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.

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.

Инфологическая модель базы данных (ИМ БД) для системы проката плавательных средств

**Таблица: Users (Пользователи)**

id (целое число, первичный ключ, автоинкремент)

admin (целое число, тип данных tinyint, для определения статуса администратора)

username (строка, максимальная длина 255 символов)

email (строка, индекс, максимальная длина 255 символов)

password (строка, максимальная длина 255 символов)

created (метка времени, отражающая момент создания записи)

Эта таблица "Users" предназначена для хранения информации о пользователях. Каждый пользователь имеет уникальный идентификатор (id), статус администратора (admin), имя пользователя (username), адрес электронной почты (email), хэшированный пароль (password), и метку времени создания (created). Идентификатор будет уникален для каждой записи благодаря автоинкрементному свойству.

**Таблица: Categories (Категории)**

id (целое число, первичный ключ, автоинкремент)

name (строка, индекс, максимальная длина 125 символов)

description (текст, описание категории)

Эта таблица "Categories" предназначена для хранения информации о различных категориях плавательных средств. Каждая категория имеет уникальный идентификатор (id), название (name) с ограничением по длине, и текстовое описание (description). Идентификатор будет уникален для каждой записи благодаря автоинкрементному свойству.

**Таблица: Posts (Посты)**

id (целое число, первичный ключ, автоинкремент)

id\_user (целое число, внешний ключ, связанный с пользовательской таблицей)

title (строка, максимальная длина 255 символов)

img (строка, путь к изображению, максимальная длина 255 символов)

content (текст, содержание поста)

status (целое число, может быть использовано как булев тип или для других статусов)

id\_topic (целое число, внешний ключ, связанный с таблицей категорий)

created\_date (дата и время создания поста, значение по умолчанию - текущая метка времени)

Эта таблица "Posts" предназначена для хранения информации о постах. Каждый пост имеет уникальный идентификатор (id), связь с пользователем (id\_user), заголовок (title), изображение (img), содержание (content), статус (status), категорию (id\_topic), и дату создания (created\_date). Внешние ключи id\_user и id\_topic связываются с соответствующими таблицами пользователей и категорий.

**Таблица: Price (Цена)**

id: уникальный идентификатор цены для плавательного средства.

gear\_name: название плавательного средства.

price: цена за аренду этого плавательного средства в формате DECIMAL(10, 2).

Эта таблица предназначена для хранения информации о ценах на аренду различных плавательных средств. Каждая запись в таблице содержит уникальный идентификатор, название плавательного средства и его цену за аренду. Эти данные могут быть использованы в приложении для отслеживания цен и предоставления пользователям актуальной информации о стоимости аренды.

**Таблица: boat\_information (Информация о плавательных средствах)**

id (Целое число, первичный ключ, автоинкремент)

Описание: Уникальный идентификатор каждой записи о плавательном средстве.

Тип данных: Целое число (int).

name (Строка переменной длины, VARCHAR(255))

Описание: Название плавательного средства.

Тип данных: Строка переменной длины (varchar) с максимальной длиной 255 символов.

category\_id (Целое число, Индекс)

Описание: Идентификатор категории, к которой принадлежит плавательное средство.

Тип данных: Целое число (int).

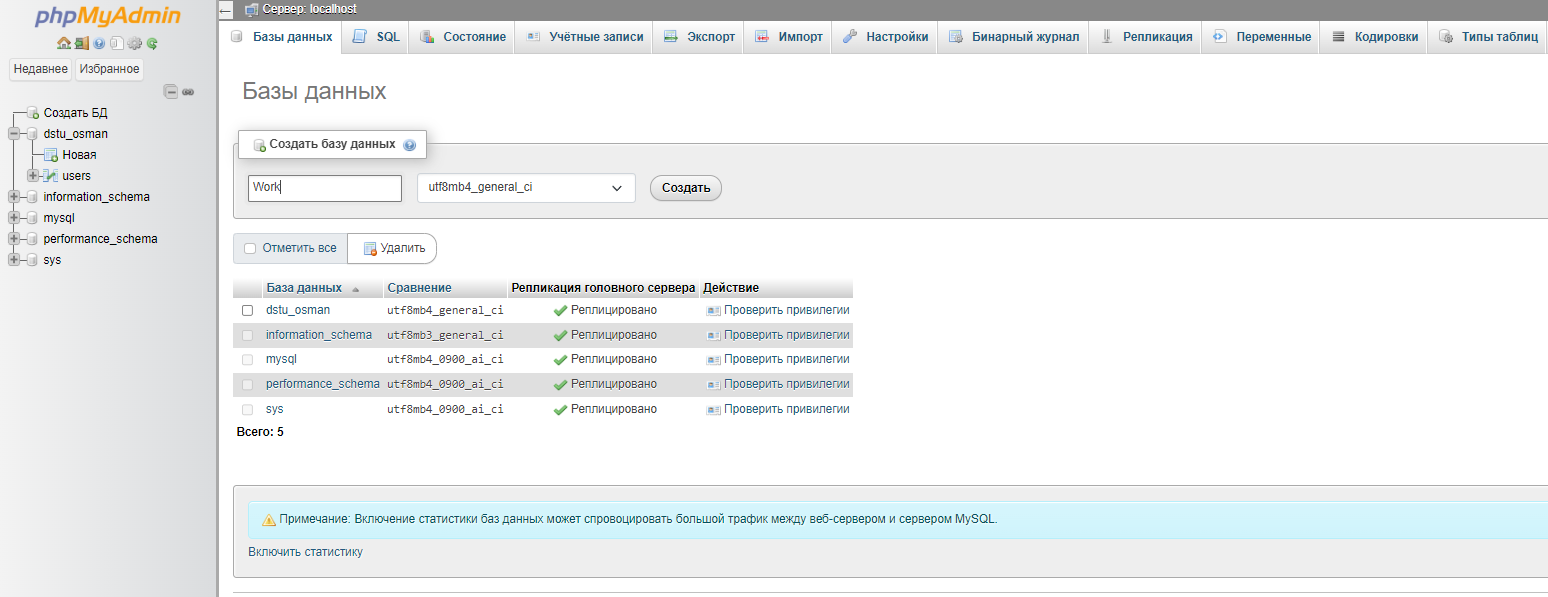
MySQL

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

Для этого необходимо:

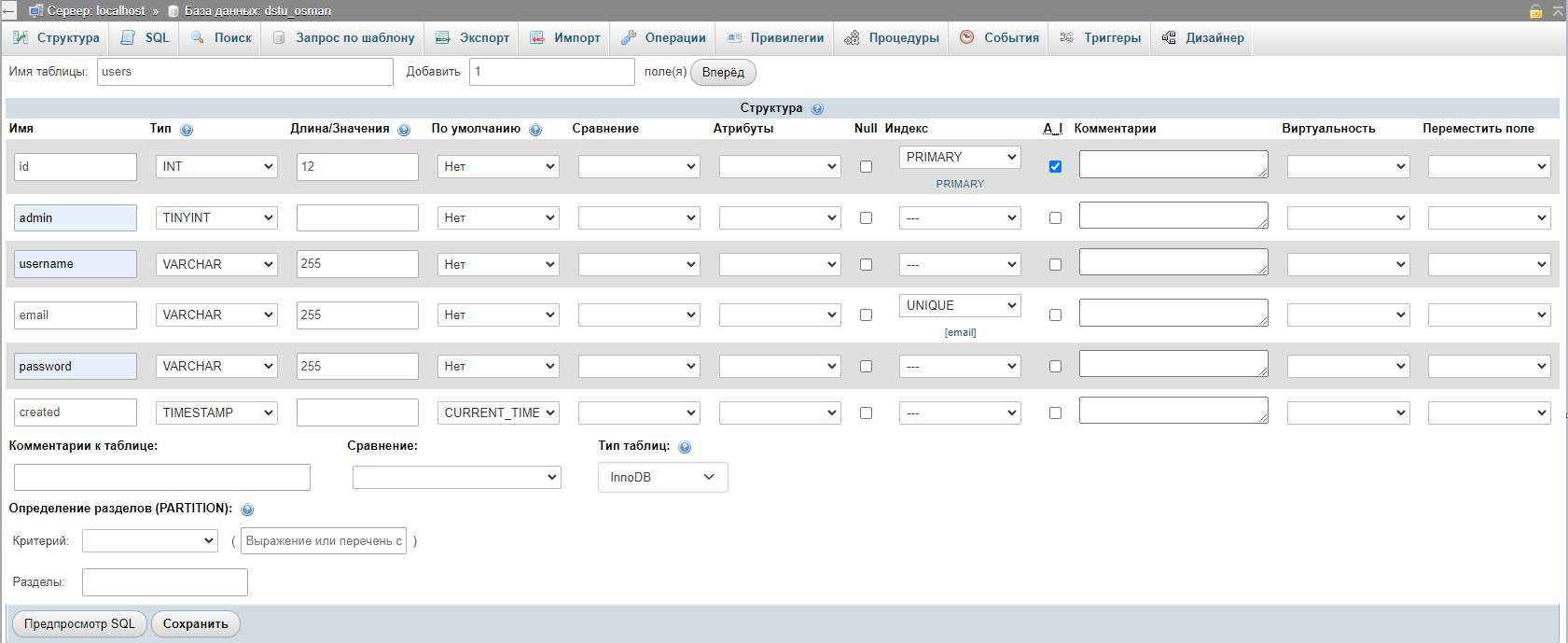
1. Открыть веб-браузер и введите в адресной строке <http://localhost/phpmyadmin/>
2. На странице входа ввести логин и пароль. Обычно логин – это «root», а пароль может быть пустым или установлен по умолчанию во время установки **AMPPS**.
3. Нажимаем на кнопку «Вход» или нажмите Enter, чтобы войти в **phpMyAdmin**.

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.

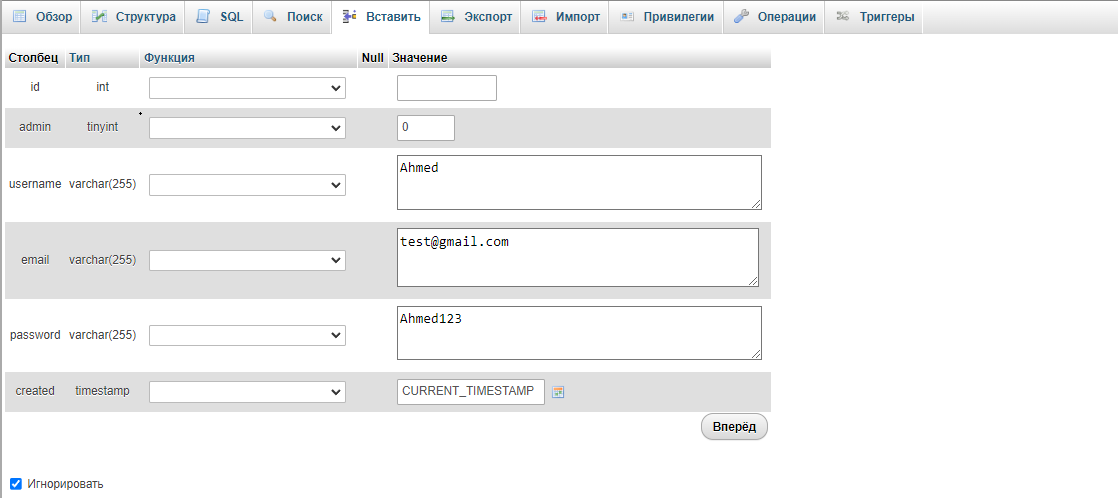
TIMESTAMP – Тип данных для хранения даты и времени. Данные хранятся в виде количества секунд, прошедших с начала «эпохи Юникса».

