1. Общее описание БД и решаемых ею задач

База данных разработана для организации по прокату плавсредств. Она предназначена для учета клиентов, плавсредств, а также операций по прокату. БД включает в себя таблицы для хранения информации о клиентах и плавсредствах. Она позволяет регистрировать новых клиентов, отслеживать наличие плавсредств, а также **записывать** и **контролировать** арендные операции через специальную **админ панель**.

2. Цели и задачи:

Регистрация клиентов: Система позволяет вести базу данных клиентов с основной информацией, такой как ФИО, контактные данные и история аренд.

Учет плавательных средств: БД хранит данные о различных типах плавательных средств, их моделях, техническом состоянии и стоимости аренды.

~~Управление арендными операциями: Система записывает информацию о совершенных арендных сделках, включая даты начала и окончания, продолжительность, стоимость и статус (активная, завершенная).~~

Для упрощения управления системой предусмотрена **административная панель** с функционалом для добавления/удаления клиентов, плавательных средств, а также контроля за арендными операциями.

Заключение:

В рамках работы необходимо будет ознакомиться с различными типами баз данных, методами их создания, а также практически изучать основные операции с данными, включая добавление, удаление, изменение и поиск информации.

Целью данной курсовой работы является изучение основ работы с базами данных и освоение основных команд MySQL.

В процессе выполнения курсовой работы будет рассмотрено множество аспектов работы с базами данных, начиная с создания таблиц, добавления и изменения данных, заканчивая выполнением сложных запросов и оптимизацией баз данных.

Я буду работать в среде разработки **PHPStorm**, используя ее возможности для комфортной и эффективной работы над проектом.

Для создания и развертывания веб-приложения буду использовать **AMPPS** - удобное и функциональное средство для работы с **Apache**, **MySQL** и **PHP**.

**AMPPS** предоставляет возможность создания локального сервера и установки необходимых компонентов для разработки проекта.

Для придания веб-приложению современного и качественного дизайна буду использовать **Bootstrap** - самый популярный фреймворк для разработки адаптивных и стильных веб-интерфейсов. Я подключил последнюю версию **Bootstrap 5.3.2** и воспользуюсь его компонентами, сеткой и стилями для быстрого создания пользовательского интерфейса проекта.

Для создания эстетически приятного внешнего вида текста на страницах веб-приложения я воспользуюсь **Google Fonts** - бесплатной библиотекой шрифтов от Google. Это позволит добавить разнообразные и стильные шрифты к текстовым элементам и придать им уникальность и привлекательность.

Для добавления иконок к кнопкам, ссылкам и другим элементам интерфейса буду использовать **Font Awesome** - иконическую библиотеку, которая предоставляет нам множество векторных иконок, которые можно использовать в нашем проекте.

**Font Awesome** позволяет легко добавлять иконки в свой дизайн без необходимости использовать изображения или растровую графику, что значительно упрощает и ускоряет работу.

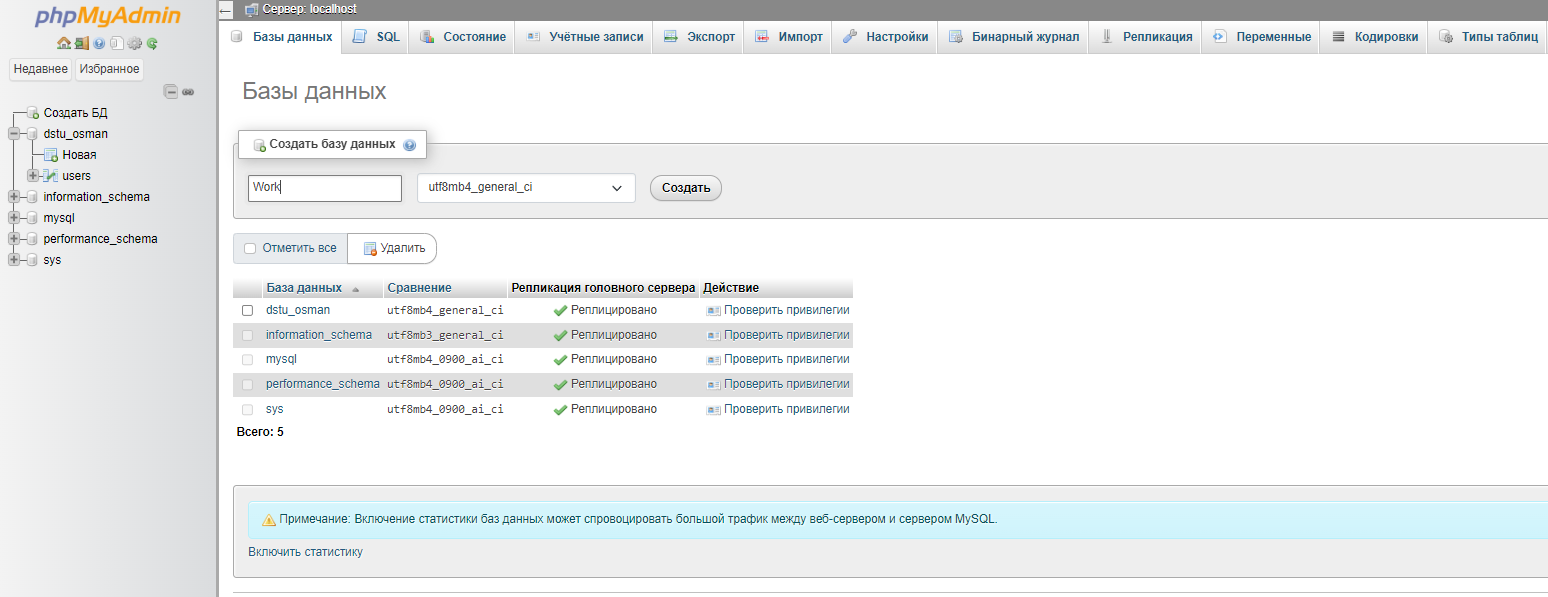
MySQL

После установки **AMPPS** и запуска сервера, можно приступить к подключению к **phpMyAdmin**.

Для этого необходимо:

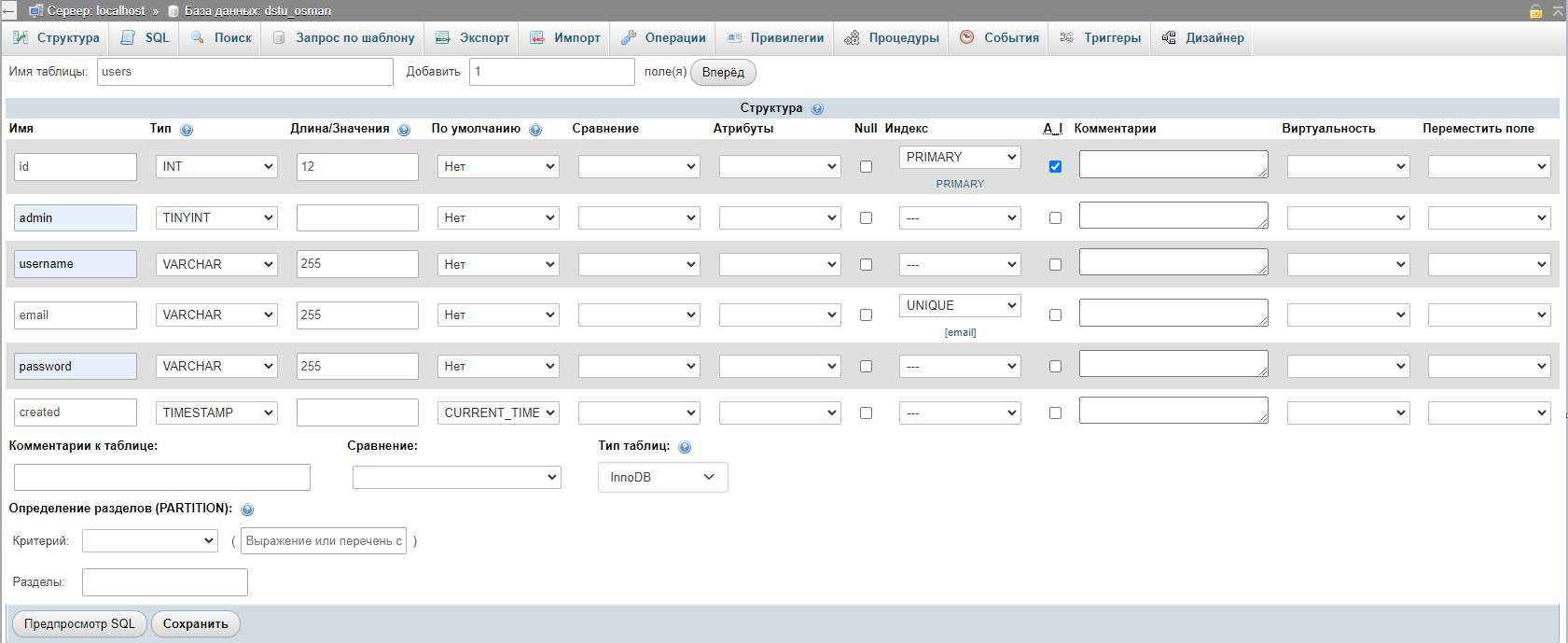
1. Открыть веб-браузер и введите в адресной строке <http://localhost/phpmyadmin/>
2. На странице входа ввести логин и пароль. Обычно логин – это «root», а пароль может быть пустым или установлен по умолчанию во время установки **AMPPS**.
3. Нажимаем на кнопку «Вход» или нажмите Enter, чтобы войти в **phpMyAdmin**.

После успешного входа пользователь будет перенаправлен на главный экран **phpMyAdmin**, где можно управлять базами данных и выполнять различные операции.



Интерфейс позволяет пользователю непосредственно работать с базами данных и таблицами. Можно выбрать базу данных, чтобы просмотреть ее содержимое или выполнить SQL-запросы. Также можете создавать, редактировать и удалять таблицы, добавлять и изменять данные в них.

**PhpMyAdmin** также обладает множеством других функций, таких как импорт и экспорт баз данных, управление пользователями и привилегиями, отображение структуры базы данных и многое другое.



Создание таблицы в **phpMyAdmin** – процесс, при котором администратор баз данных или разработчик создает новую таблицу с помощью веб-интерфейса **phpMyAdmin**.

Таблица представляет собой структурированную коллекцию данных, организованных в виде строк и столбцов.

При определении столбцов таблицы для них необходимо указать тип данных. Каждый столбец должен иметь тип данных. Тип данных определяет, какие значения могут храниться в столбце, сколько они будут занимать места в памяти.

Типы данных в MySQL:

INT – Основной целочисленный тип данных. Тип поля INT использует 4 байта для хранения значения. Минимально значение -2147483648, максимальное 2147483647.

TINYINT – Очень малое целое число. Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255

A\_I – MySQL использует ключевое слово AUTO\_INCREMENT для выполнения функции автоматического приращения. По умолчанию начальное значение для AUTO\_INCREMENT равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

VARCHAR – Хранит строки переменной длины до 65 535 символов. Причем в памяти хранится именно та длина, которая была указана при создании

UNIQUE – Правила, которые применяет ядро СУБД SQL Server. Например, ограничения можно использовать для обеспечения того, чтобы в указанные столбцы, не входящие в состав первичного ключа, не вводились повторяющиеся значения.

