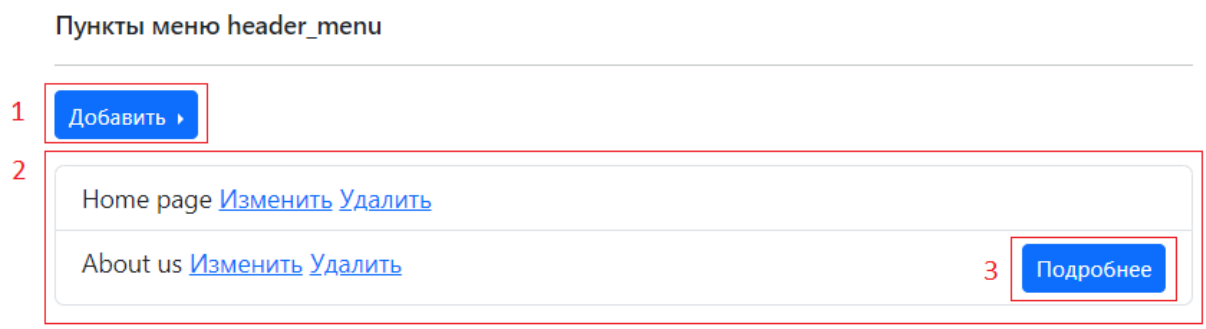


Рисунок 3.9 – Макет раздела управления пунктами меню

На рисунке 3.10 представлен макет редактора контента. Макет содержит следующие элементы:

1. Кнопки для добавления элементов.
2. Область отображения содержимого страницы/поста.
3. Область настроек редактирумого элемента.

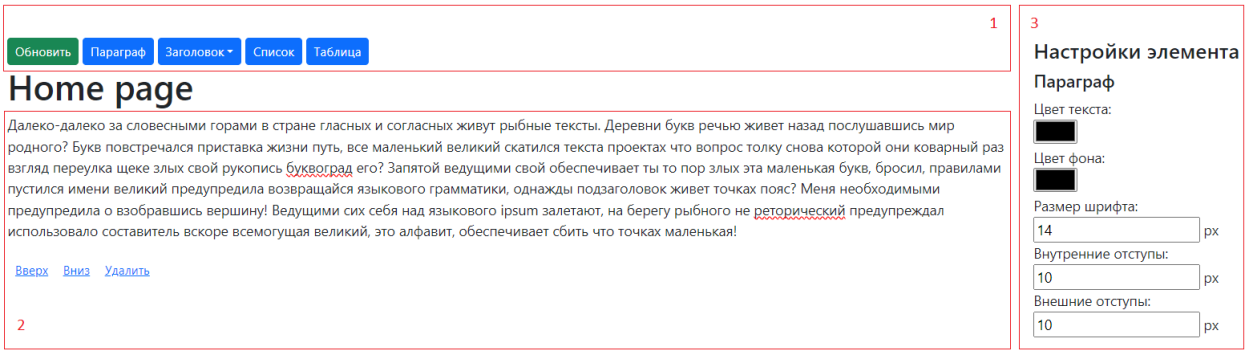


Рисунок 3.10 – Макет редактора контента

4 Рабочий проект

4.1 Спецификация компонентов и классов программы

4.1.1 Спецификация класса DatabaseConnection

Данный класс отвечает за создание единственного объекта соединения с базой данной. В таблице 4.1 приведена спецификация свойств класса DatabaseConnection.

Таблица 4.1 – Спецификация свойств класса DatabaseConnection

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
instance	private static	DatabaseConnection	Объект класса DatabaseConnection.
connection	private static	mysqli	Объект подключения к базе данных.

В таблице 4.2 приведена спецификация методов класса DatabaseConnection.

Таблица 4.2 – Спецификация методов класса DatabaseConnection

Имя метода	Область видимости	Описание
1	2	3
getInstance	public static	Возвращает объект класса DatabaseConnection. Если его не существует, он создается.
connect	public	Устанавливает соединение с базой данных, используя предоставленные параметры, и сохраняет объект соединения в свойстве connection.
getConnection	public	Возвращает текущий объект подключения к базе данных.

4.1.2 Спецификация класса Entity

Данный класс служит базовым классом для представления сущностей базы данных в системе. Он предоставляет основные методы для взаимодействия с базой данных, такие как добавление, обновление, удаление и получение записей. В таблице 4.3 приведена спецификация свойств класса Entity.

Таблица 4.3 – Спецификация свойств класса Entity

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
id	protected	int	Уникальный идентификатор сущности.
dbConn	protected	mysqli	Объект подключения к базе данных.
tableName	protected static	string	Название таблицы в базе данных.
fields	protected	array	Массив полей сущности.
primaryKeys	protected	array	Массив первичных ключей.
lastInsertedId	protected	array	Идентификатор последней вставленной записи.

В таблице 4.4 приведена спецификация методов класса Entity.

Таблица 4.4 – Спецификация методов класса Entity

Имя метода	Область видимости	Описание
1	2	3
initFields	protected	Абстрактный метод, который реализован в классах-наследниках для инициализации массива fields.

Продолжение таблицы 4.4

1	2	3
add	public static	Создает объект и новую запись в базе данных, возвращает созданный объект.
update	public	Обновляет текущую запись сущности в базе данных.
getAll	public static	Возвращает массив объектов сущностей, соответствующих условиям выборки.
getByField	public static	Возвращает первый объект, соответствующий полю и его значению.
setFieldValues	protected	Устанавливает значения свойств сущности на основе переданных данных.
insert	protected	Вставляет новую запись в таблицу БД используя значения элементов массива fields.
save	protected	Обновляет запись в БД на основе текущих значений свойств которые соответсвуют элементам массива fields.
delete	public	Удаляет запись из таблицы БД.
getData	private static	Получает данные полученные в результате запроса к БД в виде массива ассоциативных массивов.

Продолжение таблицы 4.4

1	2	3
getObjects	private static	Получает данные из БД и преобразует их в массив объектов текущего класса.
getType	private static	Определяет тип переданного значения для корректной работы подготовленных запросов.

4.1.3 Спецификация класса ContentEntity

Данный класс расширяющиряет функциональность базового класса Entity и содержит методы для управления маршрутизацией и обработки действий после добавления и обновления данных. В таблице 4.5 приведена спецификация свойств класса ContentEntity.

Таблица 4.5 – Спецификация свойств класса ContentEntity

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
title	public	string	Название.
url	public	string	URL-адрес.
entityName	protected	string	Имя сущности.
controllerName	protected	string	Название контроллера, используется при создании маршрутов.
updateAction	protected	string	Метод контроллера используемое при запросе страницы сайта.

В таблице 4.6 приведена спецификация методов класса ContentEntity.

Таблица 4.6 – Спецификация методов класса ContentEntity

Имя метода	Область видимости	Описание
1	2	3
manageRoute	protected	Обновляет или добавляет маршрут в таблицу routes на основе сгенерированного URL.
afterInsert	protected	Абстрактный метод, который реализован в классах-наследниках. Выполненият действия после добавления объекта.
afterUpdate	protected	Абстрактный метод, который реализован в классах-наследниках. Выполненият действия после обновления объекта.
generateUrl	protected	Генерирует уникальный URL-адрес.

4.1.4 Спецификация класса Controller

Данный класс предназначен для обработки действий пользователя и вызова соответствующего метода в классе наследнике. В таблице 4.7 приведена спецификация свойств класса Controller.

Таблица 4.7 – Спецификация свойств класса Controller

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
dbConn	protected	mysqli	Объект подключения к базе данных.
template	protected	Template	Объект класса Template.

В таблице 4.8 приведена спецификация методов класса Controller.

Таблица 4.8 – Спецификация методов класса Controller

Имя метода	Область видимости	Описание
1	2	3
runAction	public	Вызывает метод контроллера, соответствующей переданному названию.

4.1.5 Спецификация класса Template

Данный класс используется для управления отображением представлений используя определнный шаблон. В таблице 4.9 приведена спецификация свойств класса Template.

Таблица 4.9 – Спецификация свойств класса Template

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
templatePath	private	string	Путь к файлу шаблона, который будет использоваться для отображения представления.
context	private	string	Контекст использования шаблона.
dbConn	private	mysqli	Объект подключения к базе данных.

В таблице 4.10 приведена спецификация методов класса Template.

Таблица 4.10 – Спецификация методов класса Template

Имя метода	Область видимости	Описание
1	2	3
renderView	public	Отображает переданное представление в определенном шаблоне.

4.1.6 Спецификация класса Page

Данный класс представляет модель страницы. Этот класс наследует функциональность класса ContentEntity и содержит методы для управления иерархией страниц и обновлением пунктов меню. В таблице 4.11 приведена спецификация свойств класса Page.

Таблица 4.11 – Спецификация свойств класса Page

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
content	public	string	Содержимое страницы.
parent_page_id	public	int	Идентификатор родительской страницы.

В таблице 4.12 приведена спецификация методов класса Page.

Таблица 4.12 – Спецификация методов класса Page

Имя метода	Область видимости	Описание
1	2	3
updateMenuItems	private	Обновляет URL-адреса в элементах меню, связанных с текущей страницей.

1	2	3
getChildPages	public	Возвращает массив объектов Page, представляющих дочерние страницы.

4.1.7 Спецификация класса Post

Данный класс представляет модель поста. Этот класс наследует функциональность класса ContentEntity и содержит методы для управления категориями постов. В таблице 4.13 приведена спецификация свойств класса Post.

Таблица 4.13 – Спецификация свойств класса Post

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
content	public	string	Содержимое поста.
author_id	public	int	Идентификатор автора поста.
author	public	User	Объект автора поста.
updated_datetime	public	Date	Дата и время последнего обновления поста.
categories	public	array	Список категорий, к которым относится пост.

