Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

ОТЧЁТ

по контрольной работе №1

по дисциплине

«Интернет-технологии и веб-программирование»

Вариант 15

|  |  |  |
| --- | --- | --- |
| Выполнил |  | ст. гр. 220604  А.И. Михнюк |
| Проверила |  | Н.В. Хаджинова |

Минск 2025

1. ЦЕЛЬ РАБОТЫ

Основной целью данной контрольной работы являлось практическое закрепление ключевых аспектов *back*-*end* разработки через создание динамических веб-приложений, взаимодействующих с базой данных. Работа была направлена на комплексное освоение всего жизненного цикла разработки - от проектирования структуры данных до развертывания готового решения в контейнеризированной среде.

В рамках работы предстояло разработать два взаимосвязанных, но функционально самостоятельных веб-приложения: базовую «Систему управления задачами» и специализированный «Каталог рецептов» по индивидуальному варианту №15. Такой подход позволил не только освоить типовые паттерны разработки, но и научиться адаптировать общую архитектуру под специфические требования предметной области.

Особое внимание уделялось проектированию реляционной структуры базы данных, где требовалось создать оптимальную схему хранения данных как для универсальных задач, так и для специализированных кулинарных рецептов с их уникальными атрибутами. Важной составляющей стала организация безопасного взаимодействия между *PHP*-скриптами и СУБД *MySQL* с использованием современных методов защиты от потенциальных уязвимостей.

Ключевой практической задачей являлась реализация полного цикла *CRUD*-операций, обеспечивающих базовый функционал управления данными. Это включало создание интуитивного пользовательского интерфейса для добавления, просмотра, редактирования и удаления записей, а также формирование динамических *HTML*-страниц на основе актуальных данных из базы.

Значимой частью работы стало освоение технологий контейнеризации приложений средствами *Docker*. Предстояло настроить изолированные среды выполнения для каждого приложения, обеспечив их корректное взаимодействие с базами данных и веб-серверами, что соответствует современным подходам к развертыванию веб-приложений в производственной среде.

Таким образом, работа охватывала весь спектр задач *back*-*end* разработчика - от проектирования данных и программирования бизнес-логики до обеспечения безопасности и деплоя готового решения, что способствовало формированию целостного понимания процесса создания веб-приложений.

1. ХОД РАБОТЫ
   1. Установка и настройка локального сервера, подготовка контейнеризированной среды

Первоначальная разработка веб-приложений осуществлялась в среде *Windows* с использованием установленного пакета *PHP*, интегрированного в системные переменные операционной системы. В качестве основной среды разработки была выбрана *IDE* *Visual* *Studio* *Code*, которая предоставляла широкие возможности для работы с *PHP*-кодом. Для организации локального сервера использовался плагин *PHP* *Server*, который позволял быстро запускать и тестировать приложения в процессе разработки. Этот подход обеспечивал простоту отладки и мгновенное применение изменений, однако имел ограничения при необходимости воспроизведения производственного окружения.

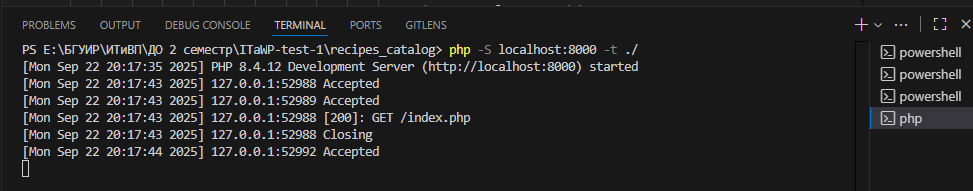


Рисунок 1 – Запуск *PHP* приложения через терминал

В ходе работы возникла необходимость обеспечения переносимости приложений и идентичности сред разработки и развертывания, что привело к решению о переходе на контейнеризированную среду *Docker*.

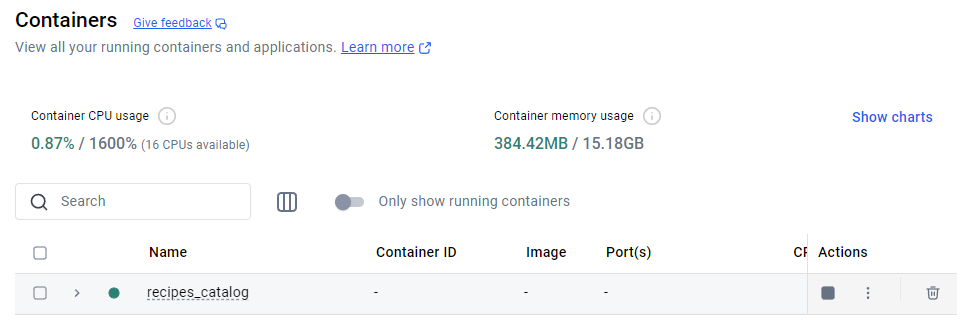
****

Рисунок 2 – Менеджер контейнеров *Docker* *Desktop*

Установка *Docker* *Desktop* позволила создать изолированные контейнеры для каждого компонента системы. Для описания инфраструктуры был разработан *docker*-*compose* файл, который определял взаимодействие между сервисами *PHP*-приложения, базы данных *MySQL* и административной панели *phpMyAdmin*.

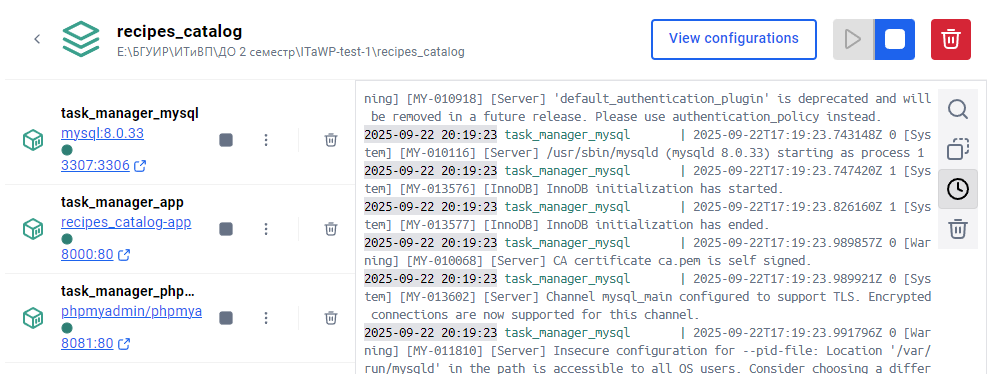


Рисунок 3 – Содержимое кластера в *Docker* *Desktop*

Конфигурация *docker*-*compose*.*yml* была тщательно продумана для обеспечения надежности работы системы. Для сервиса *MySQL* был выбран образ версии 8.0.33 с явным указанием метода аутентификации *native*\_*password* для совместимости с *PHP*-приложением. Особое внимание было уделено настройке *healthcheck* для базы данных - реализована периодическая проверка доступности *MySQL* через утилиту *mysqladmin* с интервалом 20 секунд и количеством попыток 10, что гарантировало корректный порядок запуска зависимых сервисов. Для обеспечения сохранности данных использовался *named* *volume* *mysql*\_*data*, который монтировался в стандартную директорию хранения данных *MySQL* /*var*/*lib*/*mysql*.