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.

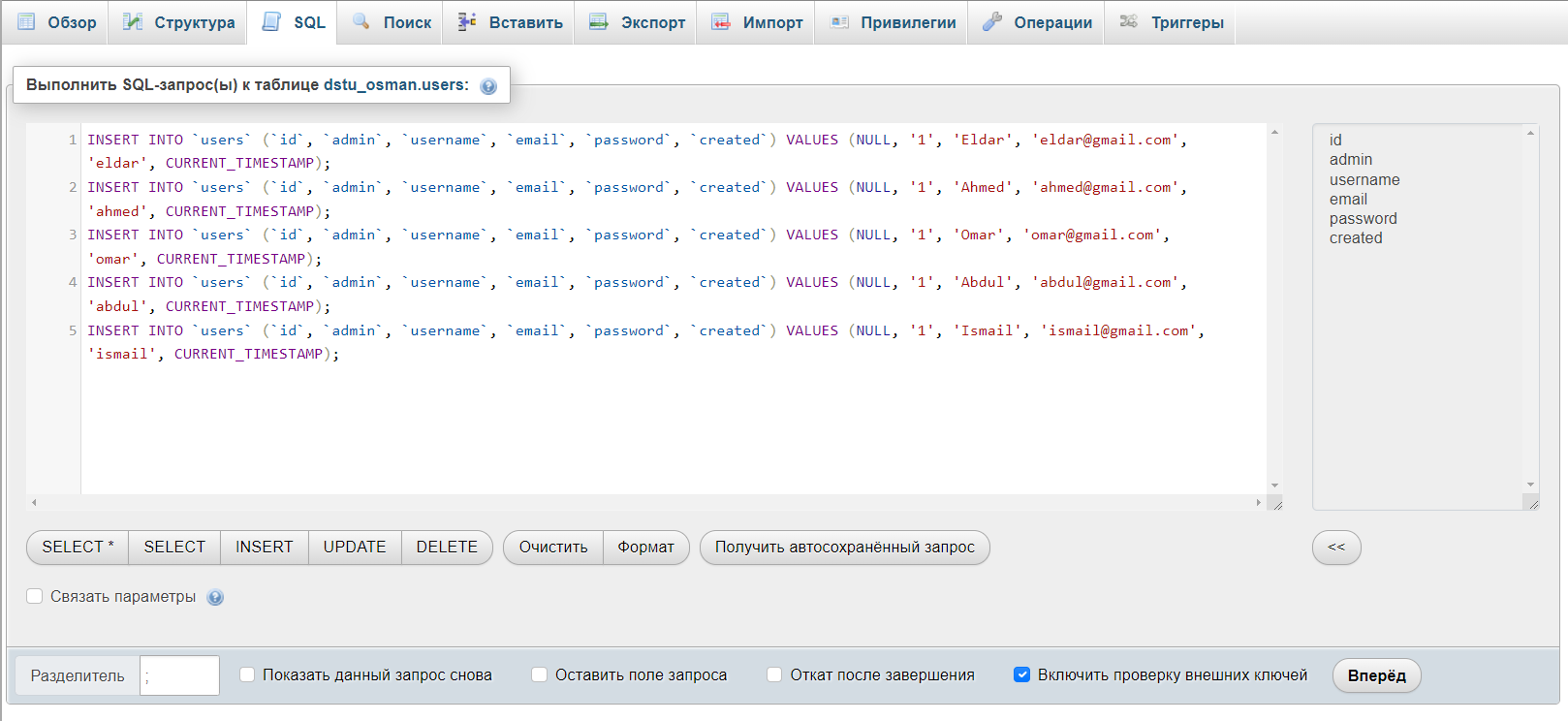
Добавляем пользователя через "Вставить" (Insert)

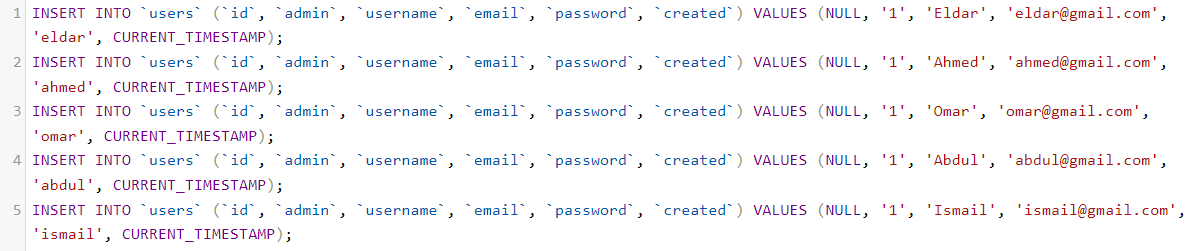


Чтобы вставить значение пользователя в таблицу phpMyAdmin через SQL, вам потребуется выполнить следующие действия:

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





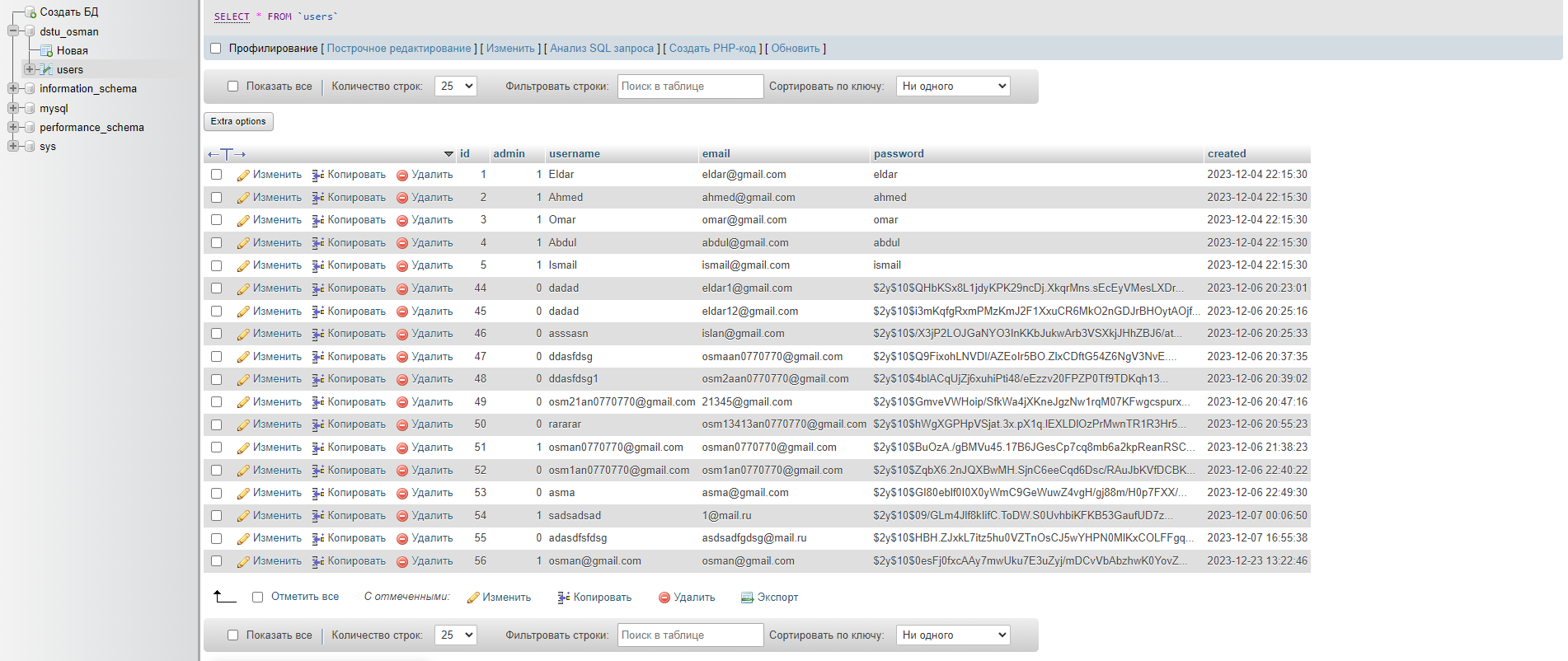
После краткого анализа принципа работы **PhpMyAdmin** необходимо создать сайт и подключить к нему базу данных.

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

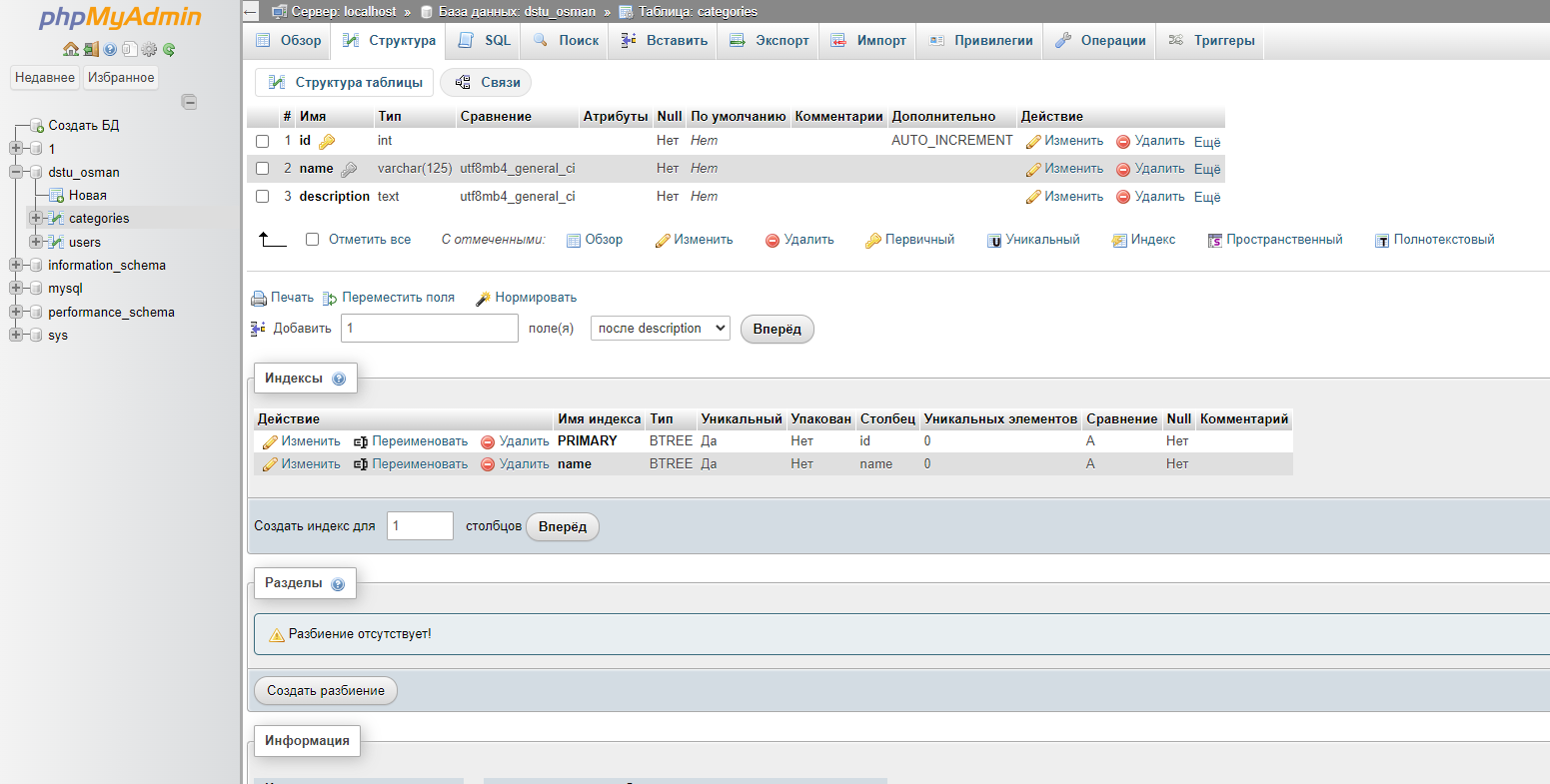


Таблица категорий в базе данных является ключевым компонентом для организации и классификации данных. Она представляет собой структурированное хранилище, где каждая запись относится к определенной категории, обладающей уникальным идентификатором.