В таблице 4.14 приведена спецификация методов класса Post.

Таблица 4.14 – Спецификация методов класса Post

Имя метода	Область видимости	Описание
1	2	3
updatePostCategories	private	Обновляет категории поста в базе данных. Удаляет текущие категории и добавляет новые.
getCategories	public	Возвращает массив объектов Category, к которым относится пост.
getCategoryIds	public	Возвращает массив идентификаторов категорий, к которым относится пост.
getPostsByCategory	private	Возвращает массив объектов Post, которые относятся к указанной категории.

4.1.8 Спецификация класса Category

Данный класс представляет модель категории. Этот класс наследует функциональность класса ContentEntity и содержит основные методы для управления категориями постов. В таблице 4.15 приведена спецификация свойств класса Category.

Таблица 4.15 – Спецификация свойств класса Category

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
parent_category_id	public	int	Идентификатор родительской категории.

В таблице 4.16 приведена спецификация методов класса Category.

Таблица 4.16 – Спецификация методов класса Category

Имя метода	Область видимости	Описание
1	2	3
getChildCategories	public	Возвращает массив объектов дочерних категорий.

4.1.9 Спецификация класса Menu

Данный класс представляет модель меню. Этот класс наследует функциональность класса Entity и содержит основные методы для управления областями меню и пунктами меню. В таблице 4.17 приведена спецификация свойств класса Menu.

Таблица 4.17 – Спецификация свойств класса Category

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
name	public	string	Название области меню.

В таблице 4.16 приведена спецификация методов класса Menu.

Таблица 4.18 – Спецификация методов класса Menu

Имя метода	Область видимости	Описание
1	2	3
getMenuItems	public	Возвращает массив объектов элементов меню.
getAddedPageIds	public	Возвращает массив идентификаторов страниц, которые были добавлены в меню.

1	2	3
getNextOrderNum	private	Возвращает следующий порядковый номер для нового элемента меню в текущем меню.
addMenuItem	public	Добавляет новый элемент меню в текущее меню.
updateMenuItemsOrder	public	Обновляет порядок элементов меню согласно новому порядку элементов.

4.1.10 Спецификация класса Route

Данный класс представляет модель маршрут. Он связывает URL-адреса с методами контроллеров. В таблице 4.19 приведена спецификация свойств класса Route.

Таблица 4.19 – Спецификация свойств класса Route

Имя свойства	Область видимости	Тип данных	Описание
1	2	3	4
path	public	string	URL-адрес.
controller	public	string	Название контроллера связанного с маршрутом.
action	public	string	Метод контроллера связанного с маршрутом.
page_id	public	int	Идентификатор связанной страницы.
post_id	public	int	Идентификатор связанного поста.
category_id	public	int	Идентификатор связанной категории.
tag_id	public	int	Идентификатор связанного тега.

В таблице 4.20 приведена спецификация методов класса Route.

Таблица 4.20 – Спецификация методов класса Route

Имя метода	Область видимости	Описание
1	2	3
add	public static	Создает новый маршрут в базе данных, используя переданные данные.
updatePath	public	Обновляет поле path маршрута и сохраняет изменения в БД.

4.1.11 Спецификация класса PageController

Данный класс предназначен для управления страницами в административной панели. В таблице 4.21 приведена спецификация методов класса PageController.

Таблица 4.21 – Спецификация методов класса PageController

Имя метода	Область видимости	Описание
1	2	3
defaultAction	public	Отображает список страниц в административной панели. Получает все страницы из БД и передает их в представление для отображения.
editPageAction	public	Обрабатывает обновление страницы. Загружает данные страницы, и отображает форму редактирования.

1	2	3
addPageAction	public	Обрабатывает добавление новой страницы. Отображает форму для добавления новой страницы.
deletePageAction	public	Обрабатывает удаление страницы.

4.1.12 Спецификация класса PostController

Данный класс предназначен для управления постами в административной панели. В таблице 4.22 приведена спецификация методов класса PostController.

Таблица 4.22 – Спецификация методов класса PostController

Имя метода	Область видимости	Описание
1	2	3
defaultAction	public	Отображает список постов в административной панели. Получает все посты из БД и передает их в представление для отображения.
editPostAction	public	Обрабатывает обновление поста. Загружает данные поста, и отображает форму редактирования.
addPostAction	public	Обрабатывает добавление нового поста. Отображает форму для добавления нового поста.
deletePostAction	public	Обрабатывает удаление поста.

4.1.13 Спецификация класса MenuController

Данный класс предназначен для управления областями и пунктами меню. В таблице 4.23 приведена спецификация методов класса MenuController.

Таблица 4.23 – Спецификация методов класса MenuController

Имя метода	Область видимости	Описание
1	2	3
defaultAction	public	Отображает список областей меню в административной панели. Получает все меню из БД и передает их в представление для отображения.
editMenuAction	public	Обрабатывает редактирование области меню. Загружает данные меню и его элементов, и отображает их в виде списка.
editMenuItemAction	public	Обрабатывает обновление элемента меню меню. Загружает данные элемента меню, и отображает форму редактирования.
addMenuItemAction	public	Обрабатывает добавление нового элемента меню. Отображает форму для добавления элемента меню.
deleteMenuItemAction	public	Обрабатывает удаление элемента меню.
updateMenuOrderAction	public	Обновляет порядок элементов меню.

4.1.14 Спецификация класса UserController

Данный класс предназначен для управления пользователями. В таблице 4.24 приведена спецификация методов класса UserController.

Таблица 4.24 – Спецификация методов класса MenuController

Имя метода	Область видимости	Описание
1	2	3
defaultAction	public	Отображает список пользователей в административной панели. Получает всех пользователей из БД и передает их в представление для отображения.
editUserAction	public	Обрабатывает редактирование информации о пользователе. Загружает данные о пользователе, и отображает форму редактирования.
addMenuItemAction	public	Обрабатывает добавление нового пользователя. Отображает форму для добавления пользователя.
deleteMenuItemAction	public	Обрабатывает удаление пользователя.

4.2 Тестирование программной системы

4.2.1 Модульное тестирование системы

Модульный тест для класса Entity представлен на рисунках 4.1 - 4.2.

```php
declare(strict_types=1);

use PHPUnit\Framework\TestCase;
use src\Entity;

class User extends Entity
{
  public string $name;
  public string $email;
  protected static string $tableName = 'users';

  protected function initFields(): void
  {
    $this->fields = ['name', 'email'];
  }
}

class EntityTest extends TestCase
{
  private $dbConn;

  protected function setUp(): void
  {
    $this->dbConn = new mysqli('localhost', 'root', '', 'test_db');

    $this->dbConn->query("
      CREATE TABLE IF NOT EXISTS users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(255) NOT NULL,
      email VARCHAR(255) NOT NULL
      )
    ");
  }

  protected function tearDown(): void
  {
    $this->dbConn->query("DROP TABLE IF EXISTS users");
    $this->dbConn->close();
  }

  public function testAdd(): void
  {
    $data = ['name' => 'John Doe', 'email' => 'john@example.com'];
    $user = User::add($this->dbConn, $data);

    $this->assertInstanceOf(User::class, $user);
    $this->assertEquals('John Doe', $user->name);
    $this->assertEquals('john@example.com', $user->email);
    $this->assertNotNull($user->id);
  }
```

Рисунок 4.1 – Модульный тест класса Entity

```
  public function testUpdate(): void
  {
    $data = ['name' => 'John Doe', 'email' => 'john@example.com'];
    $user = User::add($this->dbConn, $data);
    $user->update(['name' => 'Jane Doe']);

    $updatedUser = User::getByField($this->dbConn, 'id', $user->id);
    $this->assertEquals('Jane Doe', $updatedUser->name);
  }

  public function testGetAll(): void
  {
    User::add($this->dbConn, ['name' => 'John Doe', 'email' => 'john@example.
        com']);
    User::add($this->dbConn, ['name' => 'Jane Doe', 'email' => 'jane@example.
        com']);

    $users = User::getAll($this->dbConn);

    $this->assertCount(2, $users);
  }

  public function testGetByField(): void
  {
    $user = User::add($this->dbConn, ['name' => 'John Doe', 'email' => '
        john@example.com']);

    $retrievedUser = User::getByField($this->dbConn, 'email', 'john@example.
        com');
    $this->assertInstanceOf(User::class, $retrievedUser);
    $this->assertEquals('John Doe', $retrievedUser->name);
  }

  public function testDelete(): void
  {
    $user = User::add($this->dbConn, ['name' => 'John Doe', 'email' => '
        john@example.com']);
    $user->delete();

    $deletedUser = User::getByField($this->dbConn, 'id', $user->id);
    $this->assertNull($deletedUser);
  }
}
```

Рисунок 4.2 – Модульный тест класса Entity

Результаты модульного тестирования класса Entity представлены на рисунке 4.3.