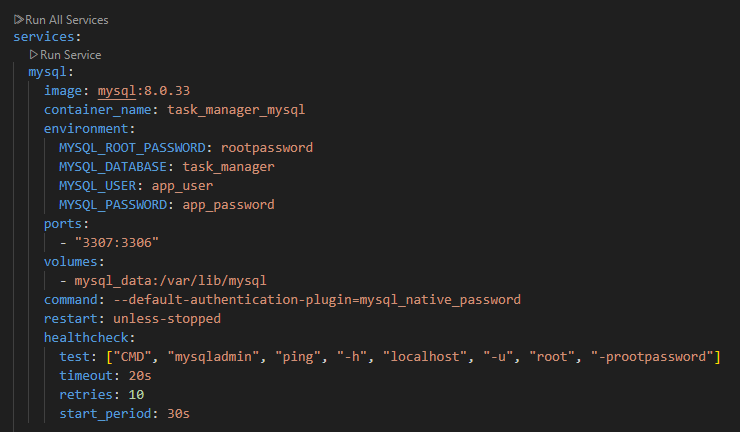


Рисунок 4 – Монтирование данных внутри *Docker* *Desktop*

Версия *PHP* 8.1 была выбрана как стабильная и поддерживающая все необходимые расширения для работы с базой данных. *Dockerfile* включал установку критически важных расширений *pdo*, *pdo*\_*mysql* и *mysqli*, без которых было невозможно организовать взаимодействие с СУБД. Особое значение имела активация модуля *rewrite* для *Apache* через команду *a*2*enmod* *rewrite*, что позволяло корректно обрабатывать *URL*-адреса в веб-приложении. Копирование файлов приложения в директорию /*var*/*www*/*html*/ и установка прав доступа для пользователя *www*-*data* обеспечивали правильное функционирование веб-сервера.

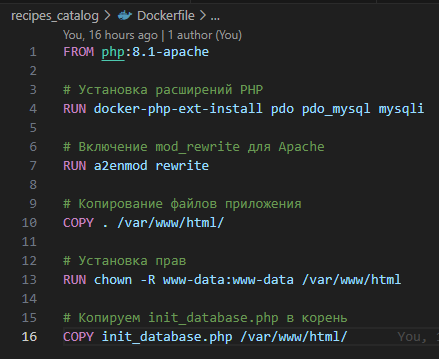


Рисунок 5 – Содержимое *Dockerfile*

Важным аспектом конфигурации стало решение проблемы занятых портов в локальной среде. Стандартный порт *MySQL* 3306 был переназначен на 3307 для приложения управления задачами, в то время как для каталога рецептов использовался порт 3306 в изолированном окружении. Аналогично были переназначены порты для *phpMyAdmin* (8081 и 8082) и веб-интерфейсов приложений (8000 и 8001), что позволило избежать конфликтов и обеспечить параллельную работу обоих проектов. Зависимости между сервисами были организованы через параметр *depends*\_*on* с условием *condition*: *service*\_*healthy*, что гарантировало запуск *PHP*-приложения только после полной инициализации базы данных.

Настройка политики перезапуска *restart*: *unless*-*stopped* обеспечивала автоматическое восстановление работы сервисов при нештатных ситуациях. В процессе настройки возникли сложности с монтированием томов в среде *Windows* из-за различий в файловых системах, что потребовало перехода на использование внутренних *docker* *volumes* вместо локального монтирования директорий. Это решение повысило стабильность работы и устранило проблемы с путями, характерные для *Windows*-окружения.

Для автоматизации процесса развертывания и устранения потенциальных проблем с кэшированием образов был разработан скрипт *reset*.*bat*, который выполнял полный цикл переустановки системы. Скрипт начинался с настройки кодовой страницы для корректного отображения кириллических символов, после чего последовательно останавливал и удалял все контейнеры и связанные тома. Команда *docker* *system* *prune* -*f* обеспечивала очистку системы от неиспользуемых образов, контейнеров и сетей, что предотвращало конфликты версий при повторном развертывании.