Таблица постов

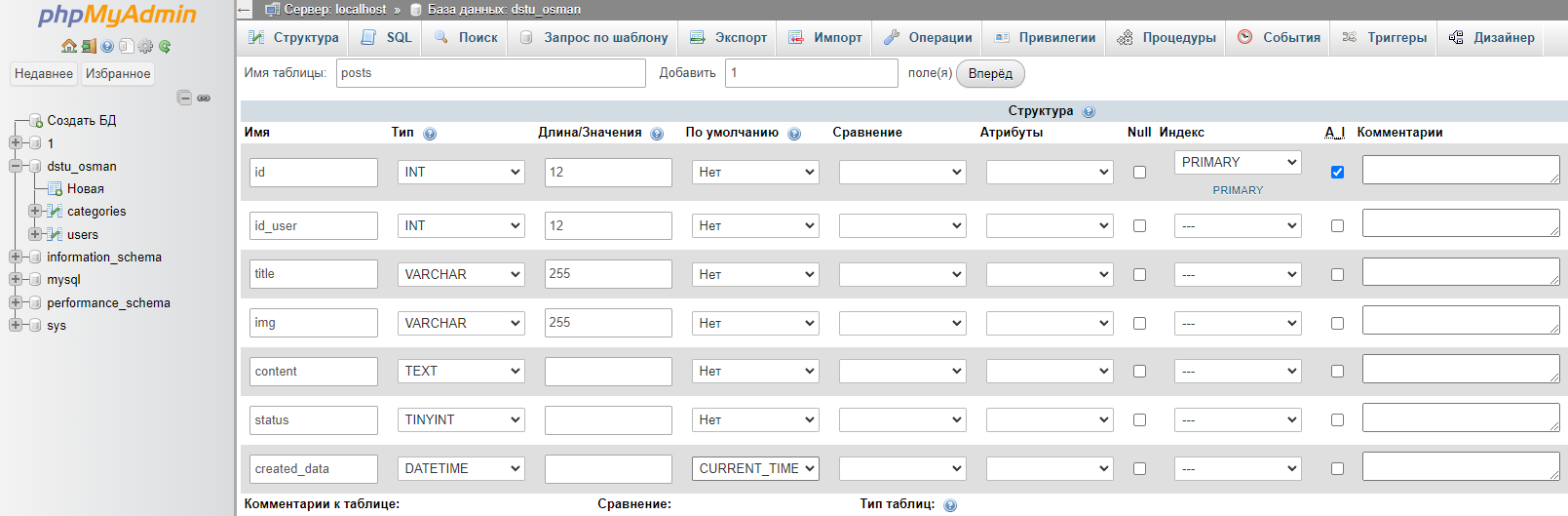


Таблица постов по прокату лодочных средств в базе данных предназначена для хранения информации о доступных лодках, их характеристиках и условиях аренды. Она позволяет удобно организовать информацию о каждом конкретном посте проката, а также обеспечить быстрый доступ к нужным данным.

Таблица цены

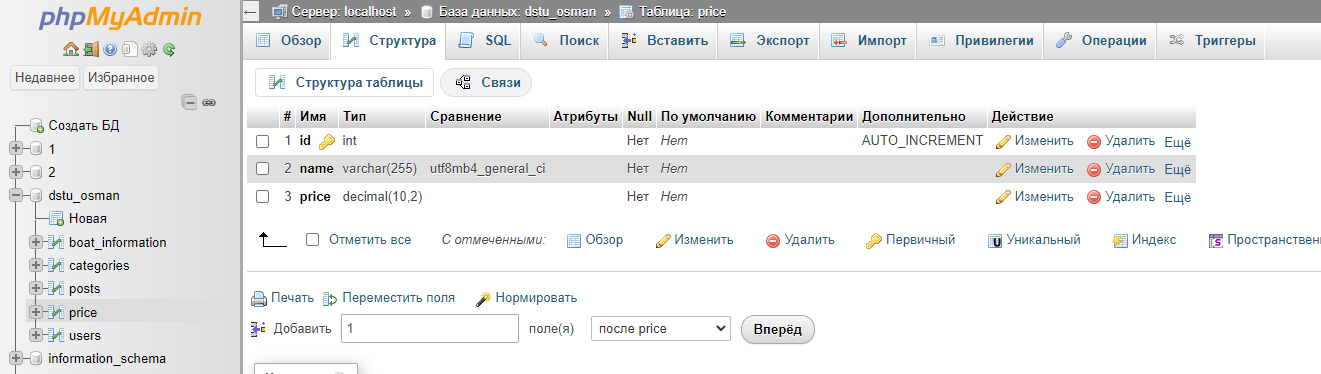


Таблица цен в базе данных может быть полезной для хранения и управления информацией о ценах на различные товары, услуги или другие объекты, которые ваше веб-приложение или система обрабатывает. В коде представлена таблица с именем price.

Таблица информации о плавательных средствах

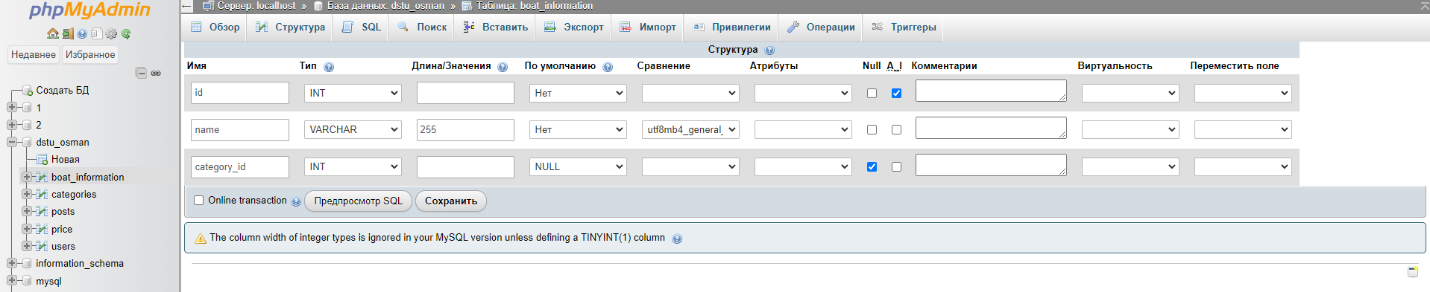
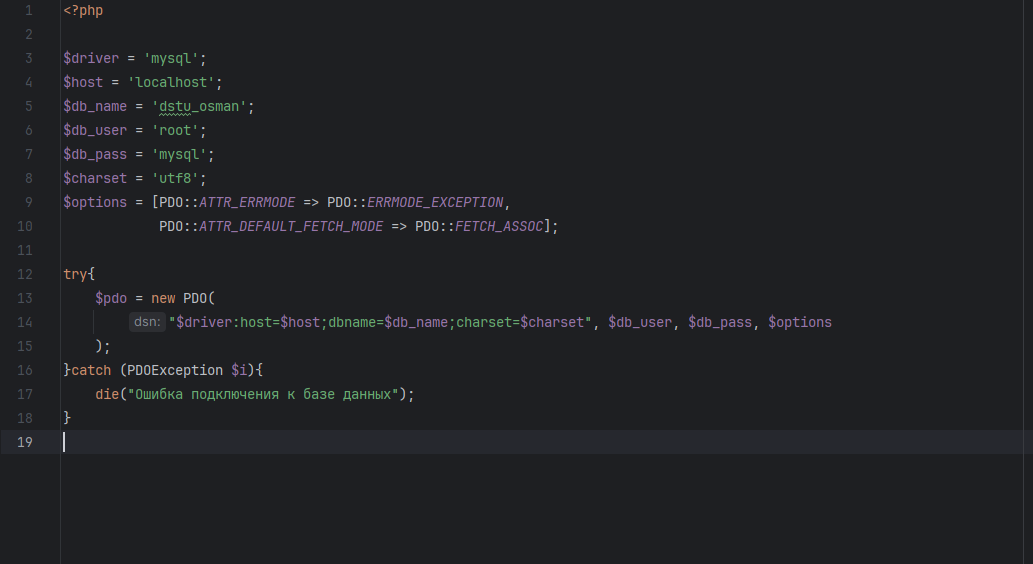


Таблица "Информация о плавательных средствах" в базе данных может быть полезной для хранения и управления данными о различных плавательных средствах. В коде представлена таблица с именем boat\_information.

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.

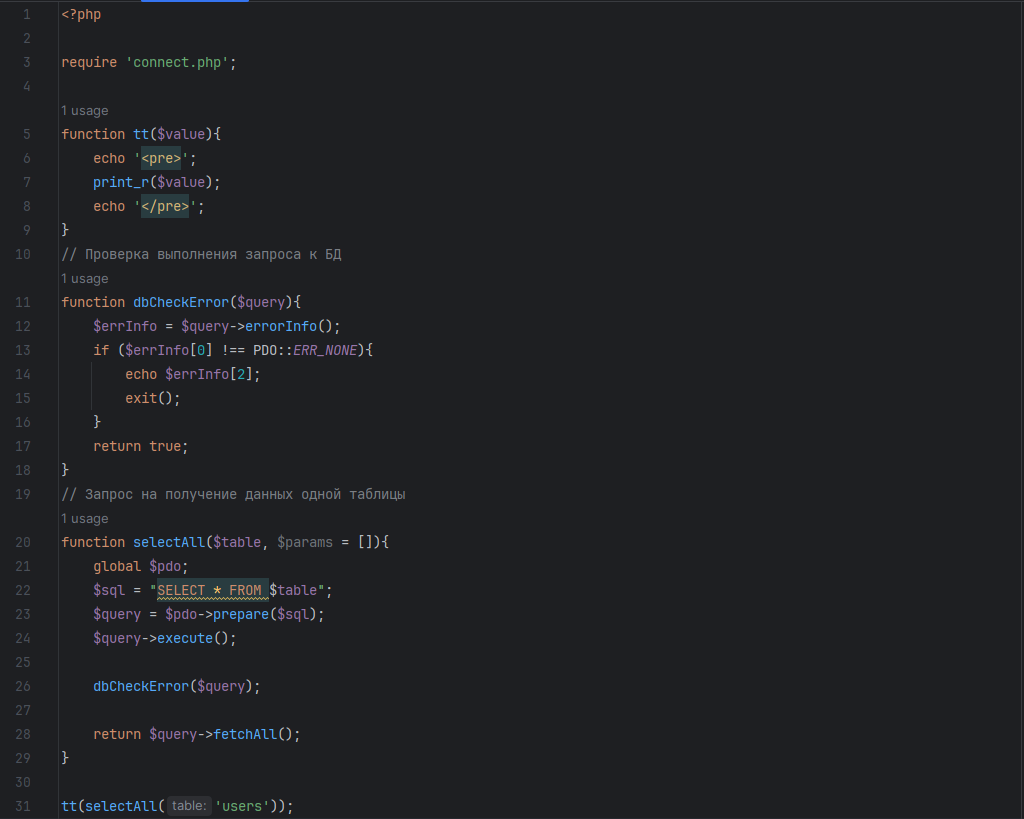
Конструкция try catch - Это специальная инструкция из двух блоков, которая позволяет перехватить все исключения и их обработать. Любые исключения, которые будут выброшены кодом, расположенным внутри этого блока, будут перехвачены и переданы во второй блок. Если ошибки не было, то этот блок пропускается.