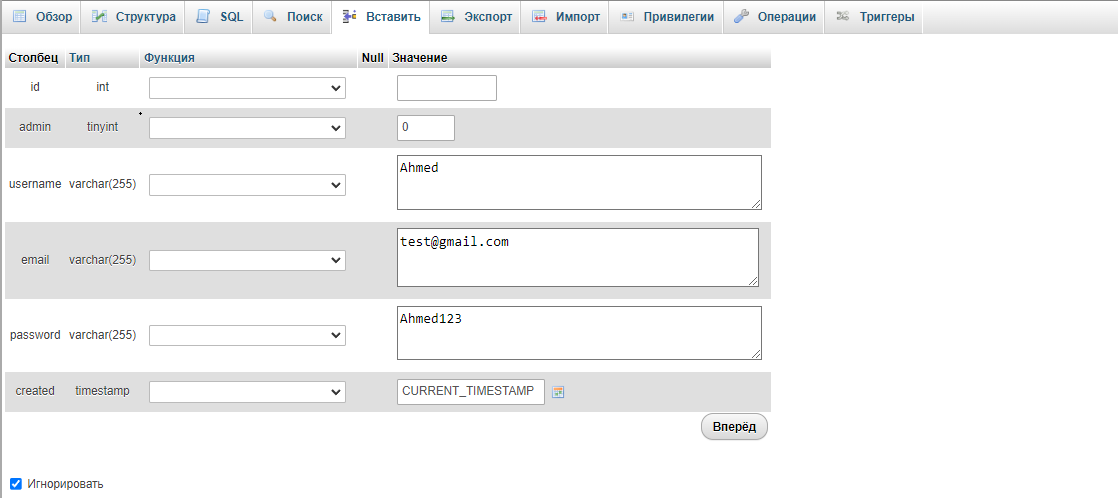
TIMESTAMP – Тип данных для хранения даты и времени. Данные хранятся в виде количества секунд, прошедших с начала «эпохи Юникса».

CURRENT\_TIMESTAMP – Функция что наследует это значение от операционной системы компьютера, на котором работает экземпляр SQL Server.

Чтобы вставить значение пользователя в таблицу phpMyAdmin, следует выполнить следующие действия:

1. Откройте phpMyAdmin веб-интерфейс и выберите базу данных, в которой находится таблица, к которой вы хотите добавить запись.
2. Перейдите на вкладку "Вставить"
3. Ввести значения пользователя.
4. Нажать на кнопку "Вперёд" для сохранения новой записи в таблице.

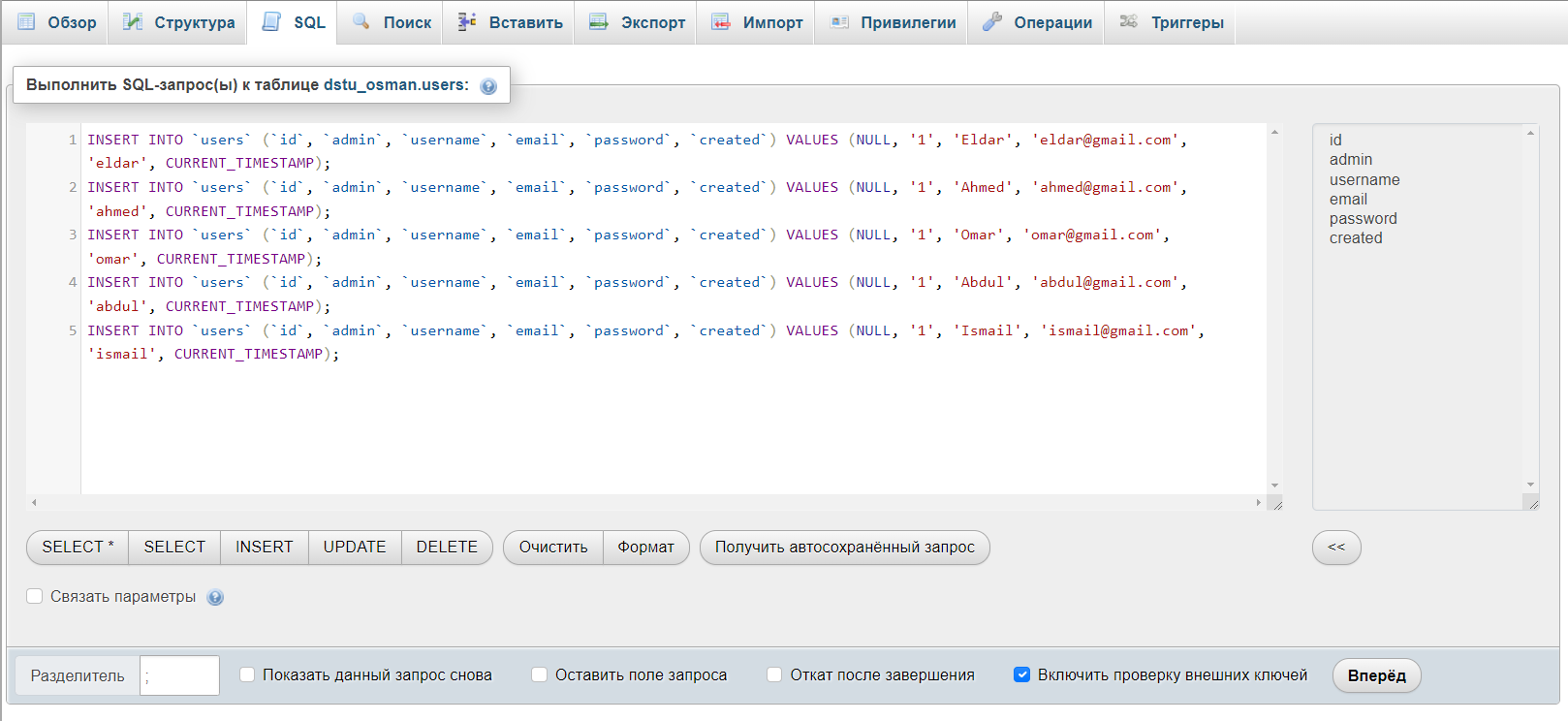
Добавляем пользователя через "Вставить" (Insert)

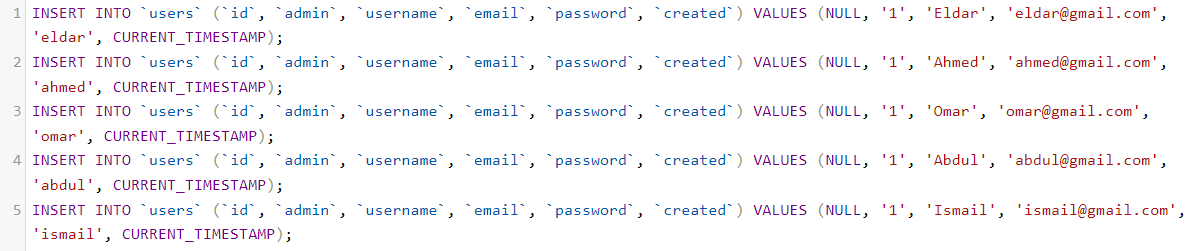


Чтобы вставить значение пользователя в таблицу phpMyAdmin через SQL, вам потребуется выполнить следующие действия:

1. Откройте phpMyAdmin веб-интерфейс и выберите базу данных, в которой находится таблица, к которой вы хотите добавить запись.
2. Перейдите на вкладку "SQL" или "Запрос" (в зависимости от версии phpMyAdmin).
3. Введите SQL-запрос INSERT INTO, указывающий название таблицы, в которую вы хотите добавить запись, и список полей, в которые нужно вставить значения

Добавляем нескольких пользователей через SQL





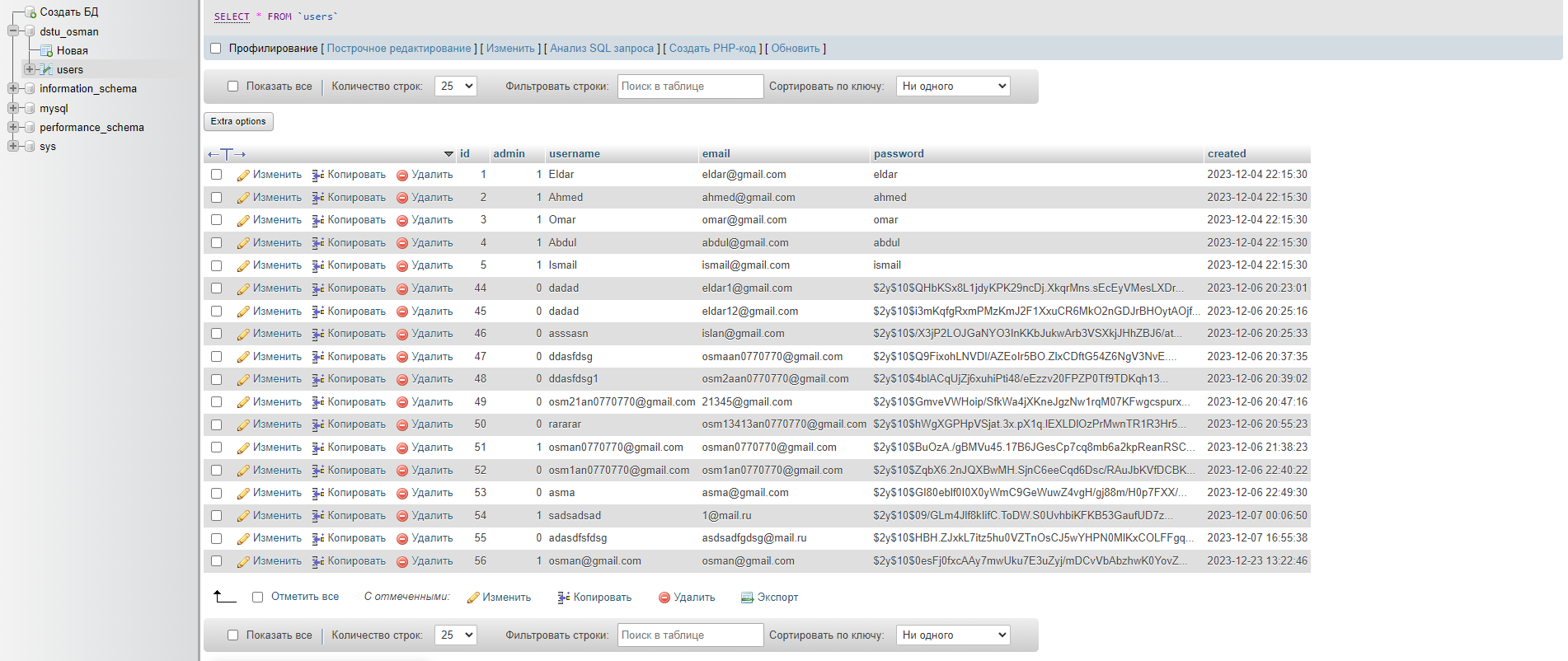
После краткого анализа принципа работы **PhpMyAdmin** необходимо создать сайт и подключить к нему базу данных.

Для работы с **MySQL** не надо ничего дополнительно устанавливать и настраивать: всё необходимое уже доступно вместе со стандартной поставкой PHP.