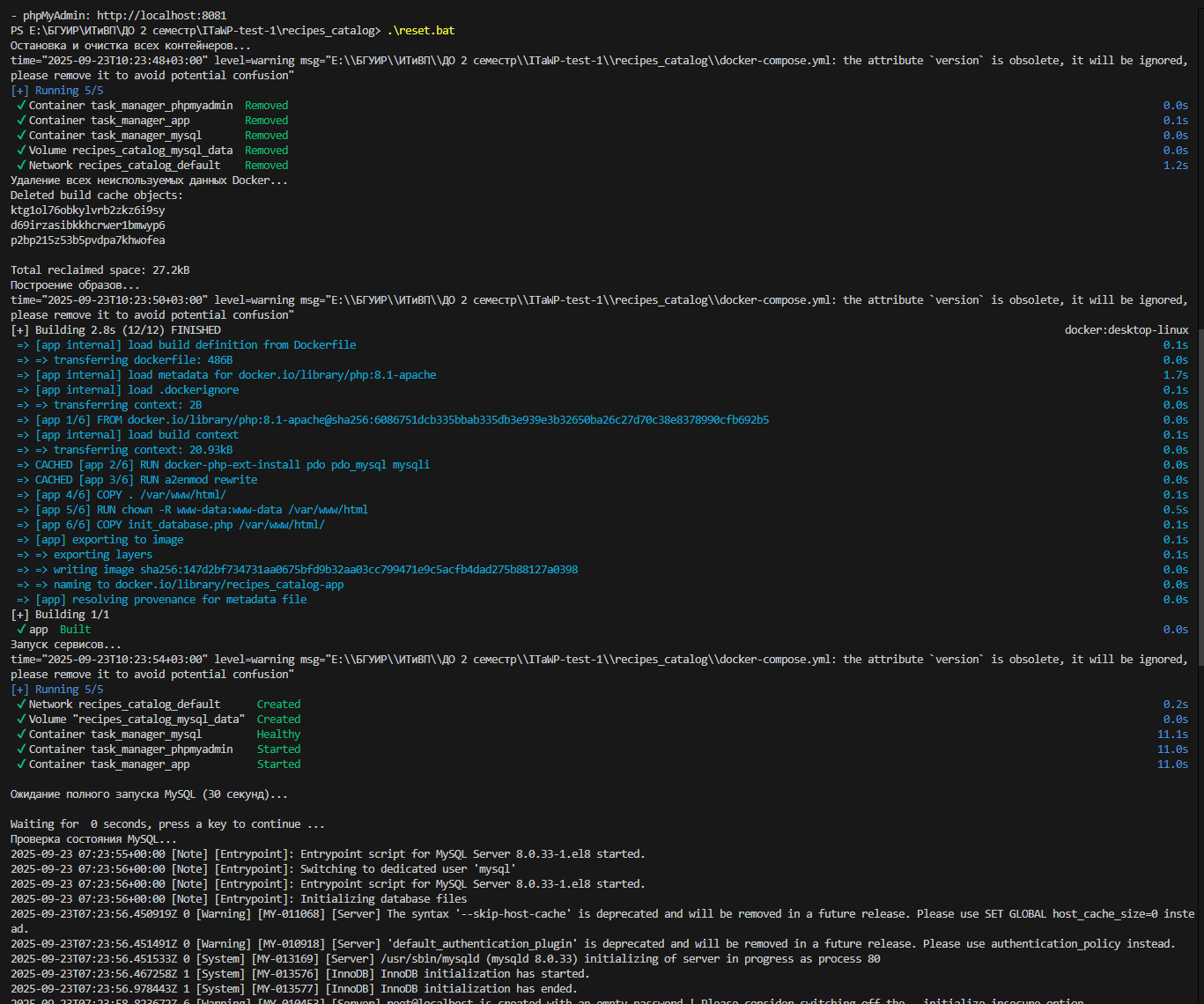


Рисунок 6 – Логи вызова *reset*.*bat*

После очистки окружения скрипт выполнял пересборку образов с учетом последних изменений в коде и запускал сервисы в фоновом режиме. Критически важным элементом была реализация паузы на 30 секунд для полной инициализации *MySQL*, так как база данных требовала времени для запуска и настройки внутренних процессов. После ожидания скрипт проверял состояние сервера *MySQL* через анализ логов контейнера и выполнял инициализацию базы данных с помощью скрипта *init*\_*database*.*php*, который запускался непосредственно внутри контейнера приложения.

Финальным этапом скрипта была комплексная проверка работоспособности системы, включающая тестирование подключения к базе данных через *PDO* и вывод информационных сообщений о доступности сервисов.

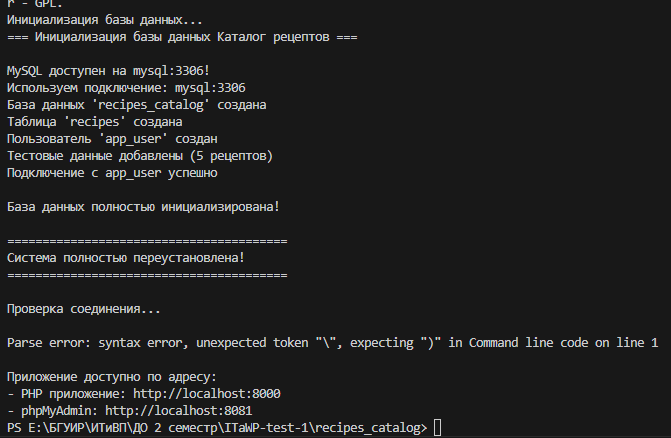


Рисунок 7 – Лог финального этапа запуска системы

Скрипт предоставлял пользователю четкие инструкции по доступу к приложениям, отображая актуальные адреса веб-интерфейсов и панели управления базами данных. Такой подход к автоматизации значительно сократил время развертывания и минимизировал вероятность ошибок, связанных с ручными операциями.

Преимущества контейнеризации проявились в значительном упрощении процесса развертывания и обеспечения идентичности сред. *Docker*-контейнеры позволили упаковать приложения со всеми зависимостями, исключив проблемы совместимости версий *PHP* и расширений. Использование *docker*-*compose* обеспечило оркестрацию многоконтейнерных приложений и автоматизацию запуска всей инфраструктуры одной командой. Изоляция сервисов в отдельных контейнерах повысила безопасность и стабильность работы, а также упростила масштабирование приложений в будущем.

2.2 Проектирование базы данных

Проектирование структуры базы данных являлось фундаментальным этапом разработки, определяющим эффективность и надежность работы веб-приложений. Для каждого из двух проектов была разработана оптимальная схема хранения данных, учитывающая специфику предметной области и функциональные требования.

В основу проектирования была положена концепция реляционной базы данных с использованием СУБД *MySQL*. Оба приложения использовали единый подход к организации данных, но с различной структурой таблиц, отражающей их предметную специфику.

Для каталога рецептов была создана таблица *recipes*.

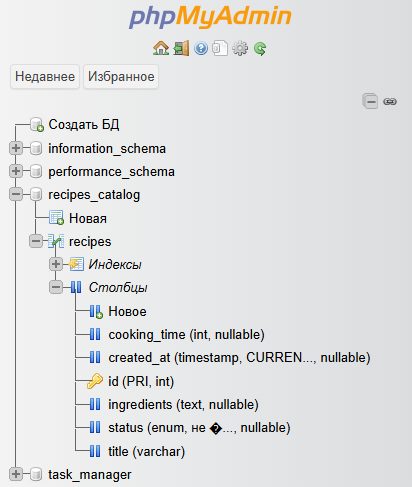
**

Рисунок *8 –* СхемаБД *recipes\_catalog*

В то время как для системы управления задачами - таблица *tasks*.

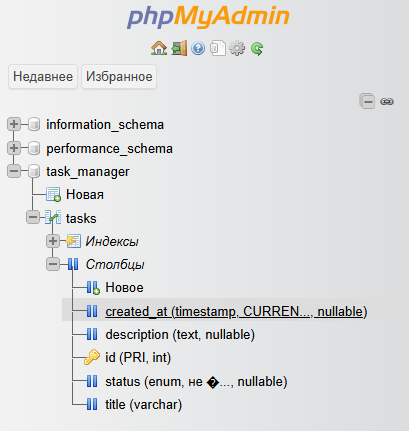


Рисунок *9 –* СхемаБД *task\_manager*

Важным аспектом проектирования стала реализация механизма отказоустойчивости при подключении к базе данных. В скриптах инициализации была применена функция *waitForMySQL*, которая осуществляла многократные попытки подключения с интервалом 5 секунд. Этот подход решал проблему асинхронного запуска контейнеров, когда *PHP*-приложение пыталось подключиться к *MySQL* до его полной готовности. Использование *PDO* с настройкой таймаута и режима исключений обеспечивало надежное установление соединения.