$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

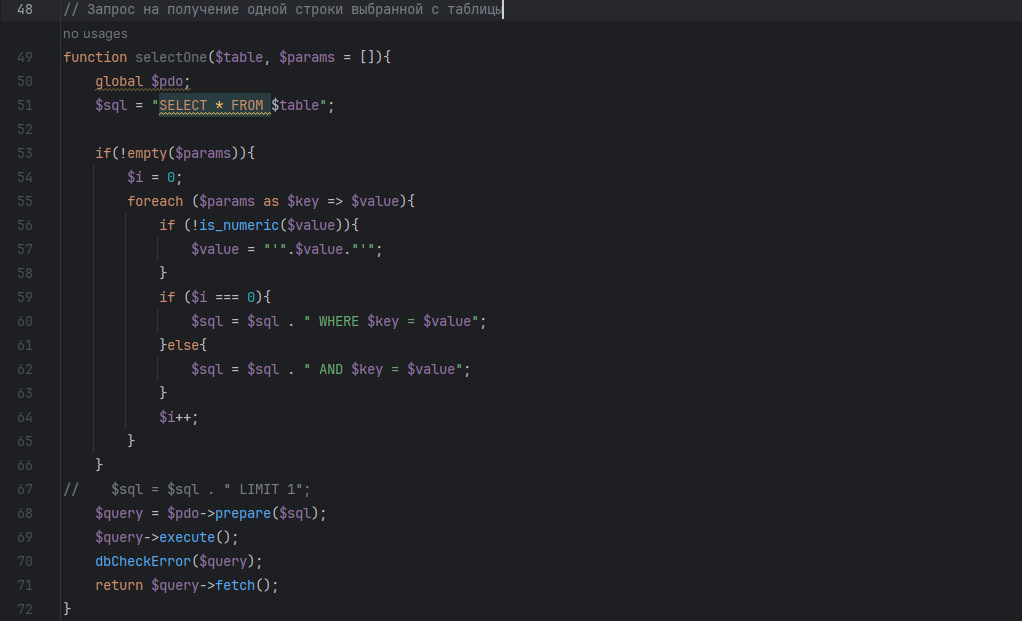
$query = $pdo->prepare($sql); - Обращение к экземпляру класса PDO из connect. Выполняется подготовка.

$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 = [

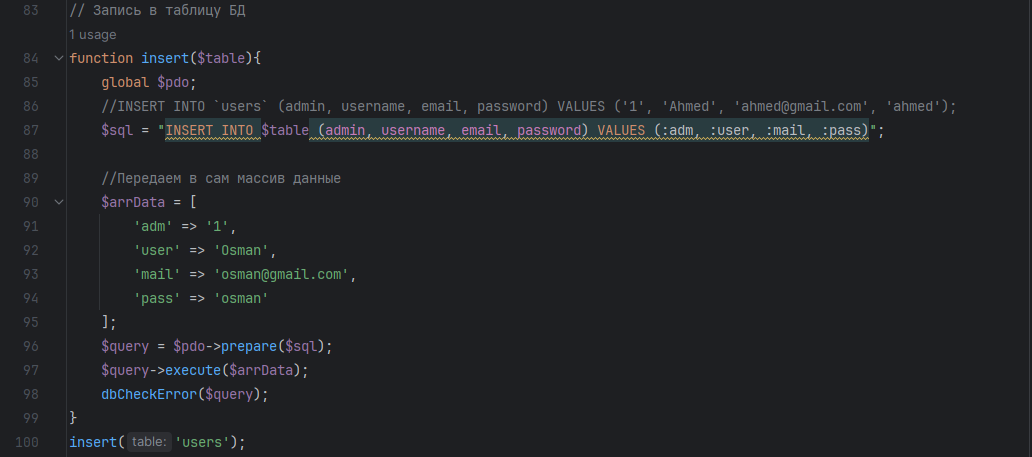
'admin' => 1,

'username' => 'Eldar'

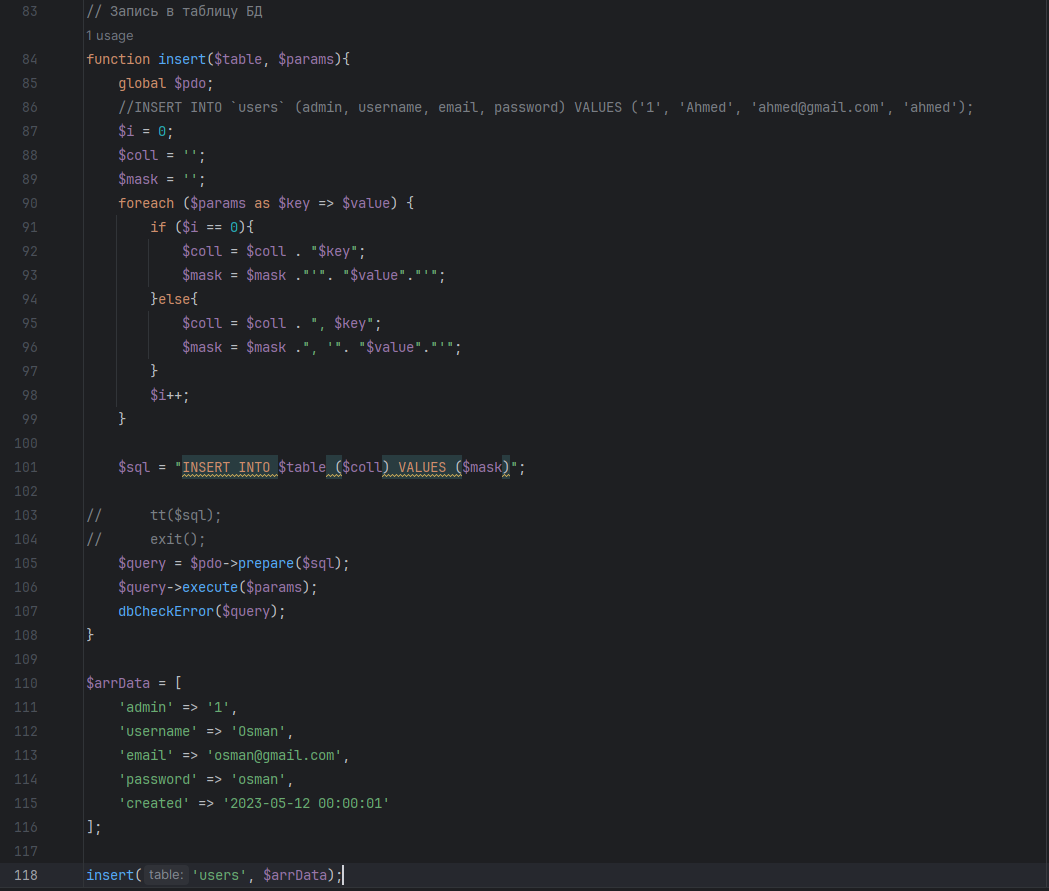
];

$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.

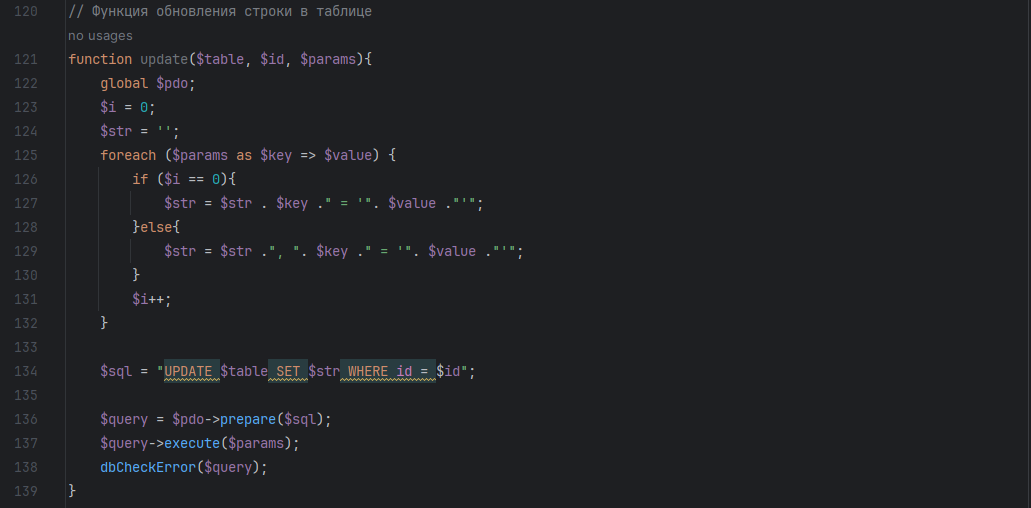
Затем происходит перебор массива $params. Для каждой пары ключ-значение создается строка, которая добавляется к переменным $coll и $mask. Если это первый элемент массива, то пара будет добавлена без запятой, в противном случае пара будет добавлена с запятой перед ней. Значение каждого элемента массива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

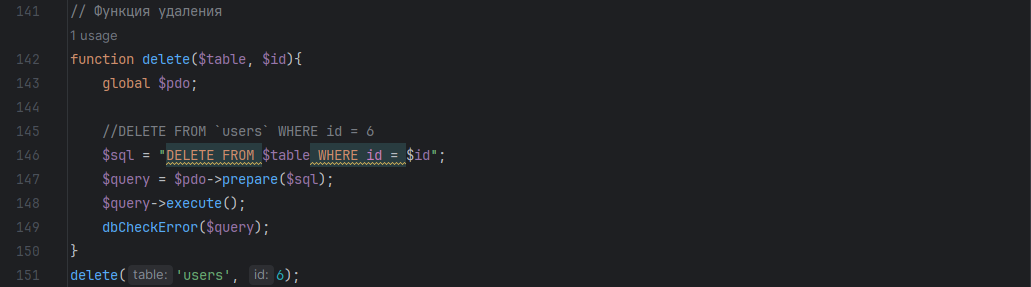


Функция update принимает три параметра: $table, $id и $params.

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

$id - идентификатор записи, которую нужно обновить.