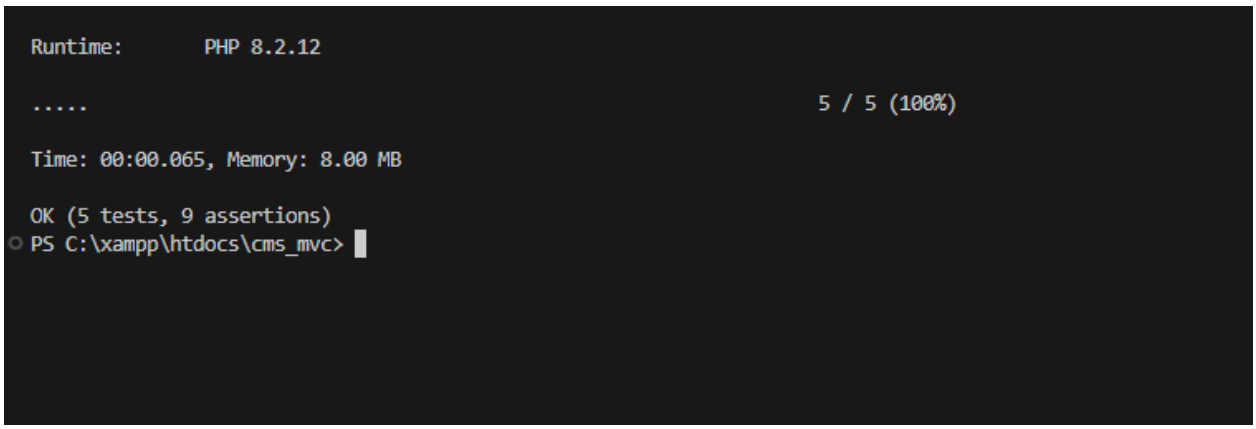


Рисунок 4.3 – Результаты модульного тестирования

4.2.2 Системное тестирование программной системы

Для проверки работоспособности разработанной программы было выполненно системное тестирование. Результаты тестирования представленны в виде снимков экрана.

На рисунке 4.4 представлена страница входа в административную панель сайта, которое открывается при запросе из адресной строки браузера. При успешной авторизации пользователь переходит на главную страницу панели и боковое меню для перемещения по соответствующим разделам панели.

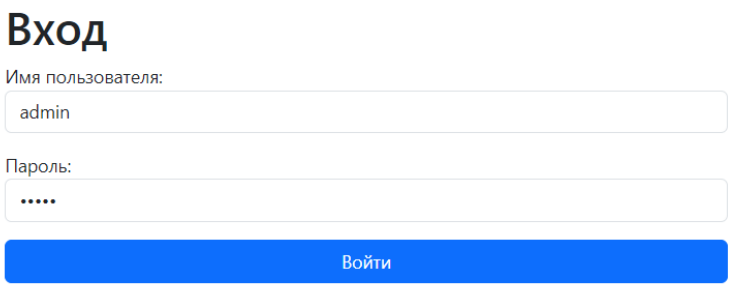


Рисунок 4.4 – Страница для входа в административную панель

Для того чтобы управлять страницами сайта, пользователю необходимо перейти в раздел «Страницы» (рис. 4.5), в котором отобразиться список страниц сайта.

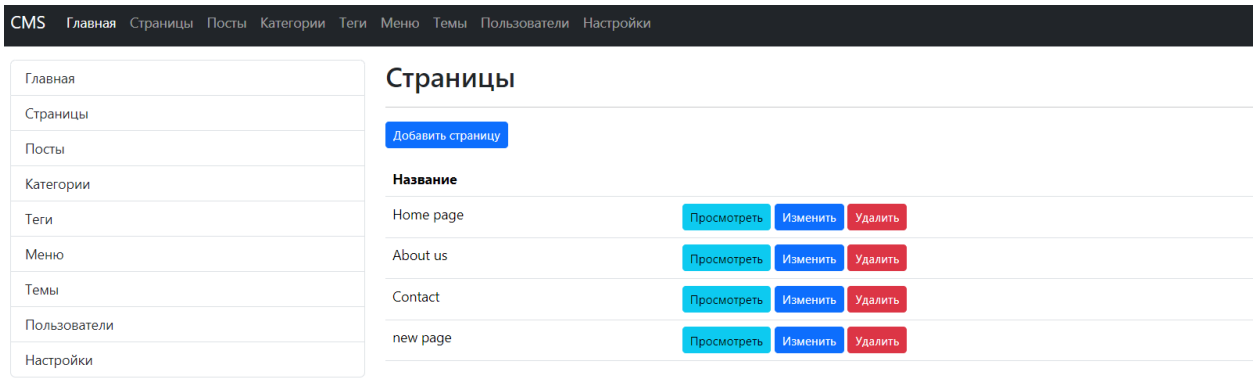


Рисунок 4.5 – Список страниц сайта

Для добавление новой страницы пользователь нажимает на кнопку «Добавить страницу», которая открывает форму добавления новой страницы (рис. 4.6). После ввода данных странцы, при нажатии на кнопку «Добавить», новая страница добаляется на сайт.

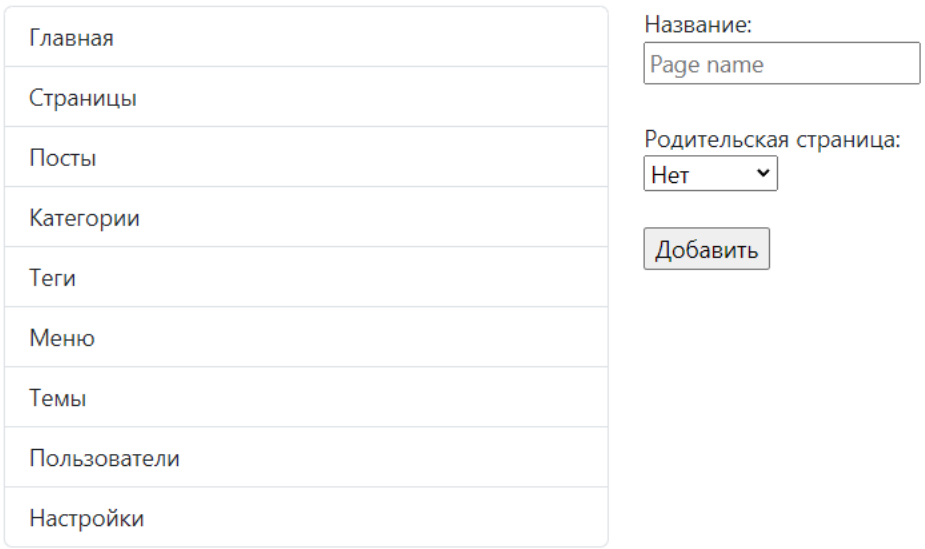


Рисунок 4.6 – Форма добавления страницы

Для редактирования страницы пользователь выбирает страницу из списка страниц и нажимает на кнопку «Редактировать», открывается редактор контента (рис. 4.7), в котором пользователь может вносить изменения в содержимое страницы, добавлять различные элементы интерфейса, форматировать текст и др. Для сохранения изменений пользователь нажимает на кнопку «Сохранить».

Рисунок 4.7 – Редактор контента

Внесенные изменения отображаюся на главной странице сайта (рис. 4.8).

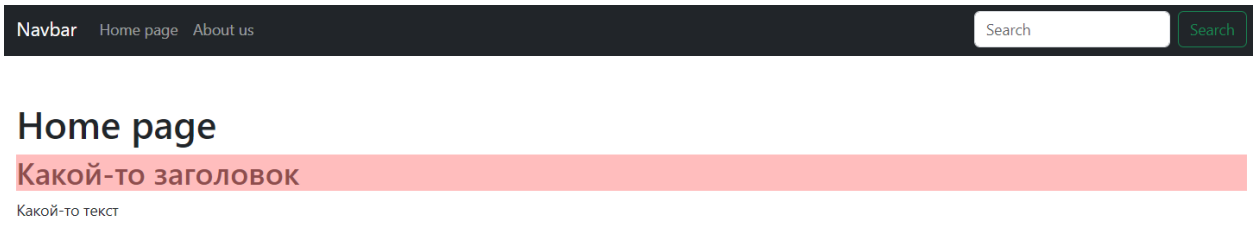


Рисунок 4.8 – Главная страница сайта

Для того чтобы управлять навигационными меню сайта, пользователю необходимо перейти в раздел «Меню» (рис. 4.9), в котором отобразиться список навигационных меню сайта.

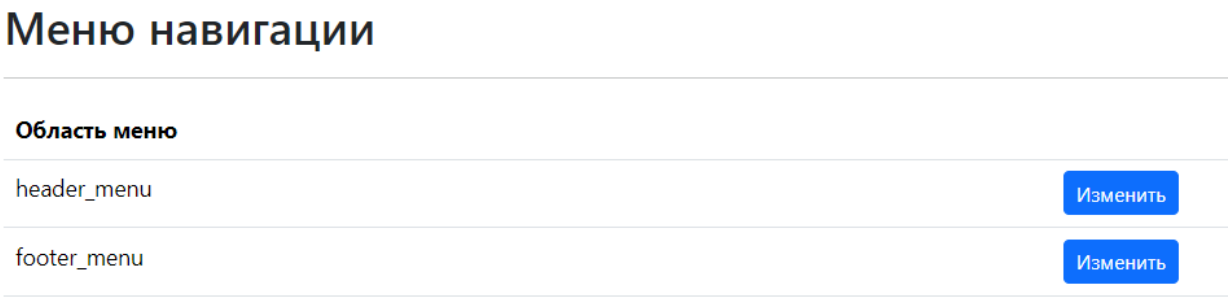


Рисунок 4.9 – Список областей меню сайта

Для добавление нового пункта меню, пользователь выбирает область меню, открывается страница со списком пунктов меню выбранной области меню (рис. 4.10), пользователь нажимает на кнопку «Добавить», появляется выпадающий список из опций: «Страница», «Произвольная ссылка».