Пример рабочего файла **connect.php**, в котором содержится код, который создает объект подключения к базе данных и устанавливает соединение с помощью параметров, таких как хост, имя пользователя и пароль.

**Ниже будут приведены готовые таблицы в базе данных и их описание:**

Таблица пользователей



Эта таблица содержит информацию о пользователях системы. Каждый пользователь имеет уникальный идентификатор, имя, фамилию, пароль и адрес электронной почты. Такая структура данных позволяет эффективно управлять информацией о зарегистрированных пользователях.

Таблица категорий

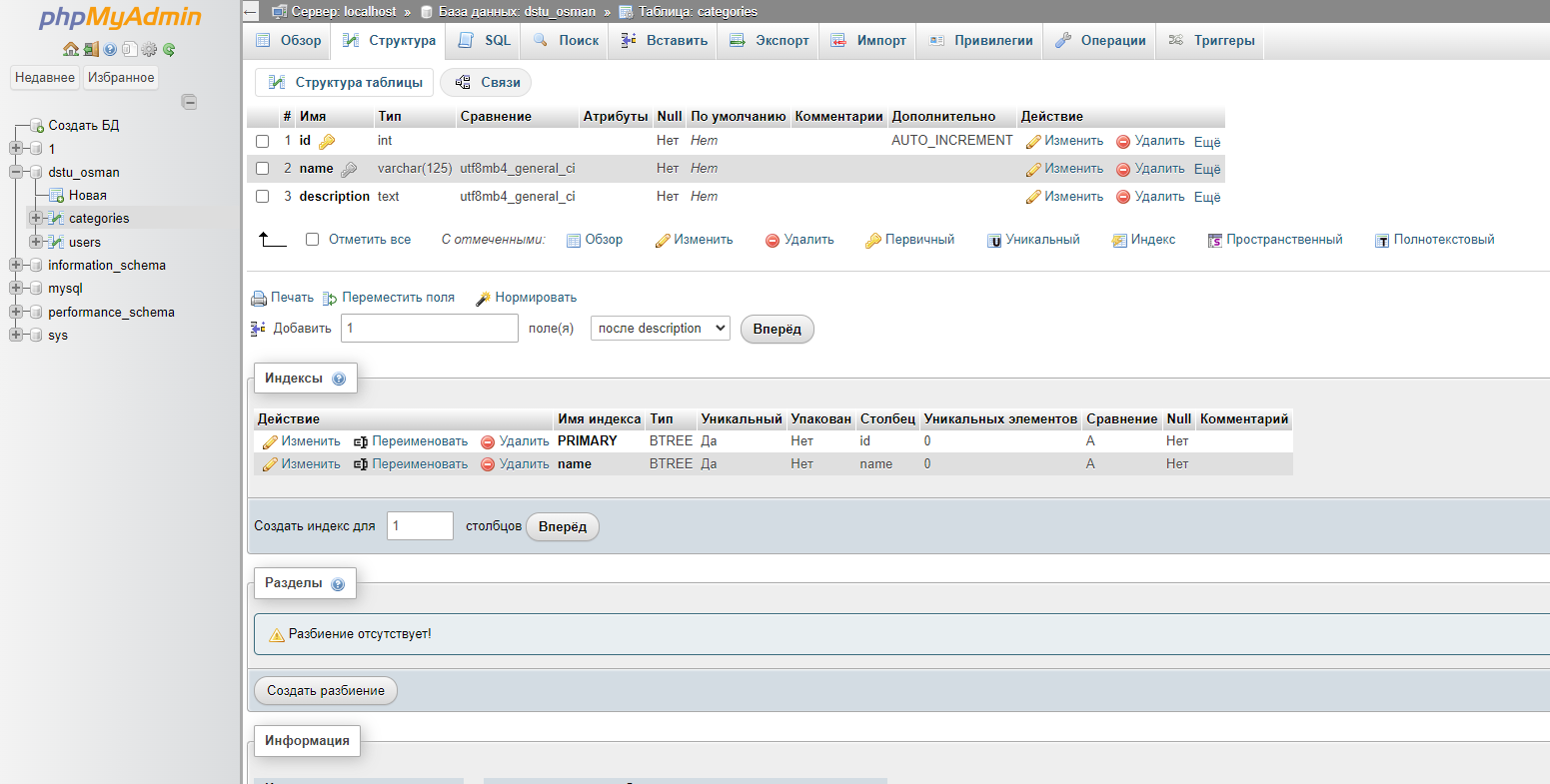
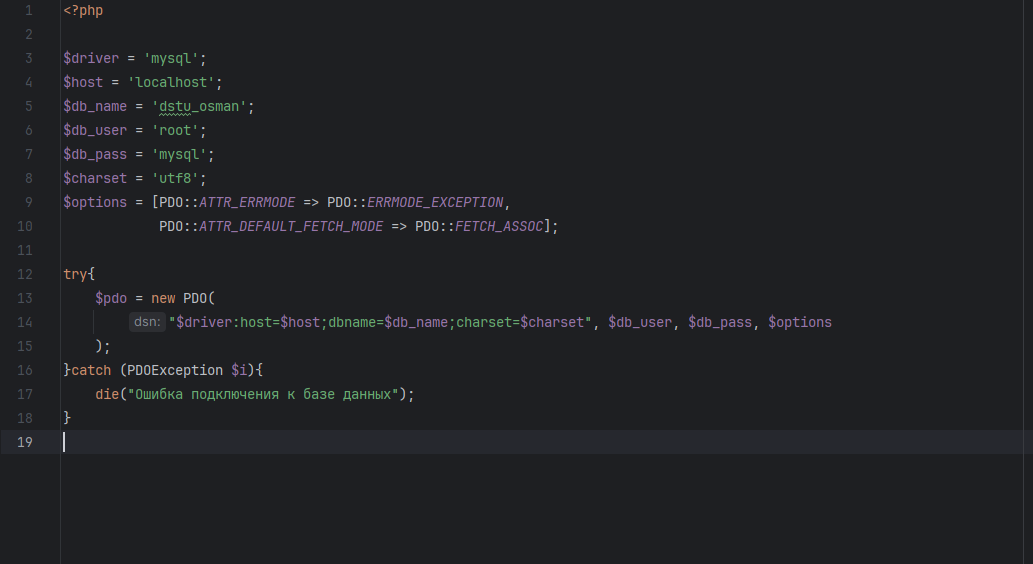


Таблица категорий в базе данных является ключевым компонентом для организации и классификации данных. Она представляет собой структурированное хранилище, где каждая запись относится к определенной категории, обладающей уникальным идентификатором.

Подключение к базе данных



**PDO** (PHP Data Objects) - Это объектно-ориентированный расширения PHP, которое предоставляет единый интерфейс для взаимодействия с различными базами данных.

PDO::ATTR\_ERRMODE - Установка способа обработки ошибок.

PDO::ERRMODE\_EXCEPTION - В дополнение к установкам уровня вывода ошибок, PDO выбросит PDOException - исключение и задаст его свойства, так чтобы отразить код ошибки и сведения об ошибке. Этот параметр также полезен при отладке, так как он будет эффективно "останавливать" сценарий в точке ошибки, очень быстро указывая «пальцем» на потенциальные проблемы в вашем коде.

PDO::ATTR\_DEFAULT\_FETCH\_MODE - Это атрибут, который устанавливает режим извлечения данных по умолчанию при использовании объекта PDO в PHP.

PDO::FETCH\_ASSOC - Это константа, которая указывает, что данные должны быть извлечены в виде ассоциативного массива.

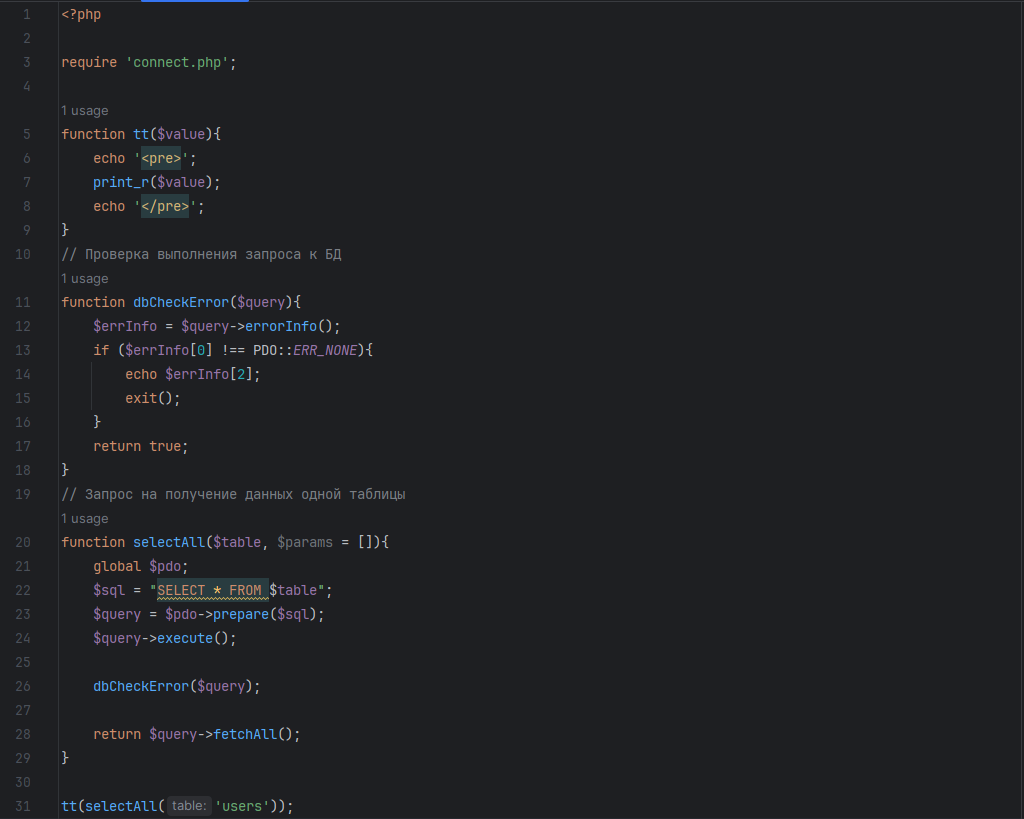
Конструкция try catch - Это специальная инструкция из двух блоков, которая позволяет перехватить все исключения и их обработать. Любые исключения, которые будут выброшены кодом, расположенным внутри этого блока, будут перехвачены и переданы во второй блок. Если ошибки не было, то этот блок пропускается.

$pdo = new PDO - Экземпляр класса.

"$driver:host=$host;dbname=$db\_name;charset=$charset", $db\_user, $db\_pass, $options - Параметры необходимые для подключения

catch (PDOException $i) - Тем самым, если при подключении возникает исключение типа PDOException, то код завершает свою работу и выводит сообщение "Ошибка подключения к базе данных".

Функция по запросам из таблицы



Приведенная ниже функция принимает значение, выводит в форматированном тексте и пробрасывает само значение $value.

function tt($value){

echo '<pre>';

print\_r($value);

echo '</pre>';

}

function dbCheckError($query) - Функция используется для проверки ошибок после выполнения запроса к базе данных.