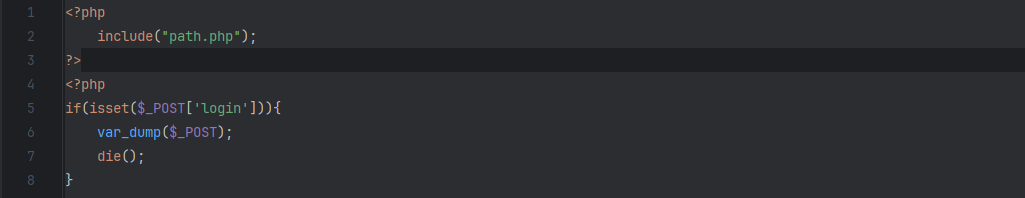
$params - массив, содержащий пары ключ-значение, где ключи - названия полей, а значения - новые значения полей, которые нужно обновить.$params.



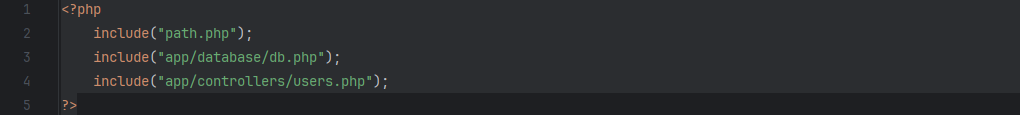
Функция delete() используется для удаления записей из таблицы базы данных. Она принимает два аргумента - название таблицы $table и идентификатор записи $id, которую нужно удалить.

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

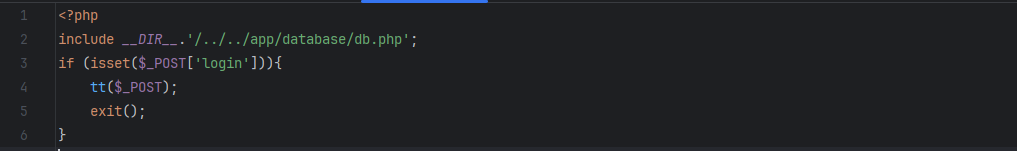
Форма регистрации



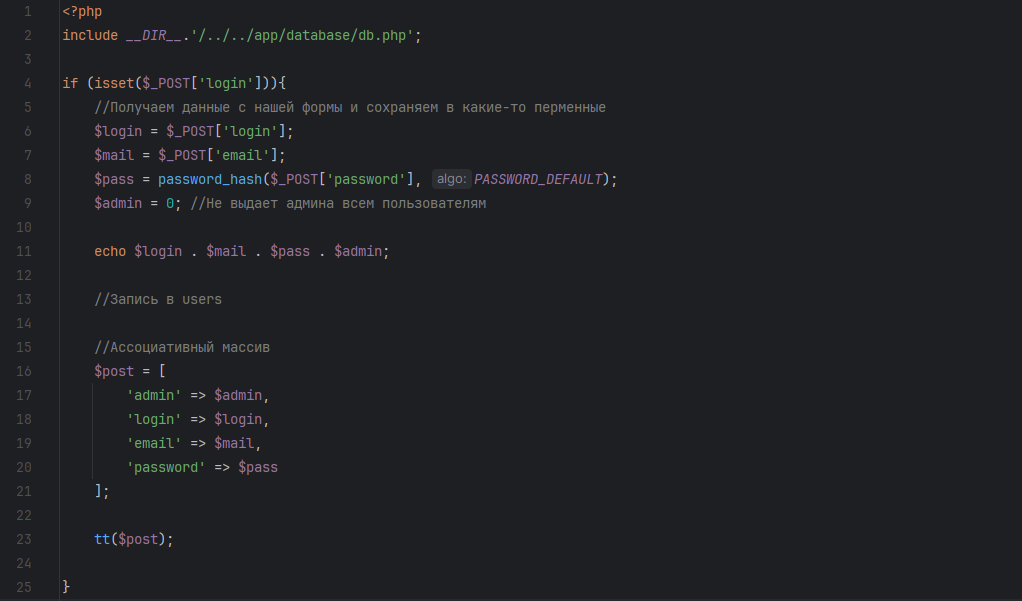
Проверка. Если у нас в супер-глобальном массиве $\_POST есть заполненные данные "login", то тогда выводит абсолютно весь массив.



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



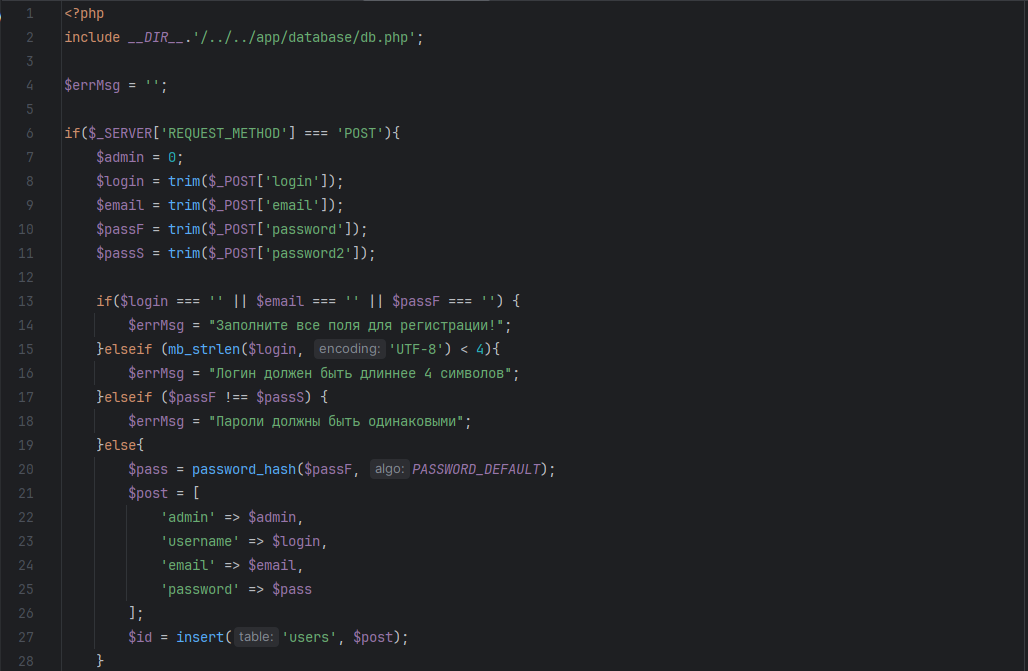
Если файлы не инклудятся - можно прописать адрес через/ *DIR* / - у меня так заработало: include \_\_DIR\_\_.'/../../app/database/db.php'; можно тоже и через set\_include\_path() и потом уже include.



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



Проверка данных с формы регистрации



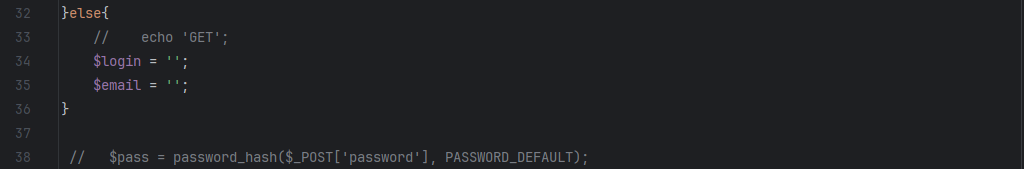
Проверки выполняются для каждого поля, такие как проверка на пустое значение, на соответствие определенному формату и т. д. При несоответствии условиям, выводится соответствующее сообщение.

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.



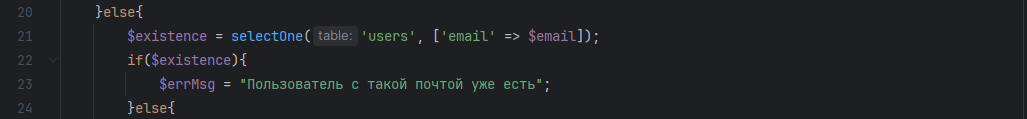
Если условие истинно (т.е. метод запроса не равен "POST"), то инициализируются переменные $login и $email пустыми значениями.

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

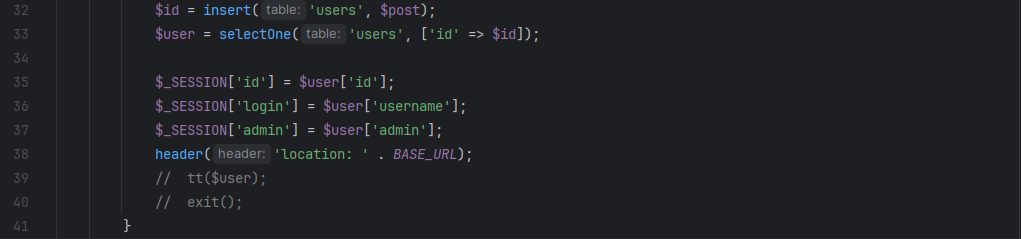


Проверяем наличие соответствующей записи о пользователе в базе данных.

$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

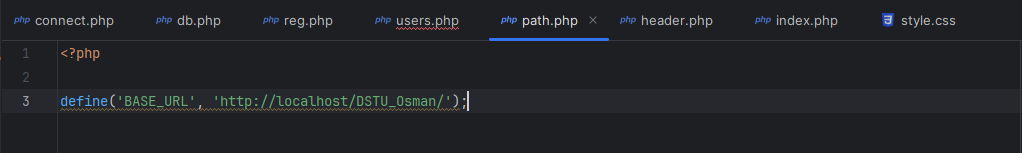


Вызывается функция insert(), которая добавляет новую запись в таблицу "users" с помощью переданных в нее данных, сохраненных в переменной $post. Результат выполнения функции - идентификатор добавленной записи, который сохраняется в переменной $id.

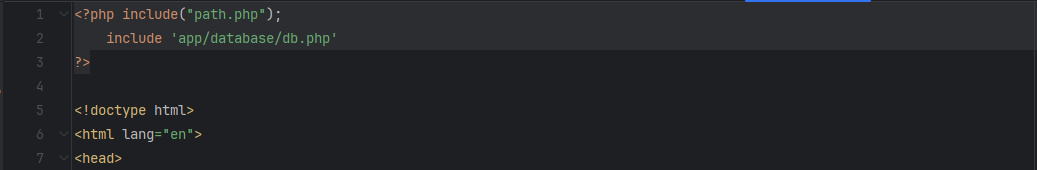
Вызывается функция selectOne(), которая выполняет запрос к таблице "users" и выбирает одну запись, соответствующую условию, переданному в виде ассоциативного массива ['id' => $id]. Результат выполнения функции - массив с данными выбранного пользователя, сохраненный в переменной $user.