Для таблицы задач была разработана структура, включающая идентификатор, название, описание, статус выполнения и метку времени создания. Поле *status* использовало тип *ENUM* с двумя значениями - «не выполнена» и «выполнена», что обеспечивало целостность данных и упрощало их валидацию. Автоматическое заполнение поля *created*\_*at* текущим временем позволяло отслеживать хронологию создания задач без необходимости ручного ввода.

Специфика каталога рецептов потребовала адаптации базовой структуры под предметную область. Вместо поля *description* были введены два специализированных поля: *ingredients* для хранения списка ингредиентов в текстовом формате и *cooking*\_*time* для указания времени приготовления в минутах. Такой подход отражал особенности кулинарных рецептов и обеспечивал удобство работы с данными. Поле *cooking*\_*time* использовало целочисленный тип, что позволяло эффективно выполнять сортировку и фильтрацию по времени приготовления.

Обе таблицы использовали механизм автоматического инкремента для первичного ключа *id*, что гарантировало уникальность записей и упрощало их идентификацию. Выбор движка *InnoDB* обеспечивал поддержку транзакций и внешних ключей, хотя в текущей реализации последние не использовались из-за простоты структуры данных. Установка кодировки *utf*8*mb*4 и *collation* *utf*8*mb*4\_*unicode*\_*ci* позволяла корректно работать с кириллическими символами и специальными знаками.

Процесс инициализации базы данных включал не только создание структуры, но и наполнение тестовыми данными. Для каждого приложения были подготовлены репрезентативные наборы записей, демонстрирующие основные возможности системы. В каталоге рецептов это были популярные блюда с реальным временем приготовления и составами ингредиентов, в то время как в системе задач - типичные этапы учебного процесса.

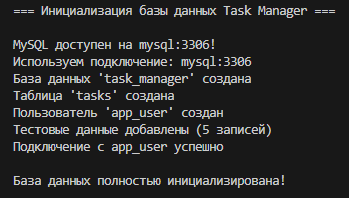


Рисунок 10 – Лог этапа инициализации структуры БД

Особое внимание было уделено безопасности доступа к данным. В скриптах инициализации создавался отдельный пользователь *app*\_*user* с ограниченными привилегиями, который использовался *PHP*-приложением для работы с базой данных. Это следовало принципу минимальных привилегий и снижало риски в случае компрометации приложения. Настройка прав доступа гарантировала, что приложение может выполнять только необходимые операции с данными.

Проверка существования базы данных и таблиц перед их созданием предотвращала ошибки при повторном запуске скриптов инициализации. Использование конструкций *CREATE* *DATABASE* *IF* *NOT* *EXISTS* и *CREATE* *TABLE* *IF* *NOT* *EXISTS* делало процесс идемпотентным и безопасным для многократного выполнения. Аналогично, проверка количества существующих записей перед добавлением тестовых данных исключала их дублирование.

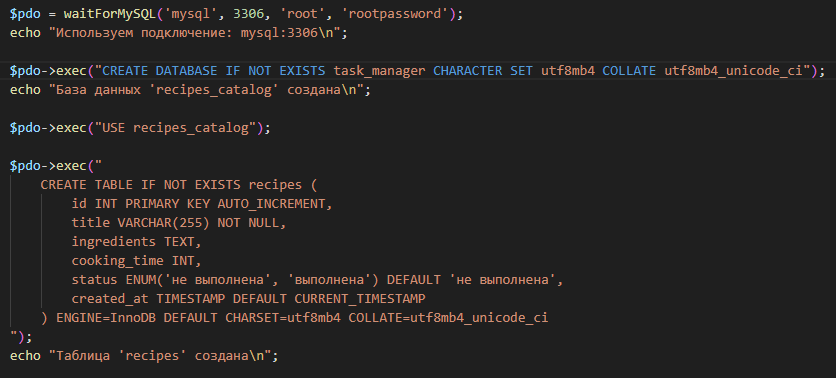


Рисунок 11 – Пример *SQL* скрипта в *PHP*

Завершающим этапом инициализации была комплексная проверка работоспособности системы, включающая тестовое подключение с учетными данными приложения. Это позволяло убедиться в корректности настройки прав доступа и готовности системы к работе. Такой многоуровневый подход к проектированию и инициализации базы данных обеспечивал надежную основу для функционирования веб-приложений и их дальнейшего развития.

2.3 Разработка структуры приложения, реализация функционала

Разработка структуры веб-приложения осуществлялась в строгом соответствии с техническим заданием, предусматривающим четкое разделение функциональности по отдельным модулям. Архитектура приложения была построена по принципу модульности, где каждый файл отвечал за конкретную задачу, что обеспечивало простоту поддержки и дальнейшего развития системы.

Основой приложения стал файл *config*.*php*, содержащий централизованные настройки подключения к базе данных. Использование *PDO* с настроенными параметрами обработки ошибок обеспечивало безопасное и надежное взаимодействие с *MySQL*. Этот подход позволял легко менять параметры подключения при переносе приложения между средами выполнения без необходимости правки кода в многочисленных файлах.

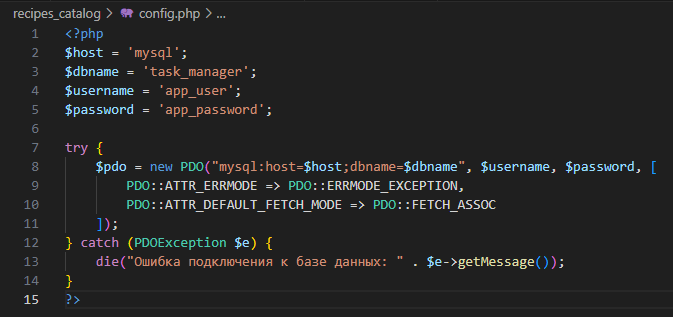


Рисунок 12 – Локальный *config.php*

Главная страница *index*.*php* выполняла функцию центрального узла приложения, обеспечивая отображение всех записей в структурированном виде. Для каталога рецептов была реализована расширенная таблица с колонками для ингредиентов и времени приготовления, в то время как система задач использовала более простой формат с описанием. Критически важным элементом стала обработка данных перед выводом - функция *htmlspecialchars* предотвращала *XSS*-атаки, а форматирование даты и времени средствами *PHP* улучшало восприятие информации пользователем.