$errInfo = $query->errorInfo(); - Получаем ошибки в переменную errorInfo. Возврат ошибки.

if ($errInfo[0] !== PDO::ERR\_NONE) - Проверяем переменную errInfo, обращаемся к массиву. Делаем проверку: если строгое неравенство не равно ошибке, то мы ее выводим.

echo $errInfo[2]; - Выводим ошибку с массива, обращаемся ко 2-му индексу.

exit(); - Прерывает выполнение кода в случае ошибки.

function selectAll($table, $params = []) - Функция получает данные с любой таблицы, которую мы хотим использовать. Принимает переменную table.

global $pdo; - Указываем что используем глобальную переменную PDO.

$sql = "SELECT \* FROM $table"; - Выполняет запрос. WHERE admin = 1 AND username = 'Ahmed' - выводит всех админов, в данном случае выводит админа только с именем Ахмед

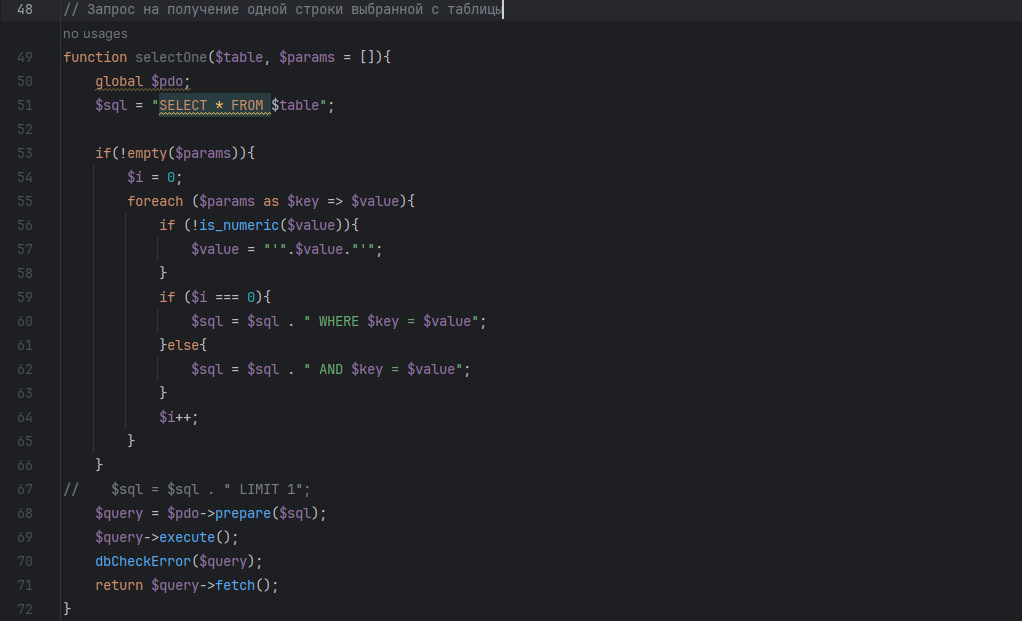
$query = $pdo->prepare($sql); - Обращение к экземпляру класса PDO из connect. Выполняется подготовка.

$query->execute(); - Вызывает пустой метод. Может получить информацию, если нет ошибки. Выполняет сам запрос.

return $query->fetchAll(); - Возвращает массив с данными. Получаем всю базу.

tt(selectAll('users')); - Получаем красивый форматированный массив.

Запрос на получение одной строки выбранной с таблицы



if(!empty($params)) - Если переменная $params не пустая (то есть содержит хотя бы одно значение), то блок кода внутри оператора if будет выполнен.

foreach ($params as $key => $value) - Позволяет перебрать каждый элемент массива.

if (!is\_numeric($value)) - Это выражение проверяет, является ли переменная $value числовым значением.

$value = " ' ".$value." ' "; - Преобразует значение переменной $value в строку.

**Если равно нулю.**

if ($i === 0){

$sql = $sql . "WHERE $key => $value";

**Если больше чем ноль.**

}else{

$sql = $sql . "AND $key => $value";

}

**Условие под запись.**

$params = [

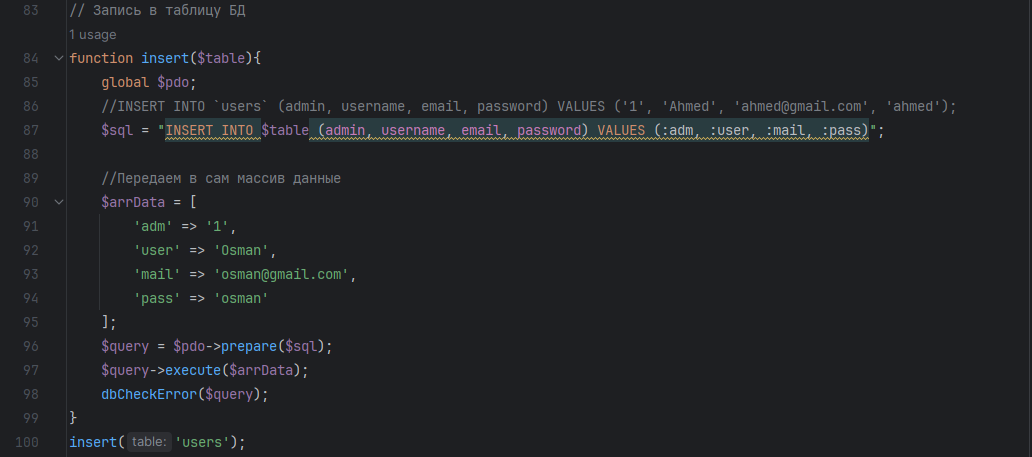
'admin' => 1,

'username' => 'Eldar'

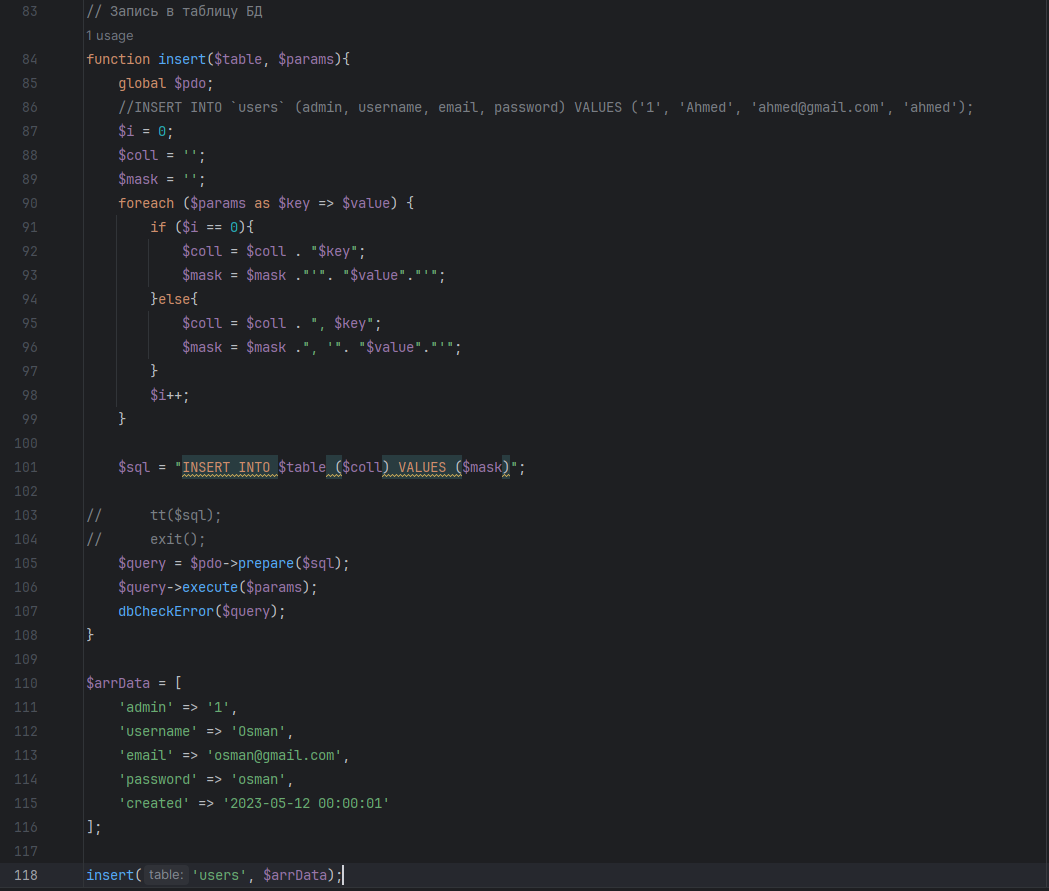
];

$1++; - Инкремент.

Запись в таблицу базы данных



По сути, мы можем получать какой-то ассоциативный массив и дальше его подставлять в наши значения (VALUES). Соответственно, наша функция может быть более гибкой.



Данный код представляет функцию insert, которая служит для записи данных в таблицу базы данных. Функция принимает два аргумента: название таблицы и массив параметров, содержащий значения для вставки.

Функция сначала формирует SQL-запрос на основе переданных параметров. Затем создается подготовленное выражение с помощью глобальной переменной $pdo (соединение с базой данных), и выполняется запрос с помощью метода execute.

Пример использования функции приведен в конце кода, где создается массив $arrData, содержащий значения для вставки в таблицу users. Функция insert вызывается с аргументами 'users' и $arrData.

В начале функции происходит инициализация необходимых переменных. Переменная $i устанавливается в 0 для отслеживания первого элемента массива $params. Переменные $coll и $mask инициализируются пустыми строками. $coll будет использоваться для хранения имен столбцов таблицы, а $mask будет использоваться для хранения значений, которые будут вставлены в таблицу.

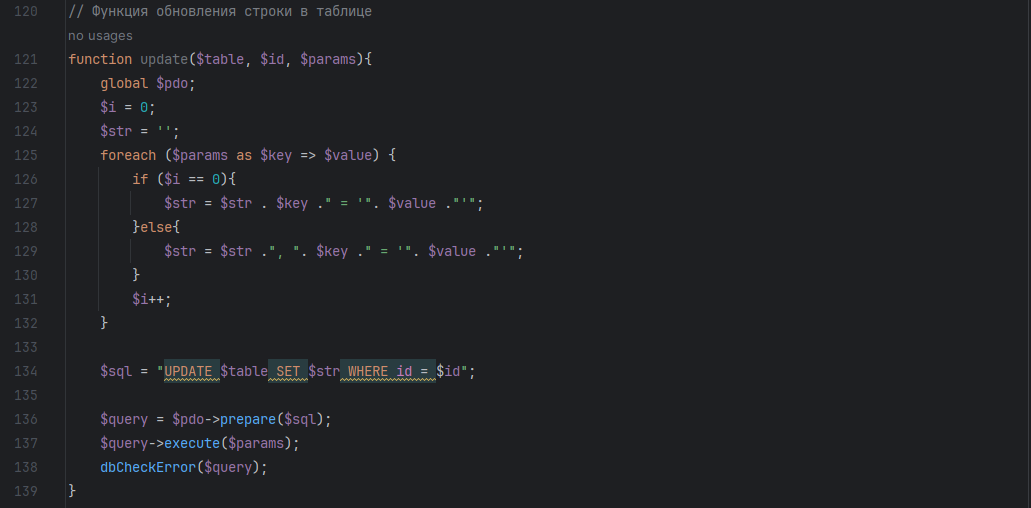
Затем происходит перебор массива $params. Для каждой пары ключ-значение создается строка, которая добавляется к переменным $coll и $mask. Если это первый элемент массива, то пара будет добавлена без запятой, в противном случае пара будет добавлена с запятой перед ней. Значение каждого элемента массива $params заключается в одинарные кавычки.