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

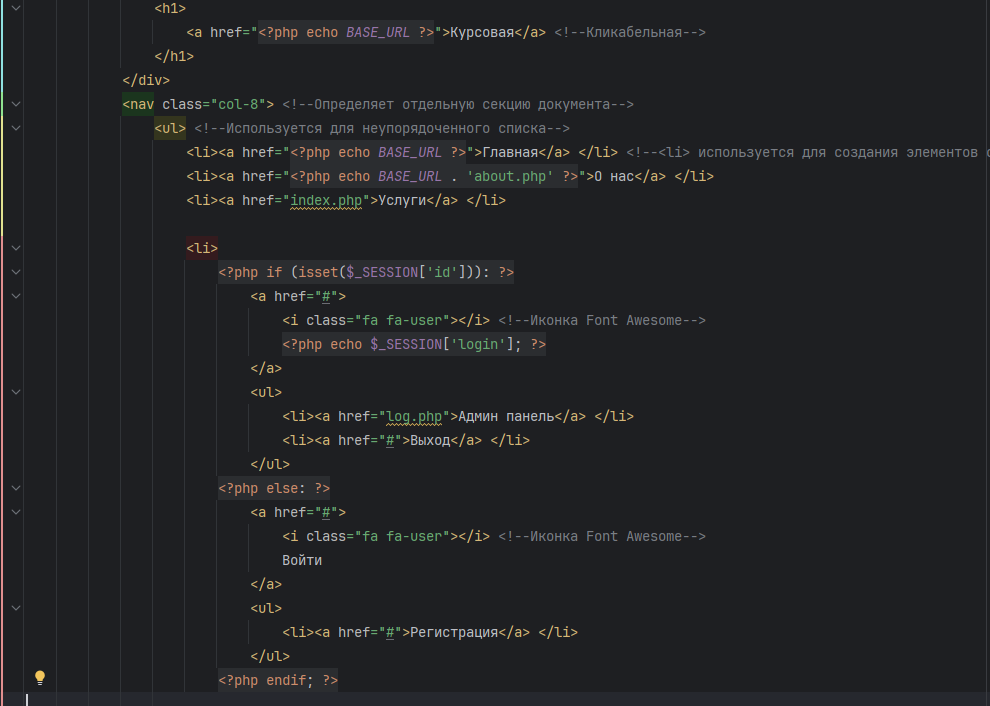
Затем выполняется перенаправление на страницу, указанную в переменной BASE\_URL с помощью функции header().



В файле php указан путь для редиректа пользователя.

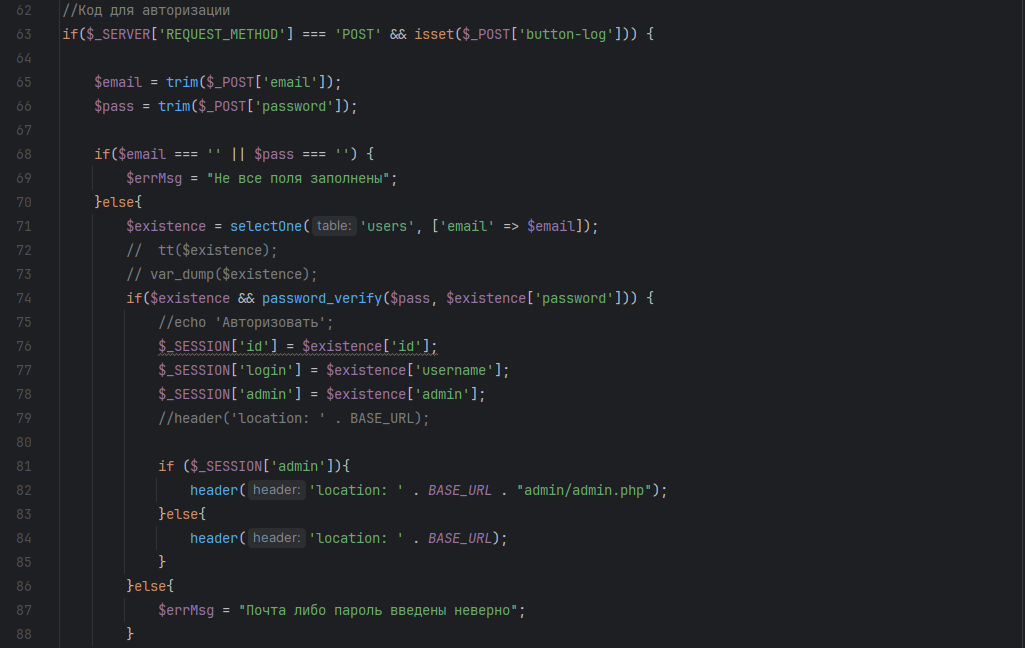


Включаем db.php в index.php.



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".

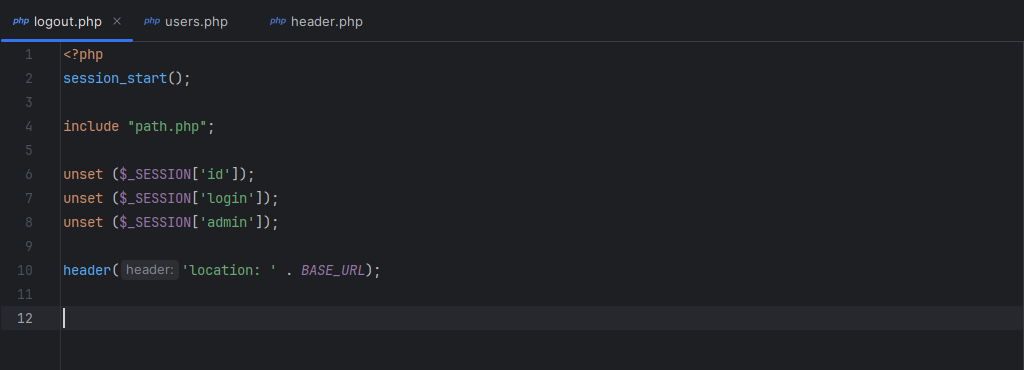
Если оба поля заполнены, код выполняет запрос к базе данных для проверки существования пользователя с указанным email.

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:

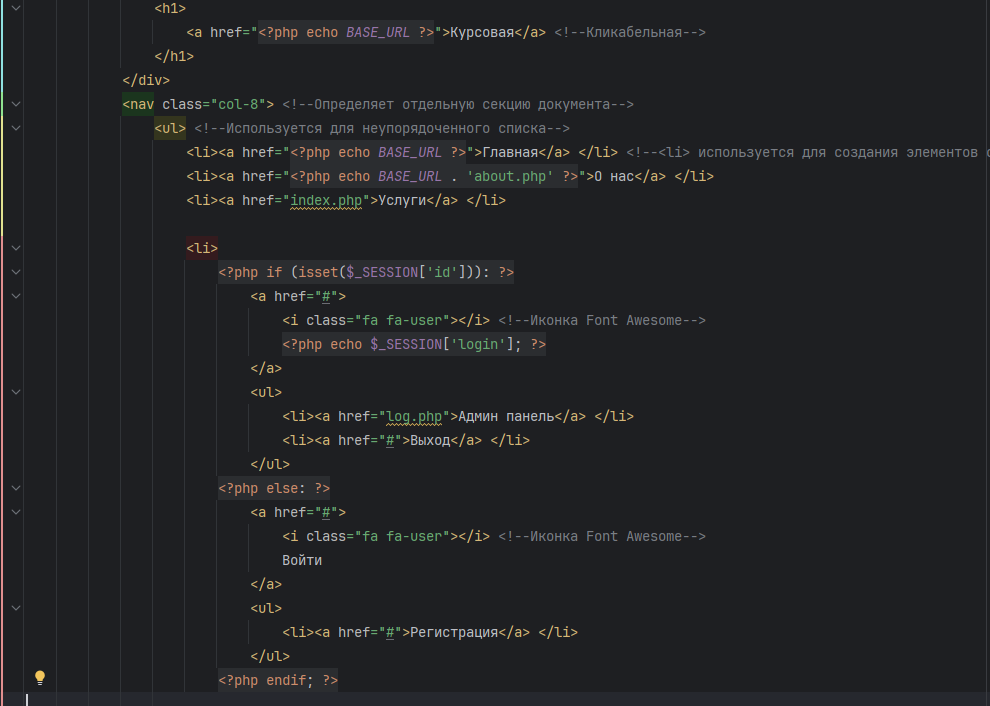
Старт сессии с помощью функции session\_start().

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">

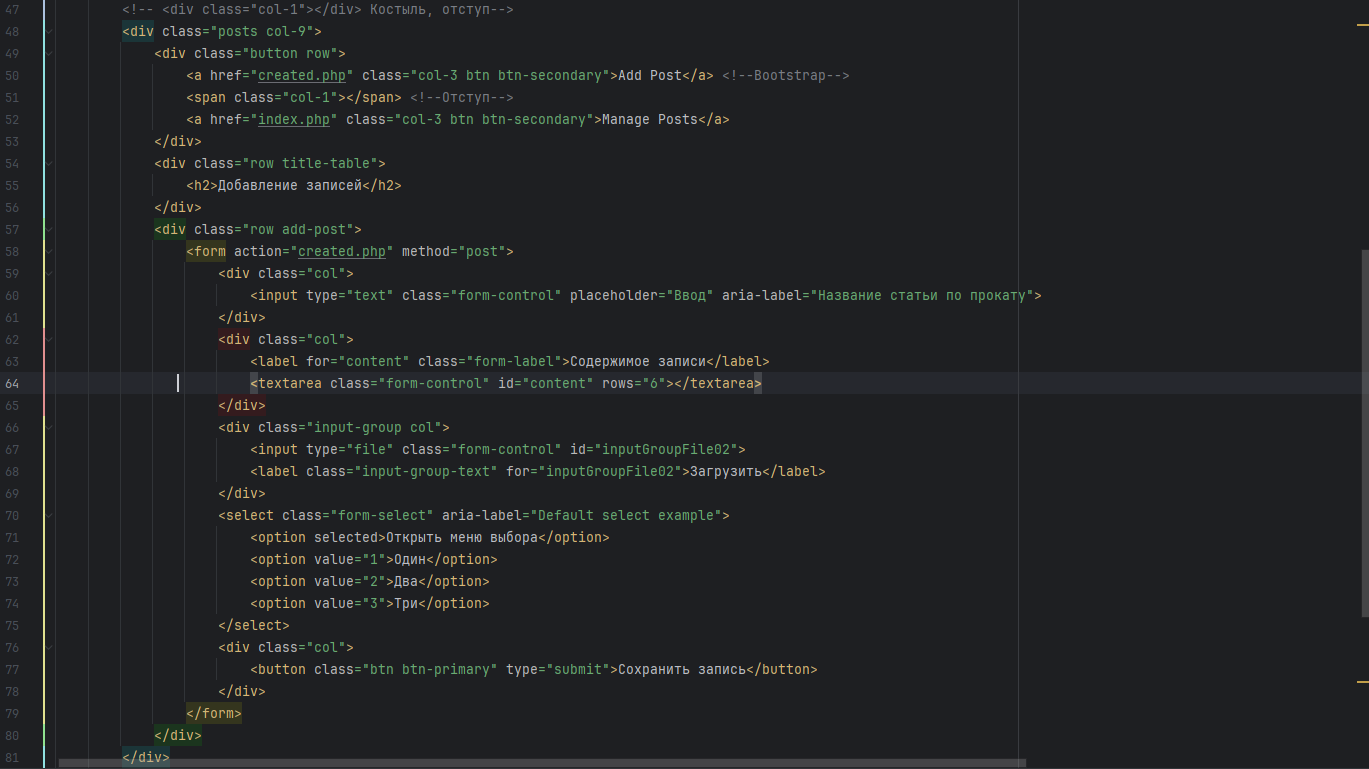
<button class="btn btn-primary" type="submit">Сохранить запись</button>

</div>

Это кнопка для отправки формы на сервер.