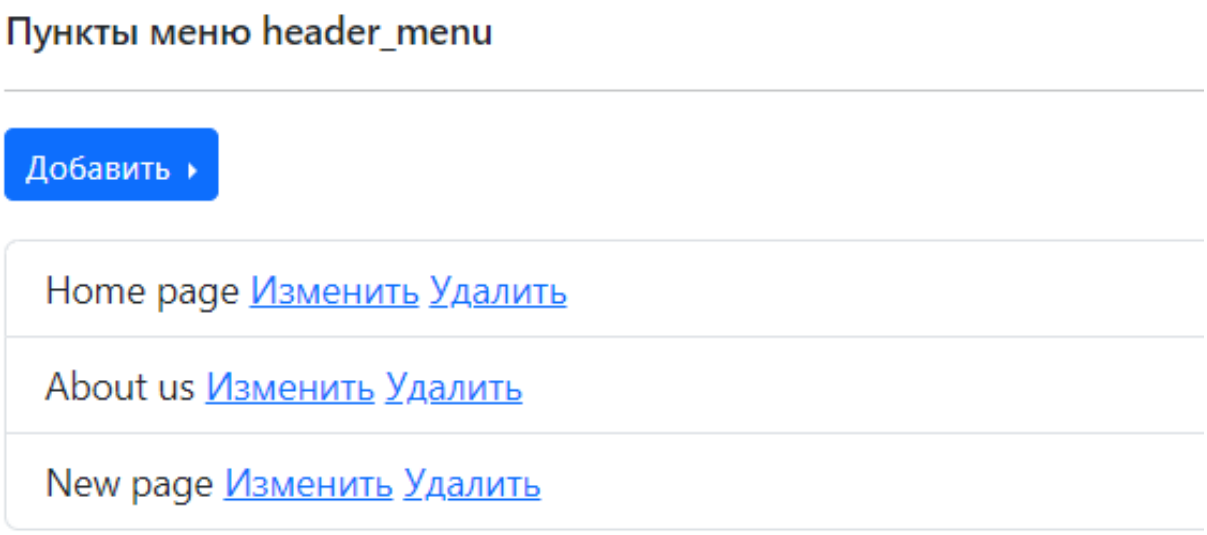


Рисунок 4.10 – Список пунктов меню

После выбора одной из опций появляется форма для добавления нового пункта меню (рис. 4.11).

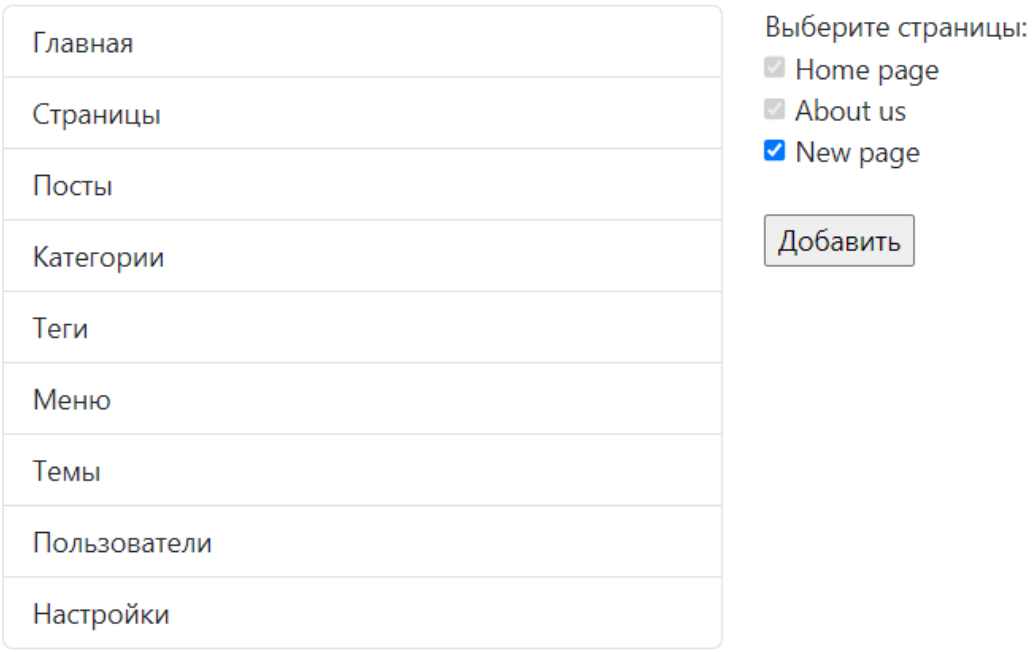


Рисунок 4.11 – Форма добавления нового пункта меню

После ввода данных пункта меню, при нажатии на кнопку «Добавить», новый пункт меню добавляется в меню и отображается в шапке сайта (рис. 4.12).

Рисунок 4.12 – Добавленный пункт меню

4.3 Сборка компонентов программной системы

Компоненты программно-информационной системы включают в себя файлы классов, скрипты, шаблоны страниц, ресурсы, библиотеки, конфигурационные файлы и прочие файлы, которые используются для функционирования системы.

Для компиляции и сборки всех программ, входящих в состав программно-информационной системы, необходимы установленный на компьютер HTTP-сервер и интерпритатор языка PHP.

Программная система запускается в браузере.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной работы была создана программно-информационная система для совместного управления содержимым сайта.

Разработанная система выполняет основные функции предоставляемые большинством существующих CMS, включая управления страницами, записями (постами), управление содержимым страниц и постов, категоризацию записей, управление пользователями, выбор темы сайта, выбор шаблона отображения страниц.

Основные результаты работы:

1. Проведен анализ предметной области. Определены перспективы и ключевые направления разработки программной системы.

2. Разработана концептуальная модель системы. Разработана модель данных системы. Определены требования к системе.

3. Осуществлено проектирование системы. Разработана база данных. Разработана архитектура серверной части. Разработан пользовательский интерфейс административной панели системы.

4. Проведено модульное и системное тестирование программной системы.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Перспективой дальнейшей разработки является расширение функционала системы: добавление новых модулей, добавления новых функций в редактор контента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фримен, А. Практикум по программированию на JavaScript / А. Фримен. – Москва : Вильямс, 2013. – 960 с. – ISBN 978-5-8459-1799-7. – Текст : непосредственный.

2. Фаулер, М. UML. Основы / М. Фаулер ; пер. с англ. А. Петухова. - 3-е изд. – Санкт-Петербург : Символ-Плюс, 2004. – 192 с. – ISBN 5-92086-060-X. – Текст : непосредственный

3. Бретт, М. PHP и MySQL. Исчерпывающее руководство / М. Бретт. – Санкт-Петербург : Питер, 2016. – 544 с. – ISBN 978-5-496-01049-8. – Текст : непосредственный.

4. Веру, Л. Секреты CSS. Идеальные решения ежедневных задач / Л. Веру. – Санкт-Петербург : Питер, 2016. – 336 с. – ISBN 978-5-496-02082-4. – Текст : непосредственный.

5. Гизберт, Д. PHP и MySQL / Д. Гизберт. – Москва : НТ Пресс, 2013. – 320 с. – ISBN 978-5-477-01174-2. – Текст : непосредственный.

6. Голдстайн, А. HTML5 и CSS3 для всех / А. Голдстайн, Л. Лазарис, Э. Уэйл. – Москва : Вильямс, 2012. – 368 с. – ISBN 978-5-699-57580-0. – Текст : непосредственный.

7. Дэкетт, Д. HTML и CSS. Разработка и создание веб-сайтов / Д. Дэкетт. – Москва : Эксмо, 2014. – 480 с. – ISBN 978-5-699-64193-2. – Текст : непосредственный.

8. Макфарланд, Д. Большая книга CSS / Д. Макфарланд. – Санкт-Петербург : Питер, 2012. – 560 с. – ISBN 978-5-496-02080-0. – Текст : непосредственный.

9. Лоусон, Б. Изучаем HTML5. Библиотека специалиста / Б. Лоусон, Р. Шарп. – Санкт-Петербург : Питер, 2013 – 286 с. – ISBN 978-5-459-01156-2. – Текст : непосредственный.

10. Титтел, Э. HTML5 и CSS3 для чайников / Э. Титтел, К. Минник. – Москва : Вильямс, 2016 – 400 с. – ISBN 978-1-118-65720-1. – Текст : непосредственный.

11. Ральф, Дж. Приемы объектно–ориентированного проектирования. Паттерны проектирования / Дж. Ральф, Влиссидес Джон. – СПб.: Питер, 2016. – 366 с. – ISBN 978-5-459-01720-5. – Текст : непосредственный..

12. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/php/tutorial/ (дата обращения: 14.05.2024).

13. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/web/javascript/ (дата обращения: 14.05.2024).

14. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/sql/tutorial/ (дата обращения: 14.05.2024).

15. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/sql/mysql/ (дата обращения: 14.05.2024).

16. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/web/jquery/ (дата обращения: 14.05.2024).

17. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/web/html5/ (дата обращения: 14.05.2024).

18. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/css/ (дата обращения: 14.05.2024).

19. METANIT.COM – Сайт о программировании : образовательная платформа : сайт. Санкт-Петербург, 2024. – URL: https://metanit.com/paterns/ (дата обращения: 14.05.2024).

20. Developing Web Content Management Systems – from the Past to the Future : сайт. – URL: https://www.shs-conferences.org/articles/shsconf/pdf/2021/21/shsconf_icemt2021_05007.pdf (дата обращения: 14.05.2024).

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.8.

Сведения о ВКРБ

Минобрнауки России
Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛВАРИАТА

«Система управления содержимым веб-сайтов»

Руководитель ВКР
д.т.н., профессор
Серебровский Вадим Владимирович

Автор ВКР
студент группы ПО-01б
Украинцев Дмитрий Игоревич

				ВКРБ 2068443.09.03.04.24.018			
	Фамилия И. О.	Подпись	Дата		Лит.	Масса	Масштаб
Автор работы	Украинцев Д. И.			**Сведения о ВКРБ**			
Руководитель	Серебровский В. В.						
Нормоконтроль	Чапыгин А. А.				Лист 1		Листов 8
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

Рисунок А.1 – Сведения о ВКРБ

Цели и задачи разработки

Цель настоящей работы – разработка системы управления содержимым веб-сайтов. Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области;
- разработать концептуальную модель программно-информационной системы;
- спроектировать и реализовать серверную и клиентскую части программной системы средствами веб-технологий;
- провести тестирование работы программно-информационной системы.