После завершения перебора массива формируется SQL-запрос с использованием переменных $table (имя таблицы), $coll (список столбцов) и $mask (список значений). SQL-запрос выглядит следующим образом:

**INSERT INTO users (admin, username, email, password, created) VALUES ('1', 'Osman', 'osman@gmail.com', 'osman', '2023-05-12 00:00:01')**

Важно отметить, что создание дублирующих таблиц с теми же данными может привести к возникновению проблем.

Функции (UPDATE & DELETE)

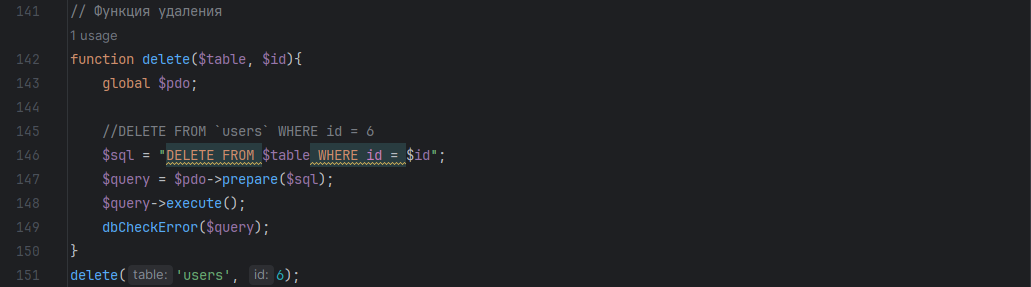


Функция update принимает три параметра: $table, $id и $params.

$table - название таблицы в базе данных, которую нужно обновить.

$id - идентификатор записи, которую нужно обновить.

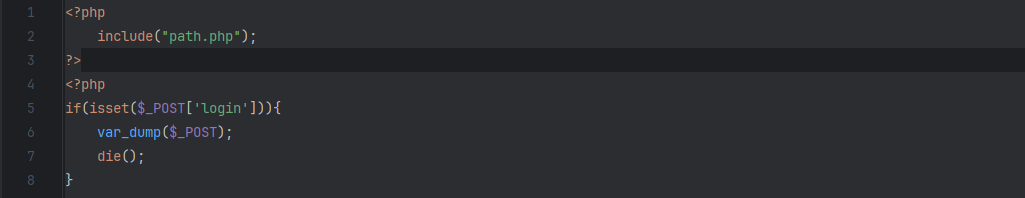
$params - массив, содержащий пары ключ-значение, где ключи - названия полей, а значения - новые значения полей, которые нужно обновить.$params.



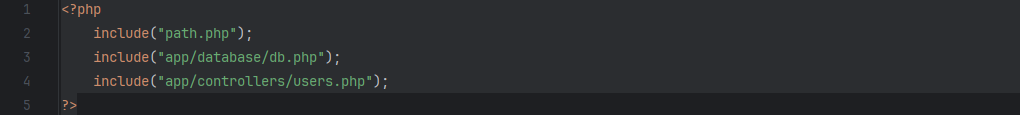
Функция delete() используется для удаления записей из таблицы базы данных. Она принимает два аргумента - название таблицы $table и идентификатор записи $id, которую нужно удалить.

Внутри функции используется глобальная переменная $pdo, которая представляет собой объект подключения к базе данных.

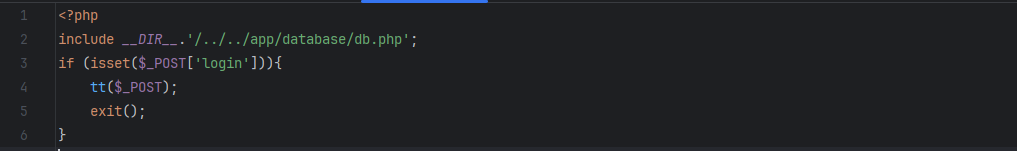
Форма регистрации



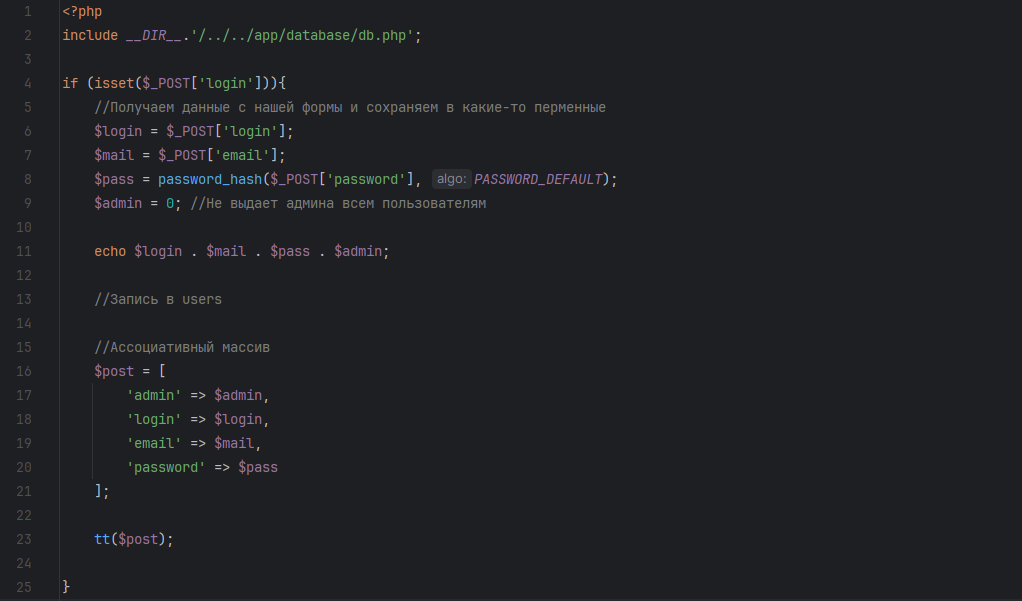
Проверка. Если у нас в супер-глобальном массиве $\_POST есть заполненные данные "login", то тогда выводит абсолютно весь массив.



После проверки мы используем функцию include, чтобы подключить файлы "path.php", "db.php" и "users.php".

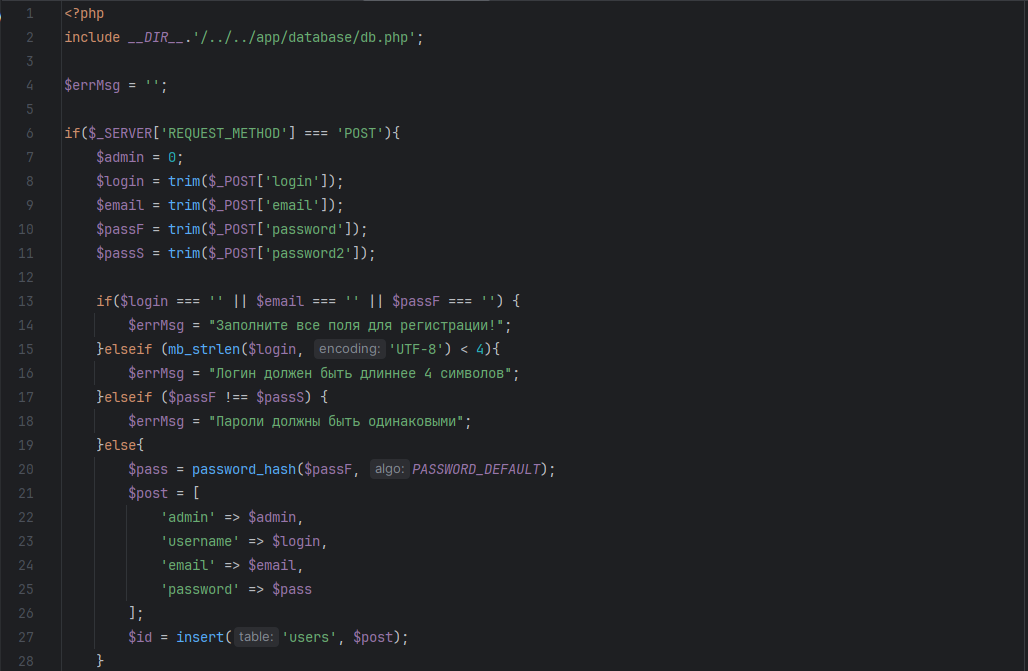


Если файлы не инклудятся - можно прописать адрес через/ *DIR* / - у меня так заработало: include \_\_DIR\_\_.'/../../app/database/db.php'; можно тоже и через set\_include\_path() и потом уже include.



Создаем обработчик формы user.php в директории DSTU\_Osman\app\controllers.

Проверка данных с формы регистрации



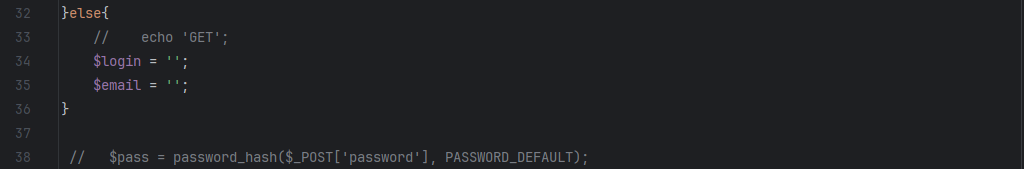
Проверки выполняются для каждого поля, такие как проверка на пустое значение, на соответствие определенному формату и т. д. При несоответствии условиям, выводится соответствующее сообщение.

Здесь переменные $admin, $login, $email и $pass должны быть предварительно установлены с помощью значений, полученных от пользователя.

Затем, эти значения присваиваются соответствующим ключам в ассоциативном массиве $post. Ключи 'username', 'email' и 'password' соответствуют столбцам в таблице "users".

Далее, функция insert() вызывается с параметрами 'users' (название таблицы) и $post (ассоциативный массив с данными пользователя). Её функция заключается в вставке данных пользователя в базу данных и возвращает идентификатор вставленной записи.

Таким образом, после выполнения данного кода, новый пользователь будет добавлен в таблицу "users" с указанными значениями для полей "username", "email" и "password". Идентификатор этой записи будет сохранен в переменной $id.



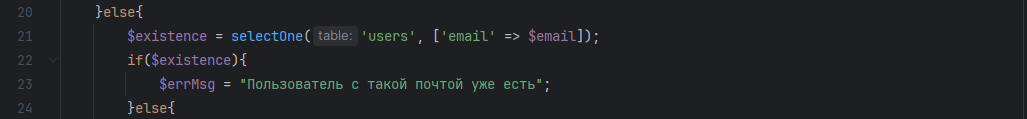
Если условие истинно (т.е. метод запроса не равен "POST"), то инициализируются переменные $login и $email пустыми значениями.

Если условие ложно, то это предполагает, что запрос был отправлен методом POST, поэтому вводимые данные будут доступны через массив $\_POST (или другой, который используется для хранения отправленных данных). Таким образом, переменные $login и $email будут заполнены введенными значениями из формы.