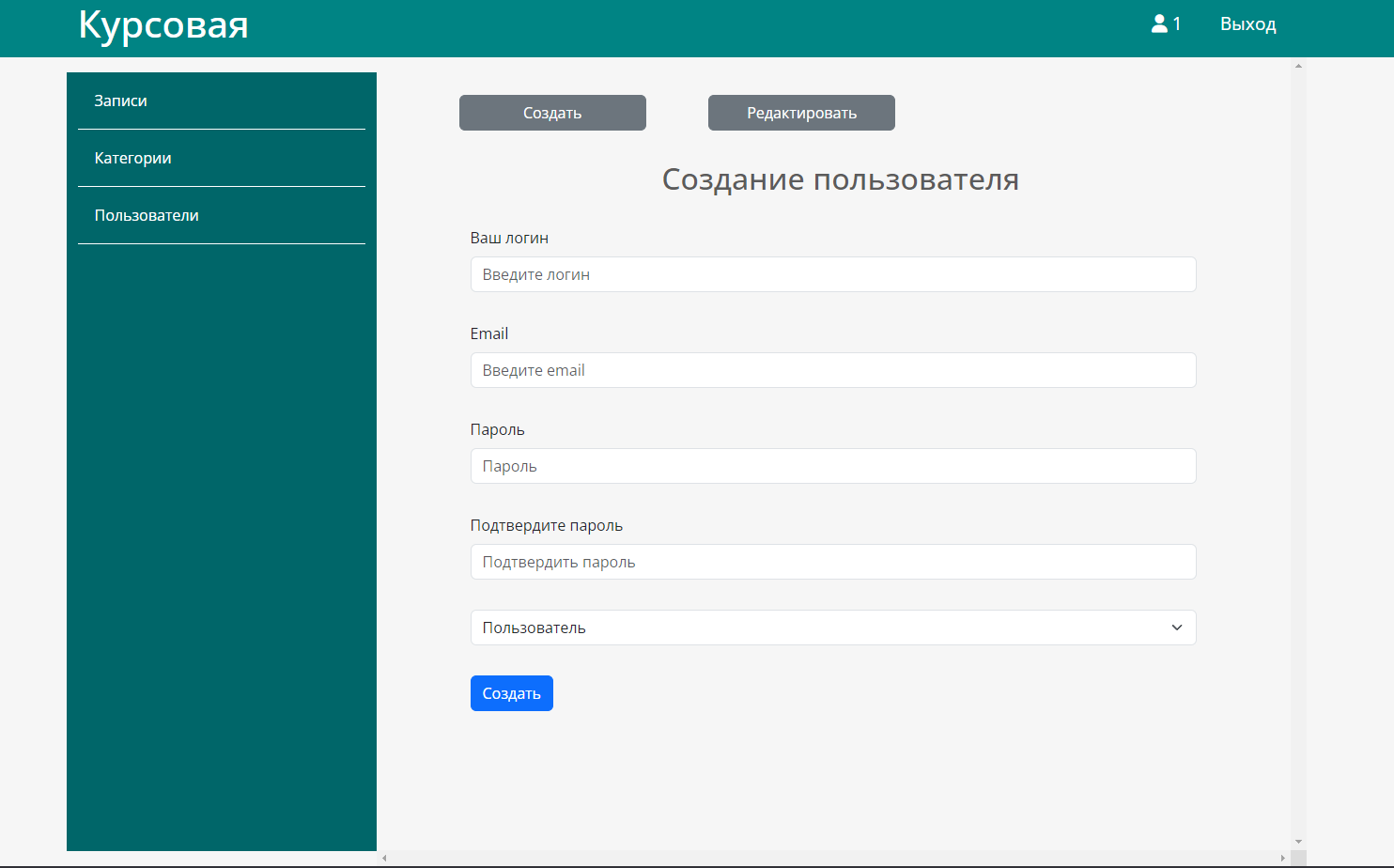
Этот код в основном использует Bootstrap для создания структуры и стилей веб-страницы.

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.

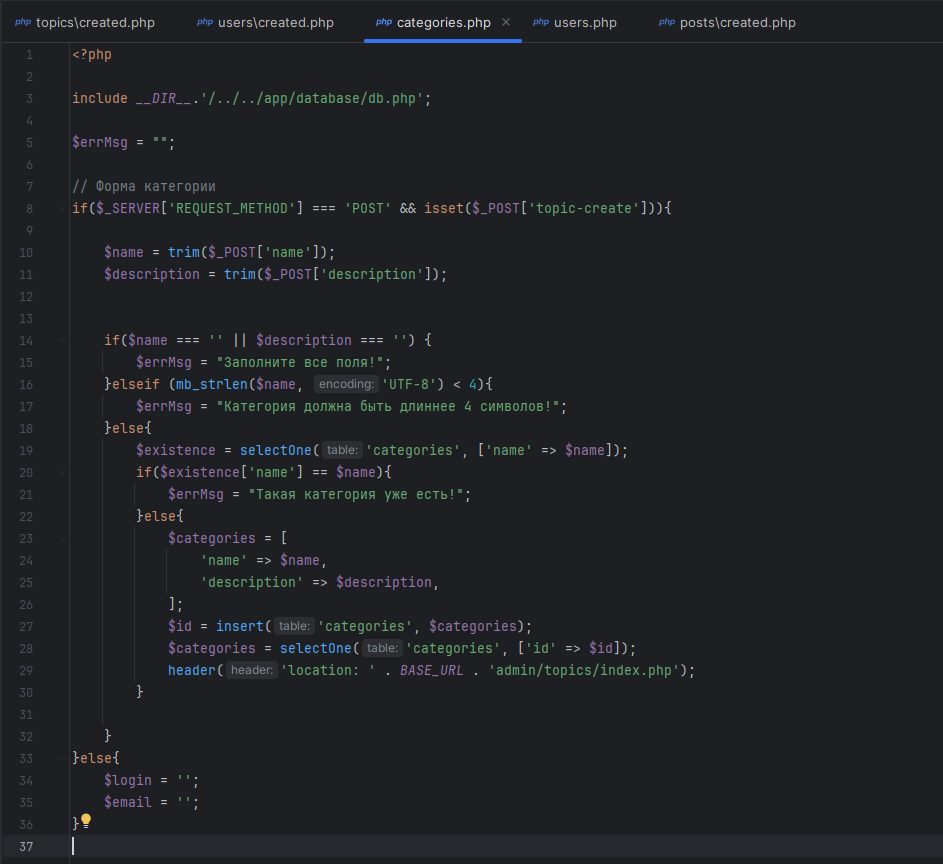
$existence = selectOne('categories', ['name' => $name]); if($existence['name'] == $name){ $errMsg = "Такая категория уже есть!";

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.

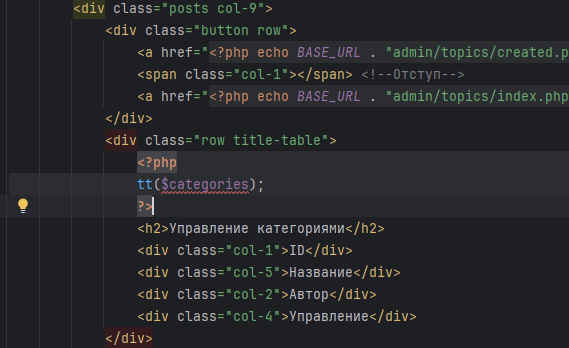


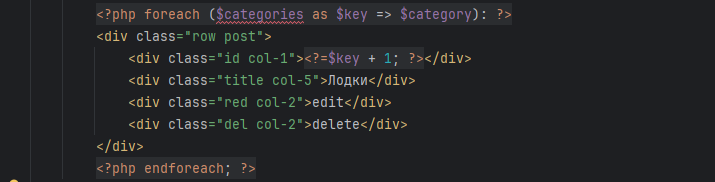
Сам массив, который выводится



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

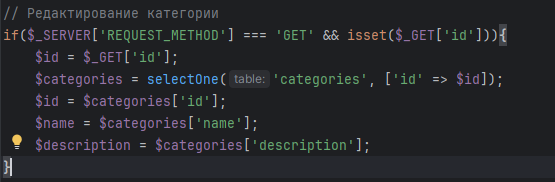
<?php tt($categories);?>





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



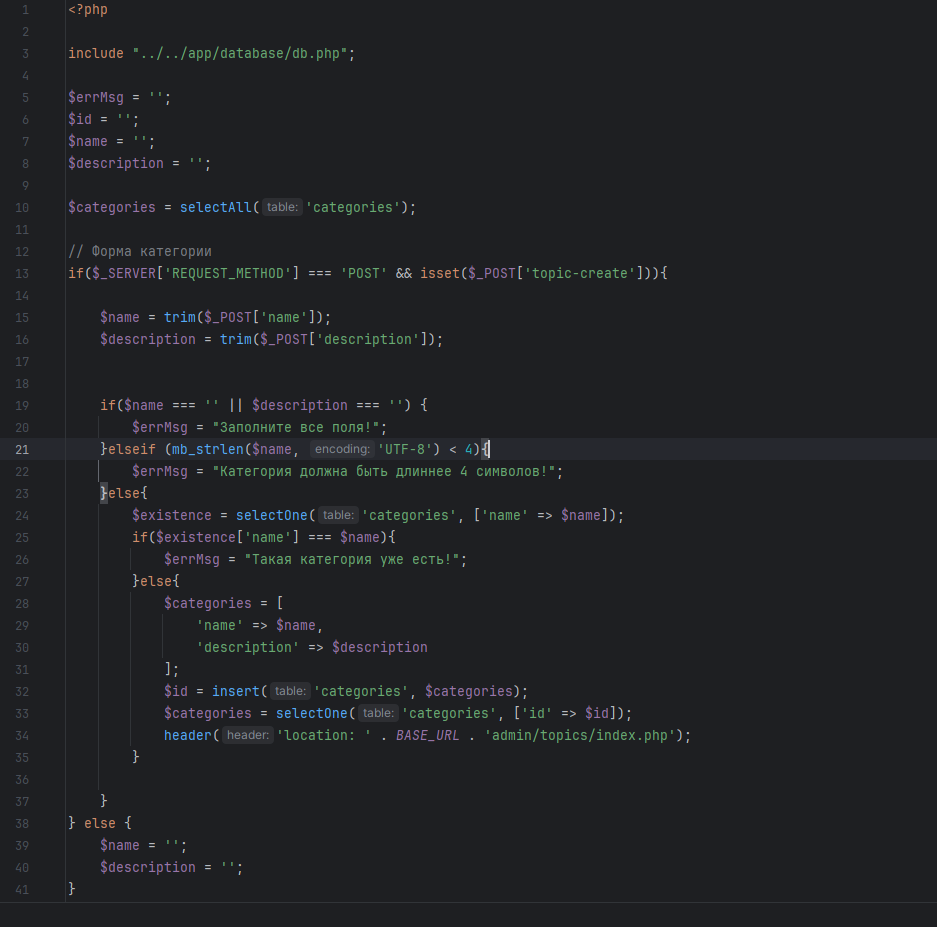
Проверяет, был ли отправлен HTTP GET-запрос и содержит ли запрос параметр id. Если условие выполняется, то:

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 ...