Механизм добавления новых записей в *add*.*php* демонстрировал правильный подход к обработке пользовательского ввода. Форма использовала метод *POST* для передачи данных, а серверная часть выполняла валидацию обязательных полей перед сохранением в базу данных. Для каталога рецептов была реализована дополнительная проверка числового значения времени приготовления, что исключало некорректные данные на уровне приложения.

Страница редактирования *edit*.*php* совмещала в себе функции чтения и обновления данных. При открытии страницы выполнялся запрос к базе данных для получения текущих значений полей, которые затем использовались для предзаполнения формы. Это обеспечивало удобство работы пользователя с существующими записями. Особое внимание было уделено безопасности - идентификатор записи передавался через *GET*-параметр, но перед использованием обязательно проверялось его существование в базе данных.

Операции удаления и изменения статуса были вынесены в отдельные скрипты *delete*.*php* и *update*\_*status*.*php*, которые не имели графического интерфейса. Такой подход соответствовал принципу единой ответственности и упрощал логику приложения. Эти скрипты получали параметры через *GET*-запросы, выполняли соответствующую *SQL*-команду и *immediately* перенаправляли пользователя обратно на главную страницу, обеспечивая *seamless* *user* *experience*.

Визуальная часть приложения была построена на основе фреймворка *Bootstrap* 5, что гарантировало адаптивный и современный интерфейс. Для каждого приложения была разработана своя цветовая схема: система задач использовала синие акценты, а каталог рецептов - зеленые, что соответствовало их тематике. Компоненты *Bootstrap*, такие как *cards*, *tables* и *badges*, обеспечивали единообразие интерфейса и удобство работы на различных устройствах.

Особенностью реализации каталога рецептов стала адаптация интерфейса под специфику кулинарной тематики. Поле описания было заменено на поле ингредиентов с соответствующими *placeholder* и подсказками. Время приготовления выводилось в виде информационных *badges* с иконкой часов, что улучшало визуальное восприятие. Статусы рецептов были переименованы с «выполнена/не выполнена» на «проверен/на проверке», что больше соответствовало контексту *moderation* рецептов.

Система управления задачами сохранила классическую структуру, но была улучшена за счет лучшей организации кнопок действий. Группировка кнопок в *btn*-*group* обеспечивала компактное и интуитивно понятное управление. Условное отображение кнопки изменения статуса только для невыполненных задач снижало визуальный шум и предотвращало *unnecessary* действия.

Важным аспектом реализации стала обработка крайних случаев и ошибок. При отсутствии записей в базе данных выводилось соответствующее сообщение с предложением добавить первую запись. Проверка существования записи при редактировании предотвращала ошибки при неверных идентификаторах. Подтверждение удаления через *JavaScript* диалог снижало вероятность случайной потери данных.

Все *SQL*-запросы использовали подготовленные выражения, что полностью исключало возможность *SQL*-инъекций. Экранирование вывода данных средствами *htmlspecialchars* защищало от *XSS*-атак. Проверка *HTTP*-метолов для форм гарантировала, что обработка данных происходит только при корректных запросах. Эти меры безопасности делали приложение готовым к использованию в *production*-среде.

Интеграция между клиентской и серверной частями была организована через единый набор *endpoint*-ов, что обеспечивало четкое разделение *concerns*. Логика *redirection* после успешного выполнения операций поддерживала целостность *user* *flow* и предотвращала повторную отправку данных при обновлении страницы. Единообразная структура всех страниц упрощала навигацию и делала интерфейс *predictable* для пользователя.

2.4 Визуальное оформление и пользовательский интерфейс

Визуальное оформление приложений было разработано с учетом современных требований к веб-интерфейсам, обеспечивая не только эстетическую привлекательность, но и высокую функциональность. Основой дизайна стал фреймворк *Bootstrap* 5.3, который предоставил готовые компоненты и адаптивную сетку для создания интерфейсов, корректно отображающихся на устройствах с различными размерами экранов.