В итоге, если были введены неверные данные, то после обработки ошибок, переменные $login и $email будут содержать введенные значения, чтобы пользователь не должен был их снова вводить.

password\_hash - это функция, которая преобразует пользовательский пароль в хеш (непонятный случайный набор символов). Хеширование пароля является безопасным способом хранения паролей, поскольку хеш невозможно обратно преобразовать в исходный пароль и защищает пароли от несанкционированного доступа.

Проверка на наличие пользователя в базе данных

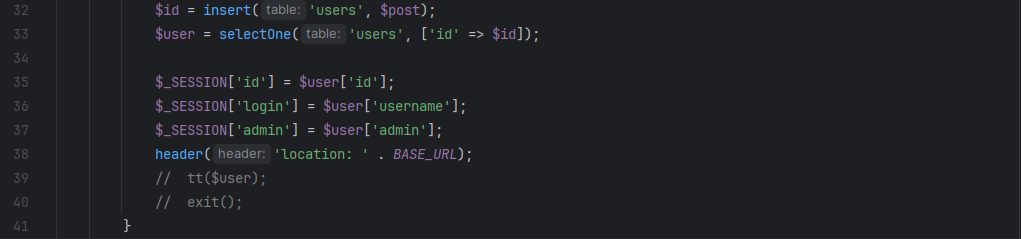


Проверяем наличие соответствующей записи о пользователе в базе данных.

$existence = selectOne('users', ['email' => $email]); - Создадим переменную $existence, которая будет проводить проверку наличия почты в базе данных.

if($existence) - Если моя переменная $existence строго равна переменной $email, введенной пользователем, то выводит ошибку.

Создание SESSION после регистрации

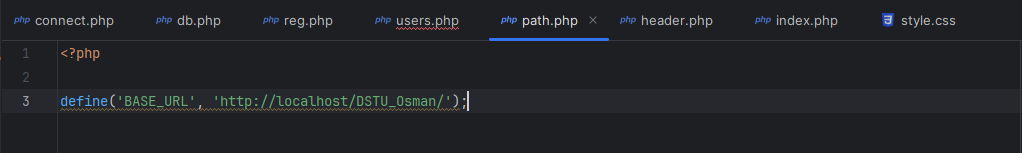


Вызывается функция insert(), которая добавляет новую запись в таблицу "users" с помощью переданных в нее данных, сохраненных в переменной $post. Результат выполнения функции - идентификатор добавленной записи, который сохраняется в переменной $id.

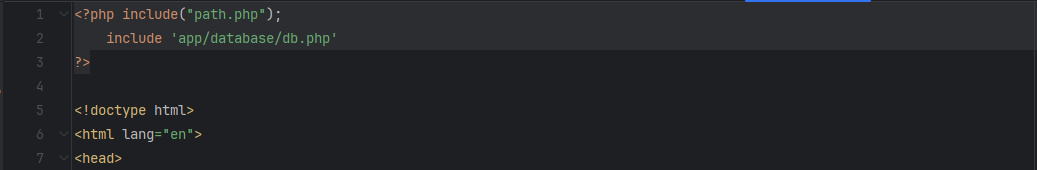
Вызывается функция selectOne(), которая выполняет запрос к таблице "users" и выбирает одну запись, соответствующую условию, переданному в виде ассоциативного массива ['id' => $id]. Результат выполнения функции - массив с данными выбранного пользователя, сохраненный в переменной $user.

Значения из массива $user присваиваются соответствующим ключам массива $\_SESSION, чтобы сохранить информацию о текущем пользователе в сессии.

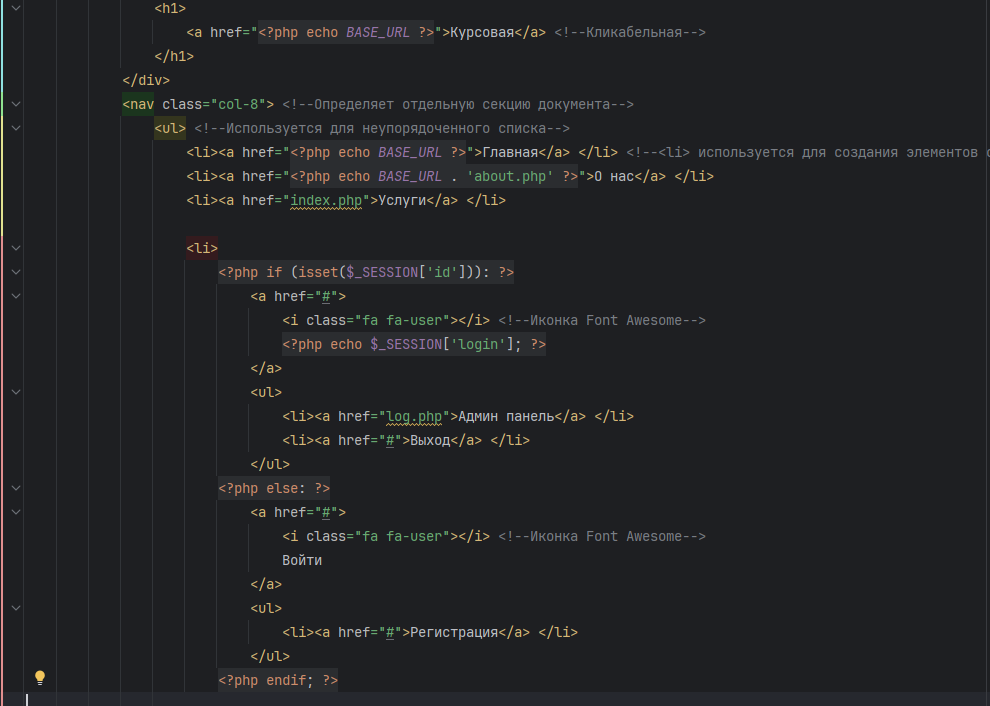
Затем выполняется перенаправление на страницу, указанную в переменной BASE\_URL с помощью функции header().



В файле php указан путь для редиректа пользователя.

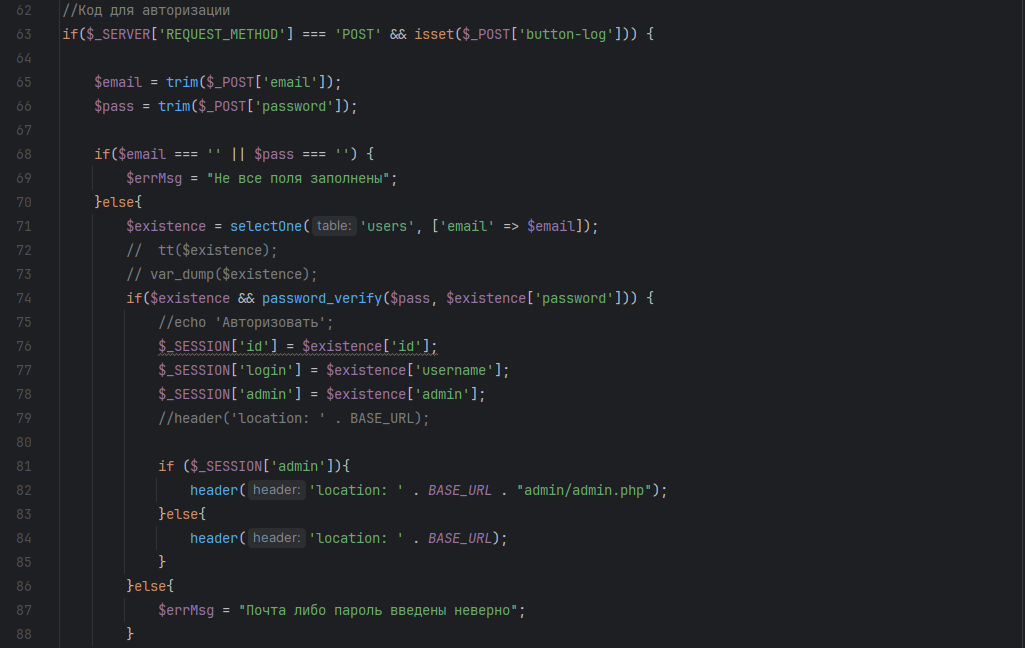


Включаем db.php в index.php.



Этот код отображает разметку верхней части сайта, которая содержит логотип, ссылку на административную панель, ссылку для выхода из административной панели, ссылку для входа в административную панель и ссылку для регистрации нового пользователя.

Логика авторизации



Когда пользователь отправляет данные через форму авторизации (метод POST) и нажимает кнопку "button-log", код выполняет следующие действия:

Получает значение email и password из массива $\_POST.

Проверяет, чтобы оба поля были заполнены. Если хотя бы одно поле не заполнено, устанавливается сообщение об ошибке "Не все поля заполнены".

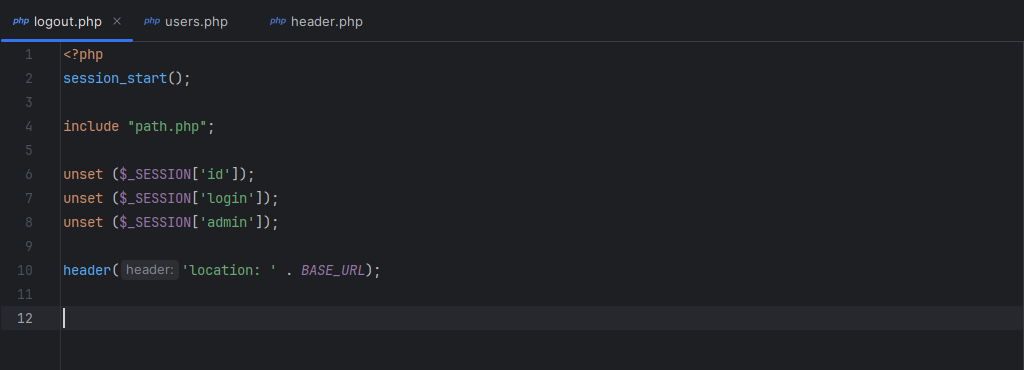
Если оба поля заполнены, код выполняет запрос к базе данных для проверки существования пользователя с указанным email.

Если пользователь с таким email существует и введенный пароль соответствует хэшированному паролю пользователя, код авторизует пользователя и сохраняет его идентификатор, имя пользователя и флаг администратора в сессии.

Затем код перенаправляет пользователя на основную страницу сайта или на страницу администратора в зависимости от того, является ли пользователь администратором.

Если пользователь с указанным email не существует или введенный пароль неверен, устанавливается сообщение об ошибке "Почта либо пароль введены неверно".