				ВКРБ 2068443.09.03.04.24.018			
	Фамилия И. О.	Подпись	Дата		Лит.	Масса	Масштаб
Автор работы	Украинцев Д. И.			**Цели и задачи разработки**			
Руководитель	Серебровский В. В.						
Нормоконтроль	Чапыгин А. А.				Лист 2		Листов 8
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

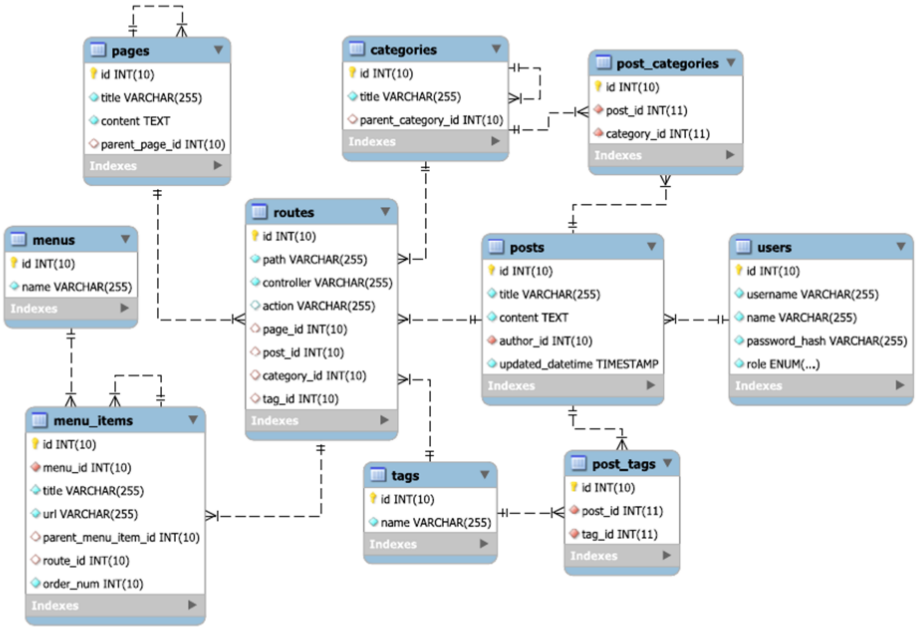
Рисунок А.2 – Цели и задачи разработки

Диаграмма прецедентов

				ВКРБ 2068443.09.03.04.24.018			
	Фамилия И. О.	Подпись	Дата		Лит.	Масса	Масштаб
Автор работы	Украинцев Д. И.			Диаграмма прецедентов			
Руководитель	Серебровский В. В.						
Нормоконтроль	Чапыгин А. А.				Лист 3	Листов 8	
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

Рисунок А.3 – Диаграмма прецедентов

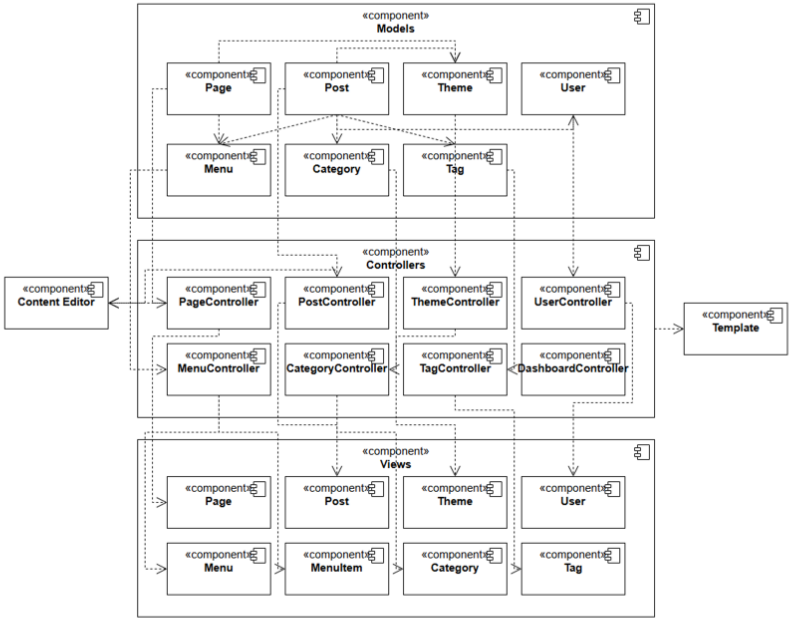
Реляционная модель данных



				ВКРБ 2068443.09.03.04.24.018			
	Фамилия И. О.	Подпись	Дата		Лит.	Масса	Масштаб
Автор работы	Украинцев Д. И.			Реляционная модель данных			
Руководитель	Серебровский В. В.				Лист 4	Листов 8	
Нормоконтроль	Чалыгин А. А.						
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

Рисунок А.4 – Реляционная модель данных

Диаграмма компонентов



				ВКРБ 2068443.09.03.04.24.018			
	Фамилия И. О.	Подпись	Дата		Лит.	Масса	Масштаб
Автор работы	Украинцев Д. И.			Диаграмма компонентов			
Руководитель	Серебровский В. В.						
Нормоконтроль	Чапыгин А. А.				Лист 5	Листов 8	
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

Рисунок А.5 – Диаграмма компонентов

Рисунок А.6 – Диаграмма классов (часть 1)

Рисунок А.7 – Диаграмма классов (часть 2)

Заключение

В процессе выполнения данной работы была создана программно-информационная система для совместного управления содержимым сайта.

Разработанная система выполняет основные функции предоставляемые большинством существующих CMS, включая управления страницами, записями (постами), управление содержимым страниц и постов, категоризацию записей, управление пользователями, выбор темы сайта, выбор шаблона отображения страниц.

Основные результаты работы:

1. Проведен анализ предметной области. Определены перспективы и ключевые направления разработки программной системы.
2. Разработана концептуальная модель системы. Разработана модель данных системы. Определены требования к системе.
3. Осуществлено проектирование системы. Разработана база данных. Разработана архитектура серверной части. Разработан пользовательский интерфейс административной панели системы.
4. Проведено модульное и системное тестирование программной системы.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Рисунок А.8 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

```php
<?php

namespace src;

use mysqli;

final class DatabaseConnection
{
  private static $instance = null;
  private static $connection;

  static function getInstance()
  {
    if (is_null(self::$instance)) {
      self::$instance = new DatabaseConnection();
    }
    return self::$instance;
  }

  static function connect($hostname, $user, $password, $dbName)
  {
    self::$connection = new mysqli($hostname, $user, $password, $dbName);
  }

  function getConnection()
  {
    return self::$connection;
  }

  private function __construct()
  {
  }

  private function __clone()
  {
  }
}

<?php

declare(strict_types=1);

namespace src;

use mysqli;

abstract class Entity
{
  public int $id;
  protected $dbConn;
```

```php
protected static string $tableName;
protected array $fields = [];
protected array $primaryKeys = ['id'];

abstract protected function initFields(): void;

protected function __construct($dbConn)
{
  $this->dbConn = $dbConn;
  $this->initFields();
}

public static function add($dbConn, $data): static
{
  $className = static::class;

  $object = new $className($dbConn);
  $object->setFieldValues((array) $data);
  $object->insert();

  return $object;
}

public function update(mixed $data): void
{
  $this->setFieldValues((array) $data);
  $this->save();
}

public static function getAll($dbConn, array $conditions = []): array
{
  return static::getObjects($dbConn, $conditions);
}

public static function getByField($dbConn, string $fieldName, $fieldValue):
    ?static
{
  $objects = static::getObjects($dbConn, [$fieldName => $fieldValue]);
  return $objects[0] ?? null;
}

protected function setFieldValues(array $data): void
{
  foreach ($this->primaryKeys as $keyName) {
    if (array_key_exists($keyName, $data)) {
      $this->$keyName = $data[$keyName];
    }
  }

  foreach ($this->fields as $fieldName) {
    if (array_key_exists($fieldName, $data)) {
      $this->$fieldName = $data[$fieldName];
    }
  }
```

```
104   }
105
106   protected function insert(): void
107   {
108     $parameterValues = [];
109     $parameterTypes = '';
110
111     foreach ($this->fields as $field) {
112       $parameterValues[] = $this->$field;
113       $parameterTypes .= self::getType($this->$field);
114     }
115
116     $fieldNames = implode(', ', $this->fields);
117     $parameters = implode(', ', array_fill(0, count($this->fields), '?'));
118
119     $sql = "INSERT INTO " . static::$tableName . " ($fieldNames) VALUES (
          $parameters)";
120
121     $stmt = $this->dbConn->prepare($sql);
122     $stmt->bind_param($parameterTypes, ...$parameterValues);
123     $stmt->execute();
124
125     $this->id = $this->dbConn->insert_id;
126
127     $stmt->close();
128   }
129
130   protected function save(): void
131   {
132     $fieldBindings = [];
133     $keyBindings = [];
134     $parameterValues = [];
135     $parameterTypes = '';
136
137     foreach ($this->fields as $field) {
138       $fieldBindings[] = $field . ' = ?';
139       $parameterValues[] = $this->$field;
140       $parameterTypes .= self::getType($this->$field);
141     }
142
143     foreach ($this->primaryKeys as $key) {
144       $keyBindings[] = $key . ' = ?';
145       $parameterValues[] = $this->$key;
146       $parameterTypes .= self::getType($this->$key);
147     }
148
149     $fieldBindingsString = implode(', ', $fieldBindings);
150     $keyBindingsString = implode(' AND ', $keyBindings);
151
152     $sql = "UPDATE " . static::$tableName . " SET $fieldBindingsString WHERE
          $keyBindingsString";
153
154     $stmt = $this->dbConn->prepare($sql);
155     $stmt->bind_param($parameterTypes, ...$parameterValues);
```

```
156     $stmt->execute();
157
158     $stmt->close();
159   }
160
161   public function delete(): void
162   {
163     $keyBindings = [];
164     $parameterValues = [];
165     $parameterTypes = '';
166
167     foreach ($this->primaryKeys as $key) {
168       $keyBindings[] = $key . ' = ?';
169       $parameterValues[] = $this->$key;
170       $parameterTypes .= self::getType($this->$key);
171     }
172
173     $keyBindingsString = join(' AND ', $keyBindings);
174
175     $sql = "DELETE FROM " . static::$tableName . " WHERE $keyBindingsString";
176
177     $stmt = $this->dbConn->prepare($sql);
178     $stmt->bind_param($parameterTypes, ...$parameterValues);
179     $stmt->execute();
180
181     $stmt->close();
182   }
183
184   private static function getData($dbConn, array $conditions = []): array
185   {
186     $sql = "SELECT * FROM " . static::$tableName;
187     $fieldBindings = [];
188     $types = '';
189     $values = [];
190
191     if ($conditions) {
192       foreach ($conditions as $fieldName => $fieldValue) {
193         $fieldBindings[] = "$fieldName = ?";
194         $types .= self::getType($fieldValue);
195         $values[] = $fieldValue;
196       }
197       $sql .= ' WHERE ' . implode(' AND ', $fieldBindings);
198     }
199
200     $stmt = $dbConn->prepare($sql);
201
202     if ($conditions) {
203       $stmt->bind_param($types, ...$values);
204     }
205
206     $stmt->execute();
207     $data = $stmt->get_result()->fetch_all(MYSQLI_ASSOC);
208
209     $stmt->close();
```

```
    return $data;
  }

  private static function getObjects($dbConn, array $conditions = []): array
  {
    $data = static::getData($dbConn, $conditions);
    $objects = [];

    if ($data) {
      $className = static::class;
      foreach ($data as $objectData) {
        $object = new $className($dbConn);
        $object->setFieldValues($objectData, true);
        $objects[] = $object;
      }
    }

    return $objects;
  }

  private static function getType($value): string
  {
    if (is_null($value)) {
      return 's';
    } elseif (is_int($value)) {
      return 'i';
    } elseif (is_float($value)) {
      return 'd';
    } elseif (is_string($value)) {
      return 's';
    } else {
      return 'b';
    }
  }
}

<?php

declare(strict_types=1);

namespace src;

abstract class ContentEntity extends Entity
{
  public string $title;
  public string $url;
  protected string $entityName;
  protected string $controllerName;
  protected ?string $updateAction = null;

  protected function initFields(): void
  {
    $this->fields = [
    'title'
```

```
264       ];
265     }
266
267     protected function setFieldValues(array $data, bool $fromDatabase = false):
            void
268     {
269       parent::setFieldValues($data);
270
271       if ($fromDatabase) {
272         $this->url = Route::getByField($this->dbConn, $this->entityName . '_id'
              , $this->id)->path;
273       }
274     }
275
276     public static function add($dbConn, $data): static
277     {
278       $object = parent::add($dbConn, $data);
279       $object->manageRoute();
280       $object->afterInsert($data);
281
282       return $object;
283     }
284
285     public function update($data): void
286     {
287       parent::update($data);
288       $this->manageRoute();
289       $this->afterUpdate($data);
290     }
291
292     protected function manageRoute(): void
293     {
294       $this->url = $this->generateUrl();
295       $route = Route::getByField($this->dbConn, $this->entityName . '_id',
            $this->id);
296
297       if ($route) {
298         $route->updatePath($this->url);
299       } else {
300         Route::add($this->dbConn, new RouteData($this->url, $this->
              controllerName, $this->entityName, $this->id, $this->updateAction));
301       }
302     }
303
304     protected function generateUrl(): string
305     {
306       $url = strtolower(str_replace(' ', '-', $this->title));
307
308       $parentUrl = $this->getParentUrlPath();
309       if ($parentUrl) {
310         $url = $parentUrl . '/' . $url;
311       } else {
312         $url = '/' . $this->entityName . '/' . $url;
313       }
```

```
314
315       return $url;
316     }
317
318     private function getParentUrlPath(): ?string
319     {
320       $parent = 'parent_' . $this->entityName . '_id';
321       if (property_exists($this, $parent) && $this->$parent !== null && $this->
             $parent != 1) {
322         return Route::getByField($this->dbConn, $this->entityName . '_id',
               $this->$parent)->path;
323       }
324       return null;
325     }
326
327     abstract protected function afterInsert($data): void;
328     abstract protected function afterUpdate($data): void;
329 }
330
331 <?php
332
333 declare(strict_types=1);
334
335 namespace src;
336
337 use modules\user\admin\models\User;
338
339 class Auth
340 {
341     function verifyLogin($dbConn, string $username, string $password): bool
342     {
343       $user = User::getByField($dbConn, 'username', $username);
344
345       if (!$user->id) {
346         return false;
347       }
348       if (!password_verify($password, $user->password_hash)) {
349         return false;
350       }
351
352       $_SESSION['current_user_id'] = $user->id;
353       $_SESSION['is_admin'] = $user->role == 'admin' ? true : false;
354       return true;
355     }
356 }
357
358 <?php
359
360 declare(strict_types=1);
361
362 namespace src;
363
364 class Route extends Entity
365 {
```

```
366   protected static string $tableName = 'routes';
367   public string $path;
368   public string $controller;
369   public ?string $action;
370   public ?int $page_id = null;
371   public ?int $post_id = null;
372   public ?int $category_id = null;
373   public ?int $tag_id = null;
374
375   protected function initFields(): void
376   {
377     $this->fields = [
378     'path',
379     'controller',
380     'action',
381     'page_id',
382     'post_id',
383     'category_id',
384     'tag_id'
385     ];
386   }
387
388   public static function add($dbConn, $data): static
389   {
390     $route = new Route($dbConn);
391     $route->setFieldValues([
392     'path' => $data->url,
393     'controller' => $data->controller,
394     $data->entity . '_id' => $data->entity_id,
395     'action' => $data->action
396     ]);
397     $route->insert();
398
399     return $route;
400   }
401
402   public function updatePath($newPath): void
403   {
404     $this->path = $newPath;
405     $this->save();
406   }
407 }
408
409 <?php
410
411 namespace modules\page\models;
412
413 use modules\menu\admin\models\MenuItem;
414 use src\ContentEntity;
415 use src\Route;
416
417 class Page extends ContentEntity
418 {
419   public function __construct($dbConn)
```

```
420   {
421     parent::__construct($dbConn);
422   }
423
424   protected static string $tableName = 'pages';