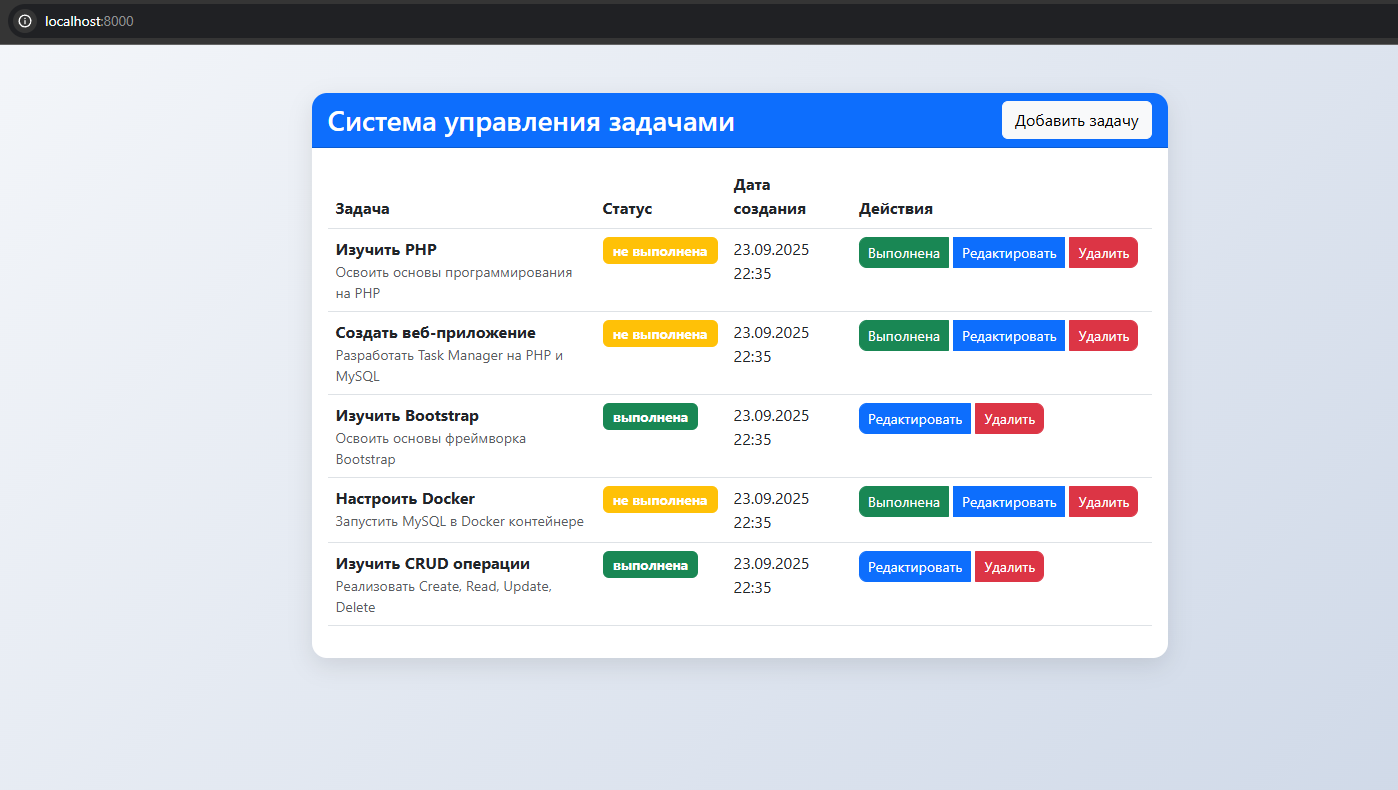


Рисунок 13 – Интерфейс *task*\_*manager*

Для каждого приложения была разработана уникальная визуальная идентичность, отражающая его предметную область. Система управления задачами получила синюю цветовую схему с градиентным фоном, создающим ощущение глубины и профессиональной серьезности. Градиент обеспечивал мягкий визуальный переход, снижающий нагрузку на глаза при длительной работе с приложением. Карточки интерфейса были оформлены с закругленными углами и тонкими тенями, что создавало эффект «парящих» элементов и улучшало визуальную иерархию.

Каталог рецептов использовал зеленую цветовую палитру, ассоциирующуюся со свежестью и натуральностью продуктов. Более сдержанный светло-серый фон подчеркивал кулинарную тематику и обеспечивал хорошую читаемость текста. В заголовке приложения была добавлена эмодзи-иконка, которая сразу визуально идентифицировала назначение приложения и создавала дружелюбную атмосферу.

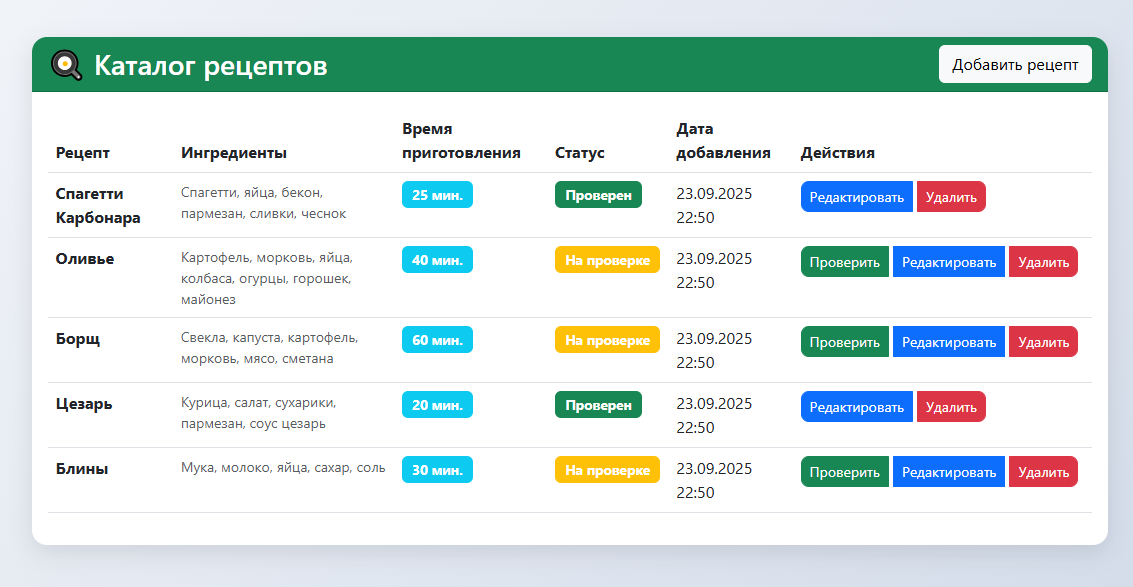


Рисунок 14 – Интерфейс *recipes\_catalog*

Ключевым элементом интерфейса стала адаптивная таблица с данными, которая оптимально использовала доступное пространство. Заголовки колонок были выделены полужирным шрифтом и лишены верхней границы, что создавало современный минималистичный вид. Для строк таблицы была реализована анимация при наведении - плавное изменение фона на светло-голубой оттенок в системе задач и более нейтральный эффект в каталоге рецептов. Это не только улучшало визуальное восприятие, но и помогало пользователю отслеживать текущую позицию в таблице.

Система статусов была визуализирована через цветные фигуры с семантическим кодированием. В системе задач статус «выполнена» отображался зеленым цветом, а «не выполнена» - оранжевым, что интуитивно понятно указывало на состояние задачи. В каталоге рецептов эти статусы были переосмыслены как «проверен» и «на проверке», но сохранили ту же цветовую схему для согласованности. Время приготовления в рецептах было выделено голубой фигурой с указанием единиц измерения, что позволяло быстро оценить временные затраты.

Формы ввода данных были организованы в центрированных карточках с четкой структурой полей. Поля имели соответствующую *HTML*5-валидацию. Для поля ингредиентов в каталоге рецептов был добавлен вариант заполнения с примером формата ввода, что снижало вероятность ошибок при заполнении. Кнопки действий были сгруппированы и имели согласованные стили - основная кнопка действия выделялась цветом, а второстепенные имели нейтральное оформление.

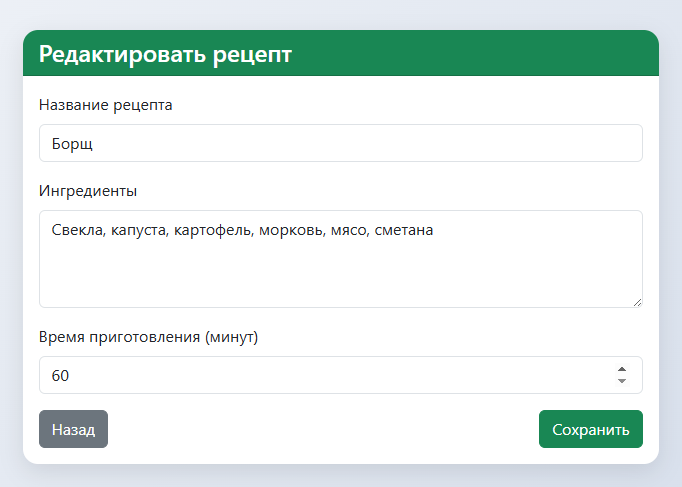


Рисунок 15 – Форма редактирования рецептов

Интерактивные элементы интерфейса получили продуманные состояния *hover* и *active*. Кнопки имели плавные переходы между состояниями, а ссылки - изменение цвета при наведении. Для опасных операций, таких как удаление записей, была реализована система подтверждения через *JavaScript*-диалог, что предотвращало случайную потерю данных. Кнопки удаления имели красное оформление, визуально сигнализирующее об опасности действия.