Logout, выход и удаление сессии

****

Файл logout.php содержит следующий код:

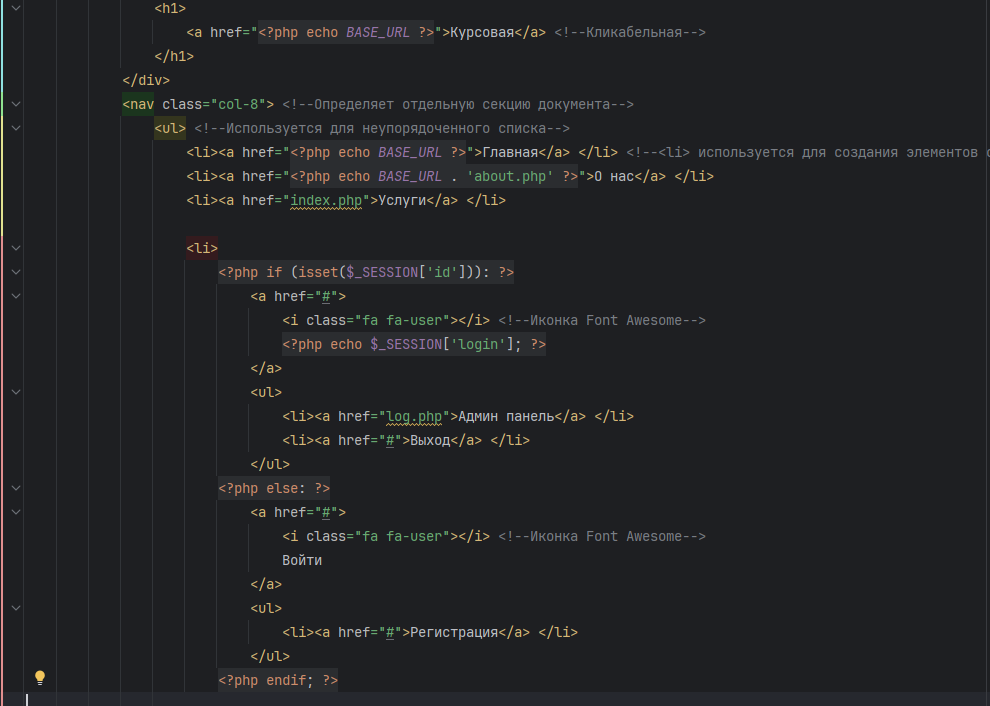
Старт сессии с помощью функции session\_start().

Включение файла path.php, который содержит путь к базовому URL.

Очищение переменных сессии, используя функцию unset(). В данном случае, удаляются переменные $\_SESSION['id'], $\_SESSION['login'] и $\_SESSION['admin'].

Используется функция header() с параметром 'location: ' . BASE\_URL, чтобы выполнить перенаправление пользователя на базовый URL.

Админка



Здесь создается блок с классом "posts" шириной в 9 столбцов. Внутри блока находится строка с двумя кнопками "Add Post" и "Manage Posts" шириной в 3 столбца каждая.

<div class="row add-post">

<form action="created.php" method="post">

<!-- ... (содержание формы) ... -->

</form>

</div>

В этом блоке расположена форма для добавления записей. Форма включает поля для ввода названия статьи, содержания, загрузки файла, выпадающего списка и кнопки для отправки данных на сервер.

<div class="col">

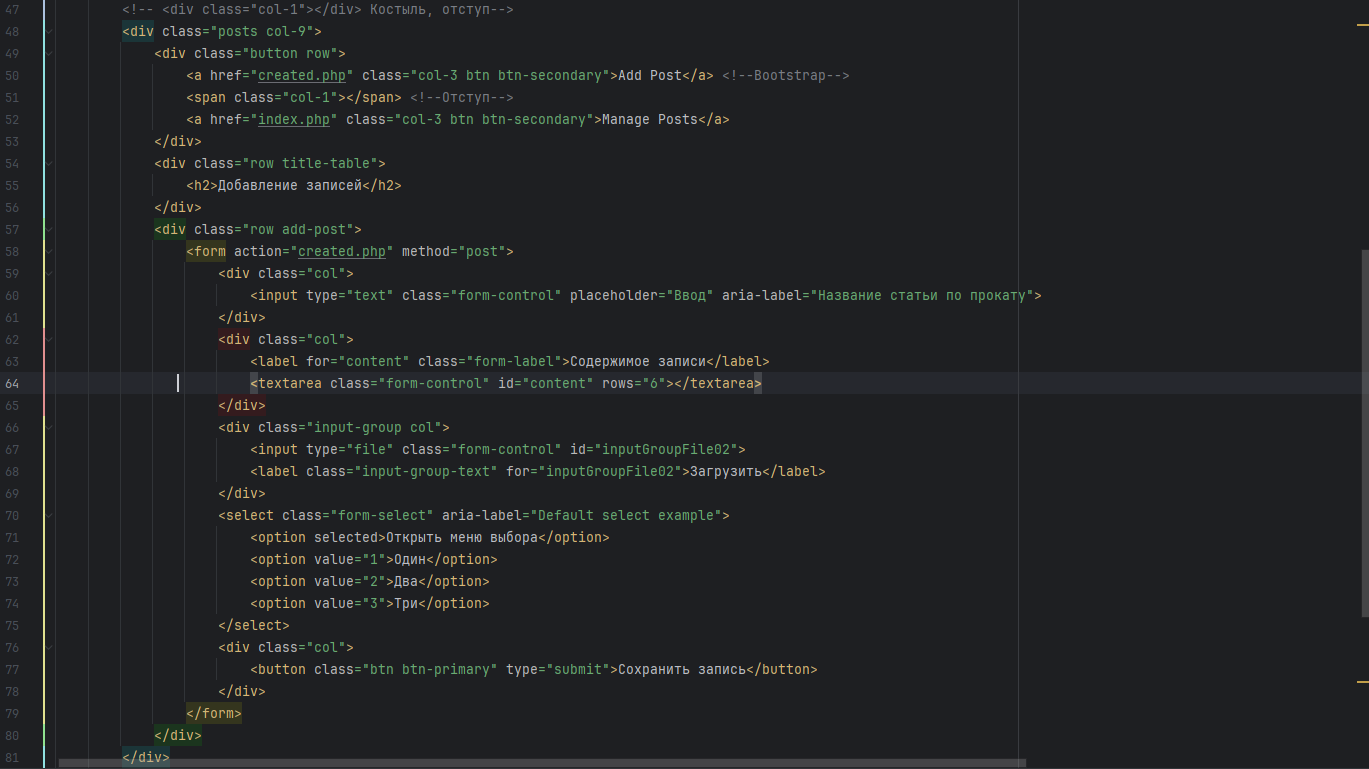
<button class="btn btn-primary" type="submit">Сохранить запись</button>

</div>

Это кнопка для отправки формы на сервер.

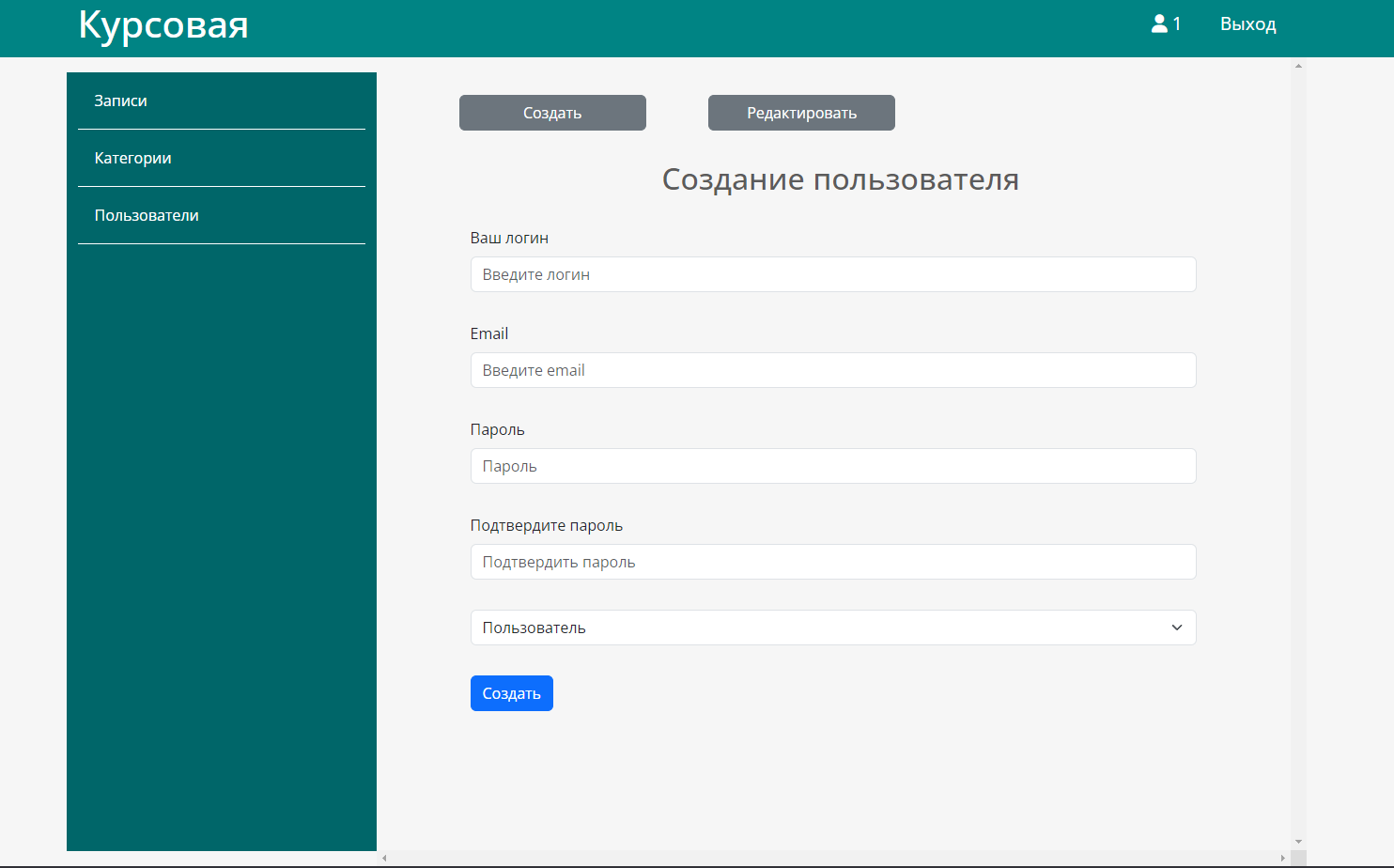
Этот код в основном использует Bootstrap для создания структуры и стилей веб-страницы.

Управление и редактирование записей

  
Простыми словами, этот код представляет собой разметку для страницы, где **админ** может добавлять записи. Используем Bootstrap для стилизации и создания макета.

Примерный код в других файлах для "post", "topics", "users": Логика и структура схожи, но с разными данными и функциональностью в зависимости от контекста.

Итоговый результат



Панель предоставляет возможность легкого добавления, редактирования и удаления контента. Элементы управления хорошо структурированы и легко доступны для пользователя.

Логика админки

Форма создания категории:

Этот PHP-код отвечает за обработку формы создания категории в административной панели.

if($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['topic-create'])) Проверяет, был ли отправлен HTTP POST- запрос и существует ли параметр 'topic-create' в данных формы.

$name = trim($\_POST['name']); $description = trim($\_POST['description']); Извлекает и очищает данные из формы.

if($name === '' || $description === '') { $errMsg = "Заполните все поля!"; } elseif (mb\_strlen($name, 'UTF-8') < 4) { $errMsg = "Категория должна быть длиннее 4 символов!";

Проверяет, что все поля формы заполнены, и имя категории имеет длину более 4 символов.