425   protected string $entityName = 'page';
426   protected string $controllerName = 'page';
427
428   public string $content;
429   public ?int $parent_page_id;
430
431   protected function initFields(): void
432   {
433     parent::initFields();
434     $this->fields[] = 'content';
435     $this->fields[] = 'parent_page_id';
436   }
437
438   protected function afterInsert($data): void
439   {
440   }
441
442   protected function afterUpdate($data): void
443   {
444     $this->updateMenuItems();
445
446     $childPages = $this->getChildPages();
447     foreach ($childPages as $page) {
448       $page->manageRoute();
449       $page->updateMenuItems();
450     }
451   }
452
453   private function updateMenuItems(): void
454   {
455     $route = Route::getByField($this->dbConn, 'page_id', $this->id);
456     $menuItem = MenuItem::getByField($this->dbConn, 'route_id', $route->id);
457
458     if ($menuItem) {
459       $menuItem->updateMenuItemUrl($this->url);
460     }
461   }
462
463   public function getChildPages(): array
464   {
465     return Page::getAll($this->dbConn, ['parent_page_id' => $this->id]);
466   }
467 }
468
469 <?php
470
471 namespace modules\post\models;
472
473 use modules\category\models\Category;
```

```php
use modules\post\models\PostCategories;
use modules\user\admin\models\User;
use mysqli;
use src\ContentEntity;

class Post extends ContentEntity
{
  protected static string $tableName = 'posts';
  protected string $entityName = 'post';
  protected string $controllerName = 'post';

  public string $content;
  public int $author_id;
  public User $author;
  public $updated_datetime;
  public array $categories = [];

  protected function initFields(): void
  {
    parent::initFields();
    $this->fields = array_merge($this->fields, [
    'content',
    'author_id',
    'updated_datetime'
    ]);
  }

  protected function setFieldValues(array $data, bool $fromDatabase = false):
      void
  {
    parent::setFieldValues($data, $fromDatabase);

    if ($fromDatabase) {
      $this->categories = $this->getCategories();
      $this->author = User::getByField($this->dbConn, 'id', $this->author_id)
          ;
    }
  }

  protected function afterInsert($data): void
  {
    $this->updatePostCategories($data->categories);
  }


  protected function afterUpdate($data): void
  {
    $this->updatePostCategories($data->categories);
  }

  private function updatePostCategories(array $categories): void
  {
    $postId = $this->id;
```

```
525       $currentCategories = PostCategories::getAll($this->dbConn, ['post_id' =>
              $postId]);
526
527       foreach ($currentCategories as $category) {
528         $category->delete();
529       }
530
531       foreach ($categories as $categoryId) {
532         $data = (object)['post_id' => $postId, 'category_id' => (int)
              $categoryId];
533         PostCategories::add($this->dbConn, $data);
534       }
535
536       $this->categories = $this->getCategories();
537     }
538
539     public function getCategories(): array
540     {
541       $categoryIds = $this->getCategoryIds();
542
543       $categories = [];
544       foreach ($categoryIds as $categoryId) {
545         $category = Category::getByField($this->dbConn, 'id', $categoryId);
546         $categories[] = $category;
547       }
548
549       return $categories;
550     }
551
552     public function getCategoryIds(): array
553     {
554       $postCategories = PostCategories::getAll($this->dbConn, ['post_id' =>
              $this->id]);
555       return array_map(fn ($postCategory) => $postCategory->category_id,
              $postCategories);
556     }
557
558     public static function getPostsByCategory(mysqli $dbConn, int $categoryId):
          array
559     {
560       $postCategories = PostCategories::getAll($dbConn, ['category_id' =>
              $categoryId]);
561       $postIds = array_map(fn ($postCategory) => $postCategory->post_id,
              $postCategories);
562
563       $posts = [];
564       foreach ($postIds as $postId) {
565         $post = Post::getByField($dbConn, 'id', $postId);
566         $posts[] = $post;
567       }
568
569       return $posts;
570     }
571 }
```

```php

<?php

namespace modules\menu\admin\models;

use src\Entity;

class Menu extends Entity
{
  protected static string $tableName = 'menus';
  public string $name;

  protected function initFields(): void
  {
    $this->fields = [
    'name'
    ];
  }

  public function getMenuItems(): array
  {
    $menuItems = MenuItem::getAll($this->dbConn, ['menu_id' => $this->id]);

    usort($menuItems, fn($a, $b) => $a->order_num <=> $b->order_num);

    return $menuItems;
  }

  public function getAddedPageIds(): array
  {
    $sql = "SELECT page_id FROM routes INNER JOIN menu_items ON menu_items.
        route_id=routes.id WHERE menu_items.menu_id=?";
    // $sql = "SELECT page_id FROM routes INNER JOIN menu_items ON menu_items
        .url=routes.path WHERE menu_items.menu_id=$menu_id";
    $stmt = $this->dbConn->prepare($sql);
    $stmt->bind_param('i', $this->id);
    $stmt->execute();
    $result = $stmt->get_result();

    $addedPageIds = [];
    while ($row = $result->fetch_assoc()) {
      $addedPageIds[] = $row['page_id'];
    }

    return $addedPageIds;
  }

  private function getNextOrderNum(): int
  {
    $sql = "SELECT MAX(order_num) AS max_order_num FROM menu_items WHERE
        menu_id=?";
    $stmt = $this->dbConn->prepare($sql);
    $stmt->bind_param('i', $this->id);
    $stmt->execute();
```

```
623       $result = $stmt->get_result();
624       $row = $result->fetch_assoc();
625
626       $maxOrderNum = $row['max_order_num'];
627
628       return $maxOrderNum !== null ? $maxOrderNum + 1 : 1;
629     }
630
631     public function addMenuItem(MenuItemData $data) : MenuItem
632     {
633       $menuItem = new MenuItem($this->dbConn);
634
635       $menuItem->setFieldValues([
636       'menu_id' => $this->id,
637       'title' => $data->title,
638       'url' => $data->url,
639       'order_num' => $this->getNextOrderNum(),
640       'route_id' => $data->route_id,
641       'parent_menu_item_id' => $data->parent_menu_item_id
642       ]);
643       $menuItem->insert();
644
645       return $menuItem;
646     }
647
648     public function updateMenuItemsOrder($newOrder)
649     {
650       foreach ($newOrder as $index => $itemId) {
651         $itemId = (int)$itemId;
652         $menu_item = MenuItem::getByField($this->dbConn, 'id', $itemId);
653         $menu_item->setFieldValues(['order_num' => $index + 1]);
654         $menu_item->save();
655       }
656     }
657 }
658
659 <?php
660
661 namespace modules\menu\admin\models;
662
663 use src\Entity;
664
665 class MenuItem extends Entity
666 {
667   protected static string $tableName = 'menu_items';
668   public int $menu_id;
669   public string $title;
670   public string $url;
671   public ?int $parent_menu_item_id;
672   public ?int $route_id;
673   public int $order_num;
674   public array $child_menu_items = [];
675
676   protected function initFields(): void
```

```
677 {
678   $this->fields = [
679   'menu_id',
680   'title',
681   'url',
682   'parent_menu_item_id',
683   'route_id',
684   'order_num'
685   ];
686 }
687
688 protected function setFieldValues(array $data, bool $fromDatabase = false):
        void
689 {
690   parent::setFieldValues($data);
691
692   if ($fromDatabase) {
693     $this->child_menu_items = $this->getChildMenuItems();
694   }
695 }
696
697 public function getChildMenuItems(): array
698 {
699   return MenuItem::getAll($this->dbConn, ['parent_menu_item_id' => $this->
        id]);
700 }
701
702 public function renderMenuItem()
703 {
704   $html = '<li class="list-group-item" id="' . $this->id . '">';
705   $html .= $this->title;
706   $html .= ' <a href="/admin/index.php?module=menu&action=editMenuItem&id='
        . $this->id . '">Изменить</a> ';
707   $html .= ' <a href="/admin/index.php?module=menu&action=deleteMenuItem&id
        =' . $this->id . '">Удалить</a> ';
708
709   if (!empty($this->child_menu_items)) {
710
711     $html .= '<button class="btn btn-sm btn-primary float-end" data-bs-
          toggle="collapse" data-bs-target="#submenu' . $this->id . '">
          Подробнее</button>';
712     $html .= '<ul class="menu-list list-group collapse" id="submenu' .
          $this->id . '">';
713
714     foreach ($this->child_menu_items as $child) {
715       $html .= $child->renderMenuItem();
716     }
717     $html .= '</ul>';
718   }
719   $html .= '</li>';
720
721   return $html;
722 }
723
```

```
724   public function update($data): void
725   {
726     $this->setFieldValues([
727     'title' => $data->title,
728     'url' => $data->url,
729     'parent_menu_item_id' => $data->parent_menu_item_id,
730     'route_id' => $data->route_id
731     ]);
732     $this->save();
733   }
734
735   public function updateMenuItemUrl(string $url): void
736   {
737     $this->url = $url;
738     $this->save();
739   }
740 }
741
742 <?php
743
744 namespace modules\category\models;
745
746 use src\ContentEntity;
747
748 class Category extends ContentEntity
749 {
750   protected static string $tableName = 'categories';
751   protected string $entityName = 'category';
752   protected string $controllerName = 'post';
753   protected ?string $updateAction = 'showByCategory';
754
755   public ?int $parent_category_id;
756
757   protected function initFields(): void
758   {
759     parent::initFields();
760     $this->fields[] = 'parent_category_id';
761   }
762
763   protected function afterInsert($data): void
764   {
765   }
766
767   protected function afterUpdate($data): void
768   {
769     $childCategories = $this->getChildCategories();
770     foreach ($childCategories as $category) {
771       $category->manageRoute();
772     }
773   }
774
775   public function getChildCategories(): array
776   {
```

```
777     return Category::getAll($this->dbConn, ['parent_category_id' => $this->id
            ]);
778   }
779 }
780
781 <?php
782
783 namespace modules\dashboard\admin\controllers;
784
785 use src\Controller;
786 use src\Auth;
787
788 class DashboardController extends Controller
789 {
790   function defaultAction()
791   {
792     header('Location: /admin/index.php?module=page');
793     exit();
794   }
795
796   function loginAction()
797   {
798     if ($_POST['postAction'] ?? 0) {
799       $username = $_POST['username'] ?? '';
800       $password = $_POST['password'] ?? '';
801
802       $auth = new Auth();
803       if ($auth->verifyLogin($this->dbConn, $username, $password)) {
804         $_SESSION['logged_in'] = true;
805         header('Location: /admin/');
806         exit();
807       }
808       // var_dump($password);
809       $_SESSION['validation']['error'] = "Username or password is incorrect";
810     }
811
812     include VIEW_PATH . 'admin/login.php';
813     unset($_SESSION['validation']['error']);
814   }
815 }
816
817 <?php
818
819 declare(strict_types=1);
820
821 namespace modules\page\admin\controllers;
822
823 use modules\page\models\PageData;
824 use src\AdminController;