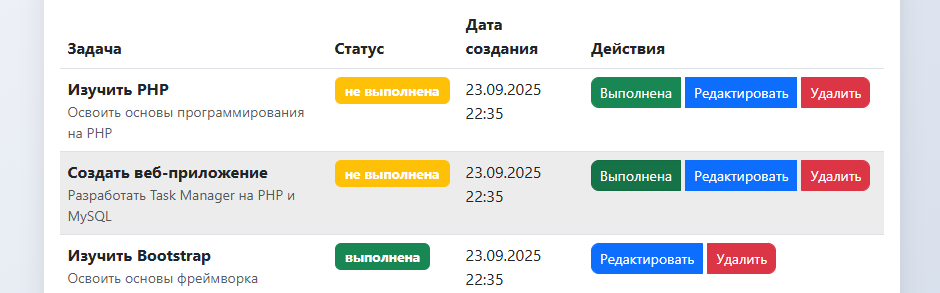


Рисунок 16 – Пример элементов интерфейса при наведении

Адаптивность интерфейса обеспечивалась средствами *Bootstrap*-сетки. На мобильных устройствах таблица преобразовывалась в вертикальный режим с переносом содержимого, а кнопки действий располагались в виде стека для удобства касания. Карточки форм занимали всю доступную ширину экрана, сохраняя при этом читаемость и удобство использования.

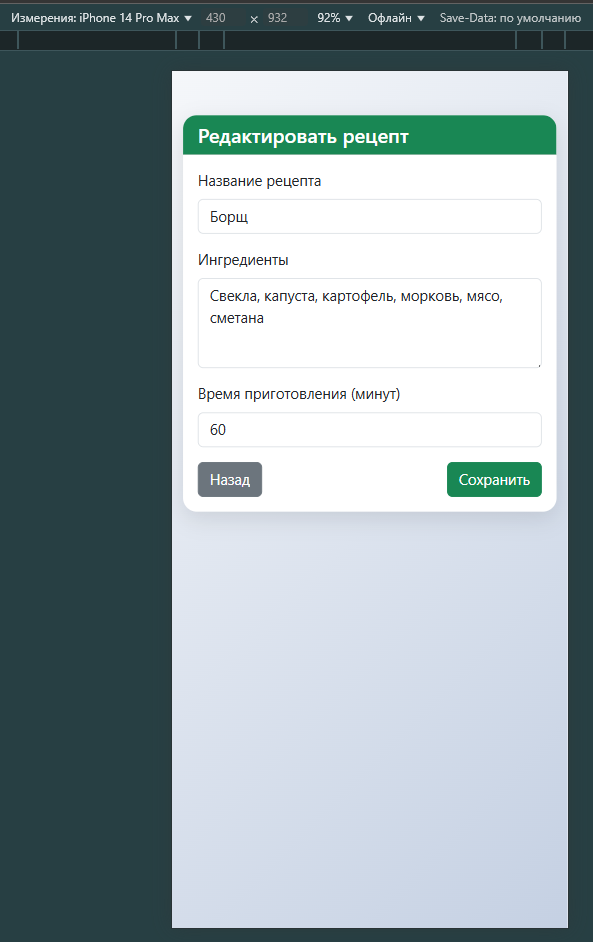


Рисунок 17 – Пример адаптивности на смартфоне

Типографика была выдержана в едином стиле с четкой иерархией заголовков. Основной текст использовал системные шрифты для оптимальной читаемости, а важные элементы выделялись полужирным начертанием. Размеры шрифтов были подобраны таким образом, чтобы обеспечить комфортное чтение как на ноутбуках, так и на мобильных экранах.

Пример использования интерфейса: пользователь открывает главную страницу каталога рецептов и видит таблицу с существующими рецептами. Для добавления нового рецепта он нажимает кнопку «Добавить рецепт» в правом верхнем углу, заполняет форму с названием, ингредиентами и временем приготовления, после чего сохраняет данные. При необходимости редактирования существующего рецепта пользователь нажимает кнопку «Редактировать» в соответствующей строке таблицы, вносит изменения и сохраняет их. Для модерации рецептов предусмотрена кнопка «Проверить», которая изменяет статус записи.

2.5 Ответ на контрольный вопрос: Обработка ошибок подключения к БД и выполнения запросов

Обработка ошибок при работе с базой данных является критически важным аспектом разработки надежных веб-приложений. В рамках данного проекта был реализован многоуровневый подход к обработке ошибок, обеспечивающий как безопасность приложения, так и удобство для пользователя.

На уровне подключения к базе данных была применена стратегия отказоустойчивости через механизм повторных попыток. Функция *waitForMySQL* в скриптах инициализации реализует повторное подключение с интервалом 5 секунд и максимальным количеством попыток 10. Это решает проблему асинхронного запуска контейнеров, когда приложение пытается подключиться к базе данных до ее полной готовности. Использование *PDO* с настроенными параметрами таймаута и режимом исключений обеспечивает детализированный контроль над процессом подключения.

В конфигурационном файле *config*.*php* используется блок *try*-*catch* для перехвата исключений при создании *PDO*-объекта. В случае ошибки подключения генерируется исключение *PDOException* с понятным сообщением, которое может быть записано в лог или отображено в режиме отладки. На *production*-сервере сообщения об ошибках должны быть скрыты от пользователя для предотвращения утечки чувствительной информации, в то время как в *development*-среде подробные сообщения помогают в диагностике проблем.

При выполнении *SQL*-запросов применяется подход подготовленных выражений (*prepared* *statements*), который не только предотвращает *SQL*-инъекции, но и обеспечивает корректную обработку ошибок выполнения запросов. Каждый вызов *prepare*() и *execute*() обернут в неявный блок *try*-*catch*, что позволяет перехватывать исключения на уровне приложения. В реальном *production*-приложении рекомендуется добавить явную обработку исключений для каждого запроса с перенаправлением пользователя на страницу ошибки или отображением понятного сообщения.

Для операций изменения данных (*INSERT*, *UPDATE*, *DELETE*) реализована проверка результата выполнения запроса. Хотя в текущей реализации используется простое перенаправление при успешном выполнении, в расширенной версии следует добавить проверку количества затронутых строк и обработку случаев, когда запрос не повлиял на данные (например, при попытке обновления несуществующей записи).