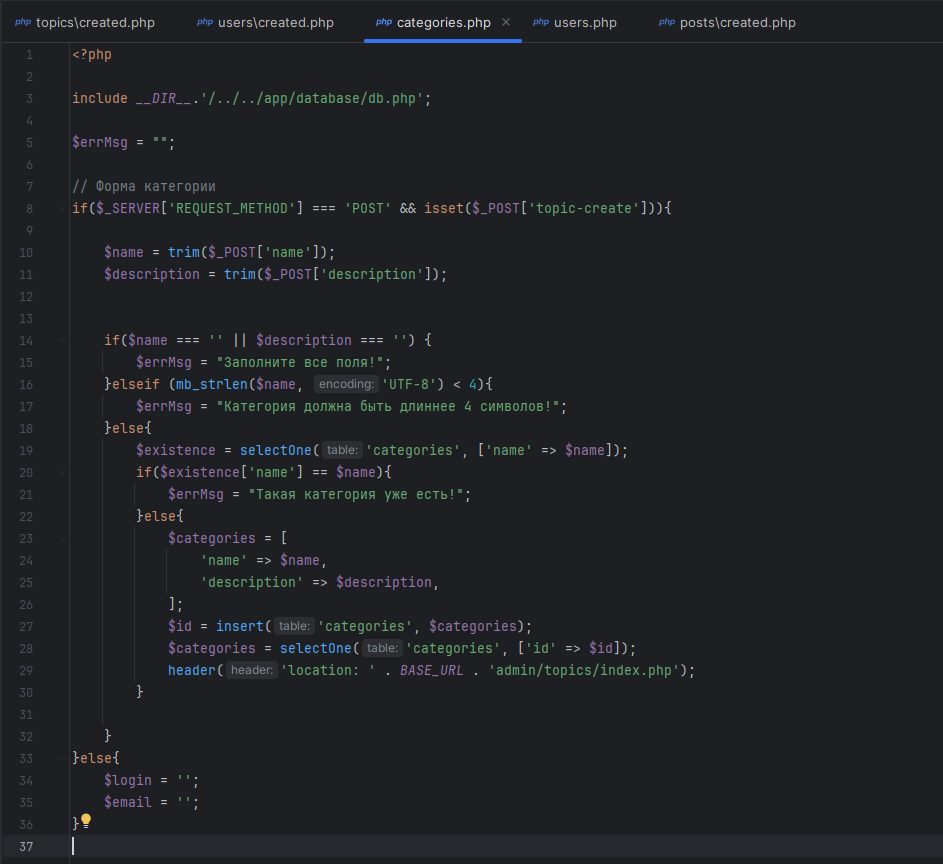
$existence = selectOne('categories', ['name' => $name]); if($existence['name'] == $name){ $errMsg = "Такая категория уже есть!";

Проверяет, не существует ли уже категория с указанным именем.

$categories = [ 'name' => $name, 'description' => $description, ]; $id = insert('categories', $categories);

Вставляет новую категорию в таблицу 'categories' и получает ее идентификатор.

$errMsg, предназначена для вывода сообщений об ошибках в пользовательском интерфейсе.



Сам массив, который выводится

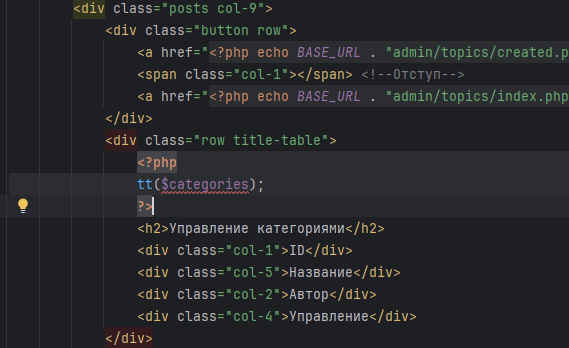


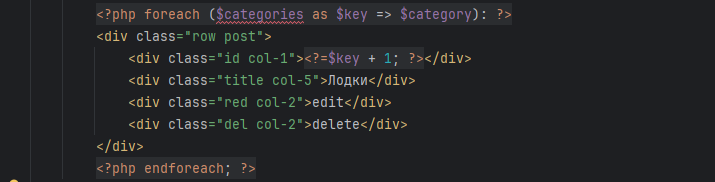
Можно через временные выводы в логи видеть, где программа проходит и где возможны проблемы:

<?php

tt($categories);

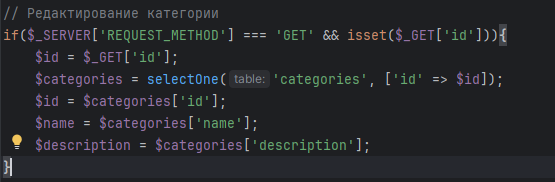
?>





Этот код используется для отображения информации о категориях в виде HTML-блоков внутри контейнера с классом "row post". Для каждой категории создается блок с номером, названием ("Лодки") и кнопками "edit" и "delete". Каждая категория из массива $categories представляется в виде отдельного блока. Кроме того, блоки генерируются внутри цикла foreach, который перебирает каждый элемент массива $categories.

Обновление категории



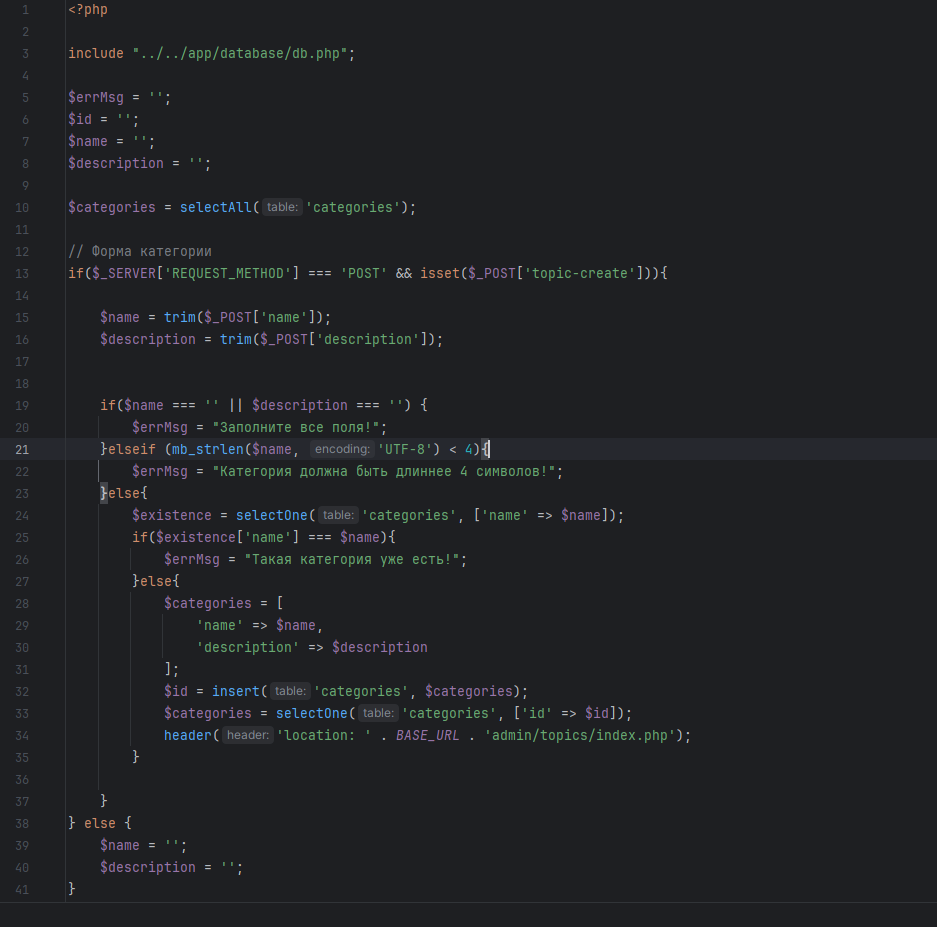
Проверяет, был ли отправлен HTTP GET-запрос и содержит ли запрос параметр id. Если условие выполняется, то:

Получает значение id из параметра запроса.

Выполняет запрос к таблице categories, выбирая запись с указанным идентификатором.

Извлекает значения id, name и description из результата запроса и присваивает их соответствующим переменным $id, $name и $description.

Обработчик формы



Этот код представляет собой обработчик формы для создания и редактирования категорий.

$errMsg = '';

$id = '';

$name = '';

$description = '';

Здесь объявляются переменные, которые будут использоваться для отслеживания ошибок, идентификатора категории, а также имени и описания категории.

$categories = selectAll('categories');

Запрашиваются все категории из базы данных и результат сохраняется в переменной $categories.

if($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['topic-create'])){

// ... обработка данных из формы, валидация, проверка существования, вставка в базу данных ...

} else {

$name = '';

$description = '';

}

Если форма была отправлена методом POST и была нажата кнопка с именем topic-create, то происходит обработка данных из формы (получение, валидация, проверка существования) и вставка новой категории в базу данных. Иначе переменные $name и $description обнуляются.



if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['topic-edit'])) {

// ... обработка данных из формы, валидация, проверка существования, обновление в базе данных ...

}

Аналогичная обработка, как и для создания, но на этот раз для редактирования существующей категории.

if ($\_SERVER['REQUEST\_METHOD'] === 'GET' && !empty($\_GET['id'])) {

// ... получение данных о категории по ID для предзаполнения формы ...

}

Если запрос пришел методом GET и содержит параметр id, то происходит запрос данных о категории с указанным id из базы данных. Эти данные используются для предзаполнения формы перед её отображением.

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['topic-edit'])) {

// ... обновление данных о категории в базе данных ...

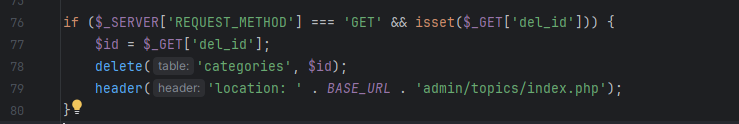
}

После обработки данных из формы при редактировании, происходит обновление данных о категории в базе данных.



Этот код в index.php использует цикл foreach, чтобы для каждой категории из массива $categories создать блок HTML с информацией о категории. В каждом блоке выводится порядковый номер, название категории, а также ссылки для редактирования и удаления категории, которые перенаправляют пользователя на страницу редактирования с соответствующим идентификатором категории.

Удаление категории



Если условие выполняется, то получает значение del\_id из параметра запроса, удаляет запись с соответствующим идентификатором в таблице categories при помощи функции delete, а затем перенаправляет пользователя обратно на страницу admin/topics/index.php.

Материал:

http://www.mysql.ru/docs/man/What-is.html

https://www.w3schools.com/php/php\_mysql\_delete.asphttps://oracleplsql.ru/functions-mysql.html

<https://www.php.net/manual/ru/function.is-array.php>

<https://stackoverflow.com/questions/2418473/difference-between-require-include-require-once-and-include-once>

<https://qna.habr.com/q/502702>

https://prowebmastering.ru/php-pdo-start.html

GitHub:

<https://github.com/Light278/MyWork>

Важно отметить, что данный проект, созданный студентом У-134 Рамазановым Османом Мурадовичем, предназначен исключительно для демонстрации учебных достижений и не подлежит коммерческому использованию без соответствующего согласования с автором.