825 use modules\page\models\Page;
826
827 class PageController extends AdminController
828 {
829   function defaultAction(): void
```

```
{
  $data['pages'] = Page::getAll($this->dbConn);

  $this->template->renderView($data, 'page/admin/views/page_list');
}

function editPageAction(): void
{
  $pageId = $_GET['id'];
  $page = Page::getByField($this->dbConn, 'id', $pageId);

  if ($_POST['action'] ?? 0) {
    $page->update(new PageData(
    $_POST['title'],
    $_POST['content'],
    (int)$_POST['parent_page_id']
    ));
  }

  $data['page'] = $page;
  $data['pages'] = Page::getAll($this->dbConn);
  $data['child_pages'] = $page->getChildPages();

  $pages = Page::getAll($this->dbConn);
  $child_pages = $page->getChildPages();
  // $this->template->renderView($data, 'page/admin/views/page_edit');
  include VIEW_PATH . 'admin/content_editor.php';
}

function addPageAction(): void
{
  if ($_POST['action'] ?? 0) {
    Page::add($this->dbConn, new PageData(
    $_POST['title'],
    $_POST['content'],
    (int)$_POST['parent_page_id']
    ));

    header('Location: /admin/');
    exit();
  }

  $data['pages'] = Page::getAll($this->dbConn);
  $this->template->renderView($data, 'page/admin/views/page_add');
}

function deletePageAction(): void
{
  $pageId = $_GET['id'];
  $page = Page::getByField($this->dbConn, 'id', $pageId);
  $page->delete();

  header('Location: /admin/');
}
```

```
884 }
885
886 <?php
887
888 namespace modules\post\admin\controllers;
889
890 use modules\category\models\Category;
891 use modules\post\models\PostData;
892 use src\AdminController;
893 use modules\post\models\Post;
894
895 class PostController extends AdminController
896 {
897   function defaultAction(): void
898   {
899     $data['posts'] = Post::getAll($this->dbConn);
900
901     $this->template->renderView($data, 'post/admin/views/post_list');
902   }
903
904   function addPostAction(): void
905   {
906     if ($_POST['action'] ?? 0 == 1) {
907       Post::add($this->dbConn, new PostData(
908       $_POST['title'],
909       $_POST['content'],
910       $_SESSION['current_user_id'],
911       $_POST['categories'] ?? [1]
912       ));
913
914       header('Location: /admin/index.php?module=post');
915       exit();
916     }
917
918     $categories = Category::getAll($this->dbConn);
919     array_shift($categories);
920
921     $data['categories'] = $categories;
922     $this->template->renderView($data, 'post/admin/views/post_add');
923   }
924
925   function editPostAction(): void
926   {
927     $postId = $_GET['id'];
928     $post = Post::getByField($this->dbConn, 'id', $postId);
929
930     if ($_POST['action'] ?? 0) {
931       $post->update(new PostData(
932       $_POST['title'],
933       $_POST['content'],
934       $_SESSION['current_user_id'],
935       $_POST['categories'] ?? [1]
936       ));
937     }
```

```
938
939       $data['post'] = $post;
940       $data['post_categories'] = $post->getCategoryIds();
941
942       $categories = Category::getAll($this->dbConn);
943       array_shift($categories);
944
945       $data['categories'] = $categories;
946       // $page = $pageObj;
947       // $pages = $pageObj->getAll();
948       // $child_pages = $pageObj->getChildPages();
949       $this->template->renderView($data, 'post/admin/views/post_edit');
950       // include VIEW_PATH . 'admin/content_editor.php';
951     }
952
953     function deletePostAction(): void
954     {
955       $postId = $_GET['id'];
956       $postObj = Post::getByField($this->dbConn, 'id', $postId);
957       $postObj->delete();
958
959       header('Location: /admin/index.php?module=post');
960     }
961   }
962
963   <?php
964
965   declare(strict_types=1);
966
967   namespace modules\menu\admin\controllers;
968
969   use modules\menu\admin\models\Menu;
970   use src\AdminController;;
971   use modules\menu\admin\models\MenuItem;
972   use modules\menu\admin\models\MenuItemData;
973   use modules\page\models\Page;
974   use src\Route;
975
976   class MenuController extends AdminController
977   {
978     function defaultAction(): void
979     {
980       $data['menus'] = Menu::getAll($this->dbConn);
981
982       $this->template->renderView($data, 'menu/admin/views/menu_list');
983     }
984
985     function editMenuAction(): void
986     {
987       $menuId = $_GET['id'];
988
989       $menu = Menu::getByField($this->dbConn, 'id', $menuId);
990
991       $data['menu'] = $menu;
```

```
992     $data['menu_items'] = $menu->getMenuItems();
993
994     $this->template->renderView($data, 'menu/admin/views/menu_edit');
995   }
996
997   function editMenuItemAction(): void
998   {
999     $menuItemId = $_GET['id'];
1000
1001    $menuItem = MenuItem::getByField($this->dbConn, 'id', $menuItemId);
1002
1003    if ($_POST['action'] ?? 0) {
1004
1005      $parentMenuItemId = (int)$_POST['parent_menu_item_id'];
1006      if ($parentMenuItemId === 0) {
1007        $parentMenuItemId = null;
1008      }
1009      $menuItem->update(new MenuItemData($_POST['title'], $menuItem->url,
           $parentMenuItemId));
1010    }
1011
1012    $data['menu_item'] = $menuItem;
1013    $data['menu_items'] = MenuItem::getAll($this->dbConn, ['menu_id' =>
         $menuItem->menu_id]);
1014
1015    $this->template->renderView($data, 'menu/admin/views/menu_item_edit');
1016  }
1017
1018  function addMenuItemAction(): void
1019  {
1020    $menuId = $_GET['id'];
1021
1022    $menu = Menu::getByField($this->dbConn, 'id', $menuId);
1023
1024    if ($_POST['action'] ?? 0) {
1025
1026      if ($_GET['type'] == 'page') {
1027        foreach ($_POST['page_ids'] as $pageId) {
1028          $page = Page::getByField($this->dbConn, 'id', $pageId);
1029          $route = Route::getByField($this->dbConn, 'page_id', $pageId);
1030
1031          $menu->addMenuItem(new MenuItemData(
1032          $page->title,
1033          $page->url,
1034          route_id: $route->id
1035          ));
1036        }
1037      } else if ($_GET['type'] == 'link') {
1038        $parentMenuItemId = (int)$_POST['parent_menu_item_id'];
1039        if ($parentMenuItemId === 0) {
1040          $parentMenuItemId = null;
1041        }
1042        $menu->addMenuItem(new MenuItemData(
1043        $_POST['title'],
```

```
1044            $_POST['url'],
1045            $parentMenuItemId
1046            ));
1047        }
1048        header("Location: /admin/index.php?module=menu&action=editMenu&id=
                $menuId");
1049        exit();
1050      }
1051
1052      $data['pages'] = Page::getAll($this->dbConn);
1053      $data['menu_items'] = $menu->getMenuItems();
1054      $data['added_page_ids'] = $menu->getAddedPageIds();
1055
1056      if ($_GET['type'] == 'page') {
1057        $this->template->renderView($data, 'menu/admin/views/menu_item_add_page
                ');
1058      } else if ($_GET['type'] == 'link') {
1059        $this->template->renderView($data, 'menu/admin/views/menu_item_add');
1060      }
1061    }
1062
1063    function deleteMenuItemAction(): void
1064    {
1065      $menuItemId = $_GET['id'];
1066      $menuItem = MenuItem::getByField($this->dbConn, 'id', $menuItemId);
1067      $menuItem->delete();
1068
1069      header("Location: /admin/index.php?module=menu&action=editMenu&id=
              $menuItem->menu_id");
1070    }
1071
1072    function updateMenuOrderAction(): void
1073    {
1074      $newOrder = $_POST['order'];
1075      $menu = new Menu($this->dbConn);
1076      $menu->updateMenuItemsOrder($newOrder);
1077    }
1078  }
1079
1080  <?php
1081
1082  namespace modules\category\admin\controllers;
1083
1084  use modules\category\models\Category;
1085  use modules\category\models\CategoryData;
1086  use src\AdminController;
1087
1088  class CategoryController extends AdminController
1089  {
1090    function defaultAction(): void
1091    {
1092      $categories = Category::getAll($this->dbConn);
1093      array_shift($categories);
1094      $data['categories'] = $categories;
```

```
1095
1096      $this->template->renderView($data, 'category/admin/views/category_list');
1097    }
1098
1099    function editCategoryAction(): void
1100    {
1101      $categoryId = $_GET['id'];
1102      $category = Category::getByField($this->dbConn, 'id', $categoryId);
1103
1104      if ($_POST['action'] ?? 0) {
1105
1106        if (empty($_POST['parent_category_id'])) {
1107          unset($_POST['parent_category_id']);
1108        }
1109
1110        $category->update(new CategoryData(
1111        $_POST['title'],
1112        $_POST['parent_category_id'] ?? null
1113        ));
1114      }
1115
1116      $data['category'] = $category;
1117      $data['child_categories'] = $category->getChildCategories();
1118      $categories = Category::getAll($this->dbConn);
1119      array_shift($categories);
1120      $data['categories'] = $categories;
1121      $this->template->renderView($data, 'category/admin/views/category_edit');
1122    }
1123
1124    function addCategoryAction(): void
1125    {
1126      if ($_POST['action'] ?? 0) {
1127
1128        if (empty($_POST['parent_category_id'])) {
1129          unset($_POST['parent_category_id']);
1130        }
1131
1132        Category::add($this->dbConn, new CategoryData(
1133        $_POST['title'],
1134        $_POST['parent_category_id'] ?? null
1135        ));
1136
1137        header('Location: /admin/index.php?module=category');
1138        exit();
1139      }
1140
1141      $categories = Category::getAll($this->dbConn);
1142      array_shift($categories);
1143      $data['categories'] = $categories;
1144      $this->template->renderView($data, 'category/admin/views/category_add');
1145    }
1146
1147    function deleteCategoryAction(): void
1148    {
```

```
1149      $categoryId = $_GET['id'];
1150      $category = Category::getByField($this->dbConn, 'id', $categoryId);
1151      $category->delete();
1152
1153      header('Location: /admin/index.php?module=category');
1154    }
1155  }
1156
1157  <?php
1158
1159  declare(strict_types=1);
1160  namespace src;
1161
1162  use modules\menu\admin\models\Menu;
1163  use modules\menu\admin\models\MenuItem;
1164  use mysqli;
1165
1166  class Template
1167  {
1168    private string $templatePath;
1169    private string $context;
1170    private mysqli $dbConn;
1171
1172    function __construct(mysqli $dbConn, string $templatePath, string $context
         = 'site')
1173    {
1174      $this->dbConn = $dbConn;
1175      $this->templatePath = $templatePath;
1176      $this->context = $context;
1177    }
1178
1179    function renderView(array $data, ?string $view = null, ?string
         $selected_theme = null) : void
1180    {
1181      extract($data);
1182
1183
1184      if ($this->context === 'admin'){
1185        include VIEW_PATH . $this->templatePath . ".php";
1186      }
1187      else{
1188        $menus = Menu::getAll($dbConn);
1189        // $index = array_search('header_menu', array_column($menus, 'name'));
1190        // $headerMenu = $index !== false ? $menus[$index] : null;
1191        include THEMES_PATH . $selected_theme . '/' . $this->templatePath . ".
             php";
1192      }
1193    }
1194  }
```

Место для диска