Особое внимание уделено валидации пользовательского ввода перед формированием запросов. Проверка на пустые значения, корректность типов данных и граничных значений выполняется на стороне сервера перед взаимодействием с базой данных. Это предотвращает выполнение заведомо некорректных запросов и снижает нагрузку на СУБД.

В текущем проекте также реализован механизм проверки существования записей перед выполнением операций редактирования и удаления. При попытке доступа к несуществующей записи через *GET*-параметры выполняется редирект на главную страницу, что предотвращает возникновение ошибок и обеспечивает целостность пользовательского опыта.

Важным аспектом является единообразная обработка ошибок *across* всего приложения. Все скрипты, работающие с базой данных, подключают единый *config*.*php*, который централизованно настраивает параметры подключения и обработки ошибок. Это обеспечивает *consistency* поведения приложения и упрощает поддержку кода.

Таким образом, комплексный подход к обработке ошибок включает предварительную валидацию данных, использование подготовленных выражений, механизмы повторных попыток подключения, понятную обратную связь для пользователя и систему логирования для администраторов. Такой подход обеспечивает надежность, безопасность и отказоустойчивость веб-приложения при работе с базой данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения контрольной работы успешно закреплены практические навыки бекенд разработки через создание двух полнофункциональных веб-приложений, интегрированных с системой управления базами данных *MySQL*. Особенностью работы стало параллельное развитие базовой системы управления задачами и специализированного каталога рецептов по индивидуальному варианту, что позволило освоить как универсальные принципы разработки, так и специфику адаптации под конкретную предметную область.

Ключевым достижением стала комплексная реализация полного цикла *CRUD*-операций с учетом требований безопасности и удобства пользователя. Разработана оптимальная структура базы данных для обеих предметных областей, обеспечивающая эффективное хранение и обработку информации. Особое внимание уделено проектированию специализированных полей для каталога рецептов, демонстрирующее умение адаптировать общие решения под конкретные задачи.

Важным аспектом работы стало внедрение современных технологий контейнеризации при помощи *Docker*, что выходит за рамки базовых требований и свидетельствует о освоении продвинутых подходов к развертыванию веб-приложений. Созданная инфраструктура обеспечивает изолированную среду выполнения, повторяемость развертывания и простоту масштабирования.

В области безопасности реализован комплекс мер, включающий использование подготовленных выражений *PDO* для защиты от *SQL*-инъекций, валидацию пользовательского ввода и экранирование выводимых данных. Разработан механизм обработки ошибок, обеспечивающий стабильную работу приложения в различных сценариях.

Интерфейсная часть приложения создана с применением фреймворка *Bootstrap* 5, что гарантирует адаптивность и современный внешний вид. Для каждого приложения разработана уникальная визуальная идентичность, соответствующая его тематике и улучшающая пользовательский опыт.

Все поставленные в ходе работы цели достигнуты: получены практические навыки проектирования баз данных, реализации *back*-*end* логики, обеспечения безопасности и развертывания приложений. Разработанные веб-приложения готовы к использованию и демонстрируют высокий уровень освоения технологий *PHP* и *MySQL*, а также умение работать с современными инструментами разработки и деплоя.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Программный код приложения**

С полным кодом веб-приложения можно ознакомиться по ссылке:

***https://github.com/POOSTOTEL/ITaWP-test-1.git***

***Файл task\_manager/index.php***

*<?php*

*require\_once 'config.php';*

*$stmt = $pdo->query(«SELECT \* FROM tasks ORDER BY created\_at DESC»);*

*$tasks = $stmt->fetchAll(PDO::FETCH\_ASSOC);?>*

*<!DOCTYPE html>*

*<html lang=«ru»>*

*<head>*

*<meta charset=«UTF-8»>*

*<meta name=«viewport» content=«width=device-width, initial-scale=1.0»>*

*<title>Task Manager</title>*

*<link href=«https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css» rel=«stylesheet»>*

*<link rel=«stylesheet» href=«style.css»>*

*</head>*

*<body>*

*<div class=«container mt-5»>*

*<div class=«row»>*

*<div class=«col-md-8 mx-auto»>*

*<div class=«card shadow»>*

*<div class=«card-header bg-primary text-white d-flex justify-content-between align-items-center»>*

*<h3 class=«mb-0»>Система управления задачами</h3>*

*<a href=«add.php» class=«btn btn-light»>Добавить задачу</a>*

*</div>*

*<div class=«card-body»>*

*<?php if (count($tasks) > 0): ?>*

*<div class=«table-responsive»>*

*<table class=«table table-hover»>*

*<thead>*

*<tr>*

*<th>Задача</th>*

*<th>Статус</th>*

*<th>Дата создания</th>*

*<th>Действия</th>*

*</tr>*

*</thead>*

*<tbody>*

*<?php foreach ($tasks as $task): ?>*

*<tr>*

*<td>*

*<div class=«fw-bold»><?= htmlspecialchars($task['title']) ?></div>*

*<?php if (!empty($task['description'])): ?>*

*<div class=«text-muted small»><?= htmlspecialchars($task['description']) ?></div>*

*<?php endif; ?>*

*</td>*

*<td>*

*<span class=«badge bg-<?= $task['status'] == 'выполнена' ? 'success' : 'warning' ?>«>*

*<?= $task['status'] ?>*

*</span>*

*</td>*

*<td><?= date('d.m.Y H:i', strtotime($task['created\_at'])) ?></td>*

*<td>*

*<div class=«btn-group btn-group-sm»>*

*<?php if ($task['status'] != 'выполнена'): ?>*

*<a href=«update\_status.php?id=<?= $task['id'] ?>&status=выполнена» class=«btn btn-success»>Выполнена</a>*

*<?php endif; ?>*

*<a href=«edit.php?id=<?= $task['id'] ?>« class=«btn btn-primary»>Редактировать</a>*

*<a href=«delete.php?id=<?= $task['id'] ?>« class=«btn btn-danger» onclick=«return confirm('Вы уверены?')»>Удалить</a>*

*</div>*

*</td></tr>*

*<?php endforeach; ?>*

*</tbody></table></div>*

*<?php else: ?>*

*<div class=«text-center py-4»>*

*<p class=«text-muted»>Задачи не найдены. Добавьте первую задачу!</p>*

*<a href=«add.php» class=«btn btn-primary»>Добавить задачу</a>*

*</div>*

*<?php endif; ?>*

*</div>*

*</div>*

*</div></div></div><script src=«https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js»></script>*

*</body>*

*</html>*