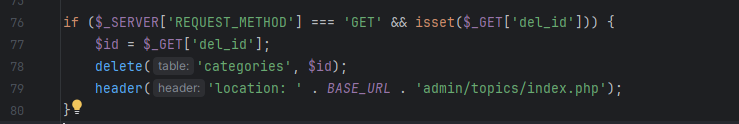
}

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



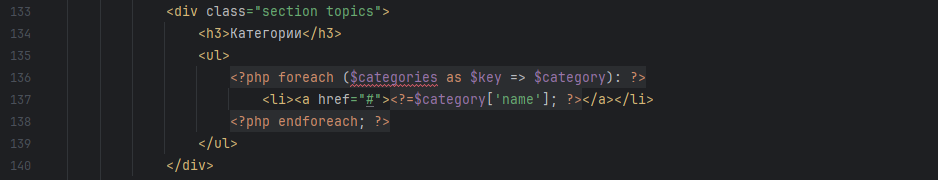
Этот код в index.php использует цикл foreach, чтобы для каждой категории из массива $categories создать блок HTML с информацией о категории. В каждом блоке выводится порядковый номер, название категории, а также ссылки для редактирования и удаления категории, которые перенаправляют пользователя на страницу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

Удаление категории



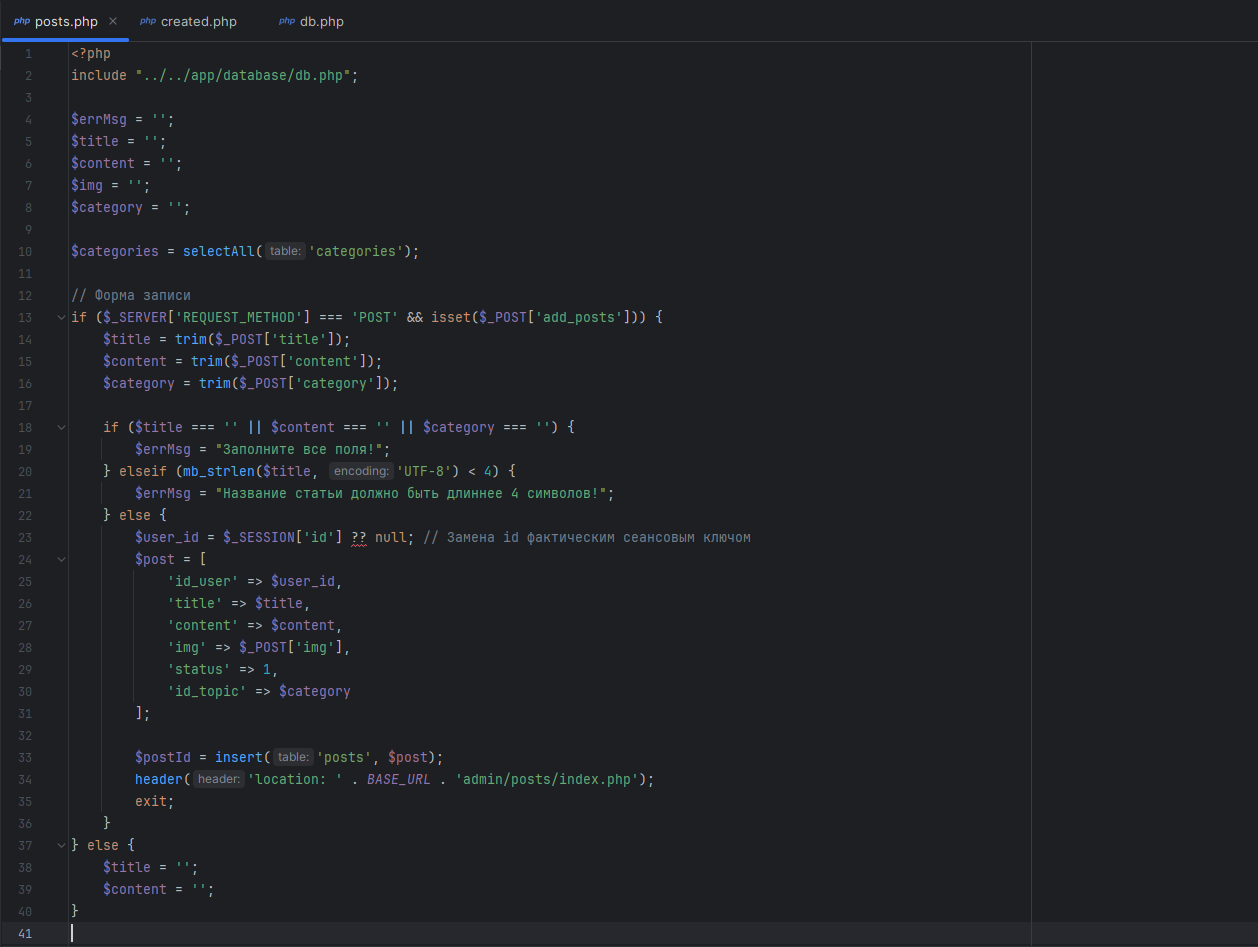
Если условие выполняется, то получает значение del\_id из параметра запроса, удаляет запись с соответствующим идентификатором в таблице categories при помощи функции delete, а затем перенаправляет пользователя обратно на страницу admin/topics/index.php.

Вывод категории на главную страницу



Таким образом, в результате выполнения этого кода на странице будет выведен список категорий в виде маркированного списка (<ul>), где каждая категория представлена ссылкой (<a>). Текст ссылки будет содержать значение из ключа 'name' каждого элемента массива $categories.

Логика создания записи



Объявляются несколько переменных, которые будут использоваться позже в коде:

$errMsg = ''; // Переменная для сообщений об ошибках

$title = ''; // Переменная для хранения заголовка статьи

$content = ''; // Переменная для хранения содержания статьи

$img = ''; // Переменная для хранения пути к изображению (возможно)

$category = ''; // Переменная для хранения категории статьи

Загружаются категории из базы данных:

$categories = selectAll('categories');

Переходим к обработке формы:

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['add\_posts'])) {

// Получение данных из формы

$title = trim($\_POST['title']);

$content = trim($\_POST['content']);

$category = trim($\_POST['category']);

// Проверка наличия данных в полях формы

if ($title === '' || $content === '' || $category === '') {

$errMsg = "Заполните все поля!";

} elseif (mb\_strlen($title, 'UTF-8') < 4) {

$errMsg = "Название статьи должно быть длиннее 4 символов!";

} else {

// Если данные прошли проверку, формируется массив данных для вставки в базу

$user\_id = $\_SESSION['id'] ?? null;

$post = [

'id\_user' => $user\_id,

'title' => $title,

'content' => $content,

'img' => $\_POST['img'], // Возможно, здесь должен быть код для обработки изображения

'status' => 1,

'id\_topic' => $category

];

// Вставка данных в базу

$postId = insert('posts', $post);

// Перенаправление на страницу с постами в случае успешного добавления

header('location: ' . BASE\_URL . 'admin/posts/index.php');

exit;

}

} else {

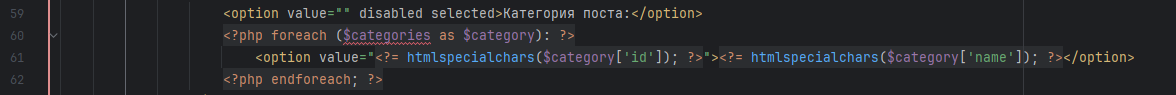
// Если форма не отправлена, поля данных обнуляются

$title = '';

$content = '';

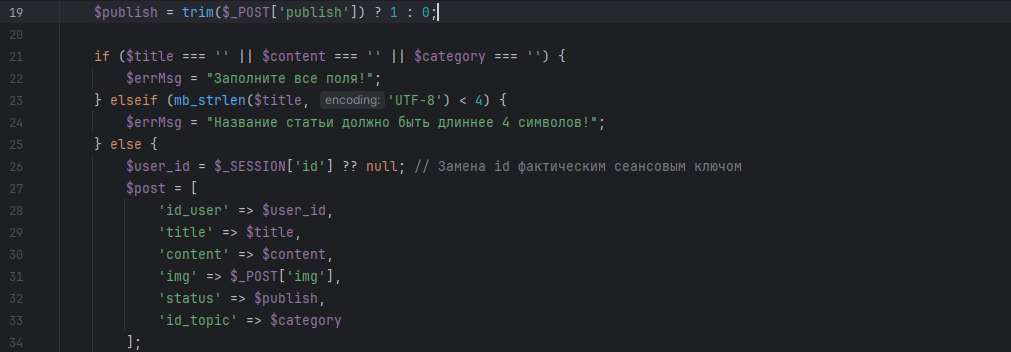
}

Краткое описание: Этот код отвечает за добавление новых статей в базу данных. Он проверяет, была ли отправлена форма, затем извлекает данные из формы, проверяет их наличие и корректность, и, если всё в порядке, добавляет статью в базу данных. В случае ошибок ввода, выводится сообщение об ошибке.



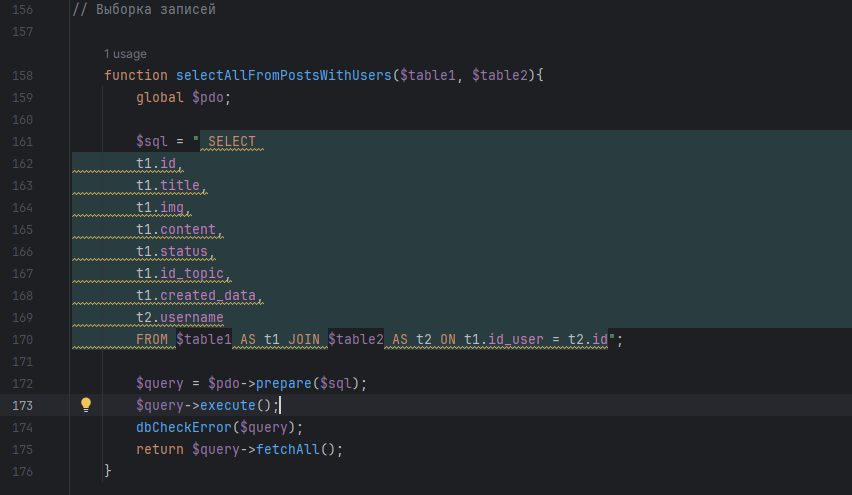
Этот код создает выпадающий список категорий для постов. Первая опция служит заглушкой и отключена для выбора. Затем используется цикл foreach, чтобы для каждой категории вставить опцию в список с уникальным id в значении и именем в тексте.

Вывод публикаций



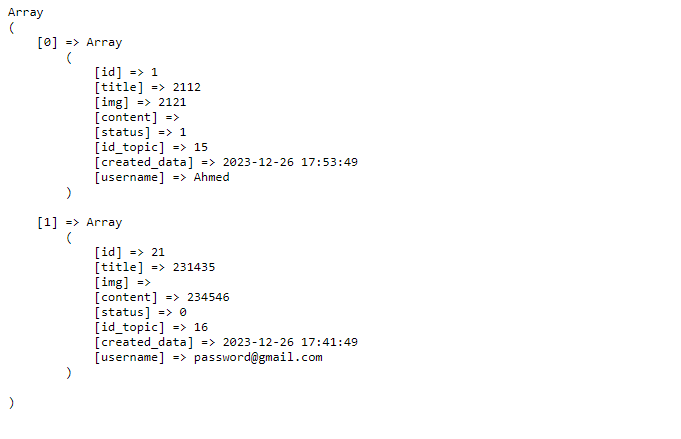
$publish будет равно 1, если чекбокс был выбран, и 0, если не был выбран. После этого значение $publish используется для установки статуса статьи: Таким образом, статус статьи будет либо активным (1), если чекбокс "опубликовать" был выбран, либо неактивным (0), если чекбокс не был выбран.

Выборка записей

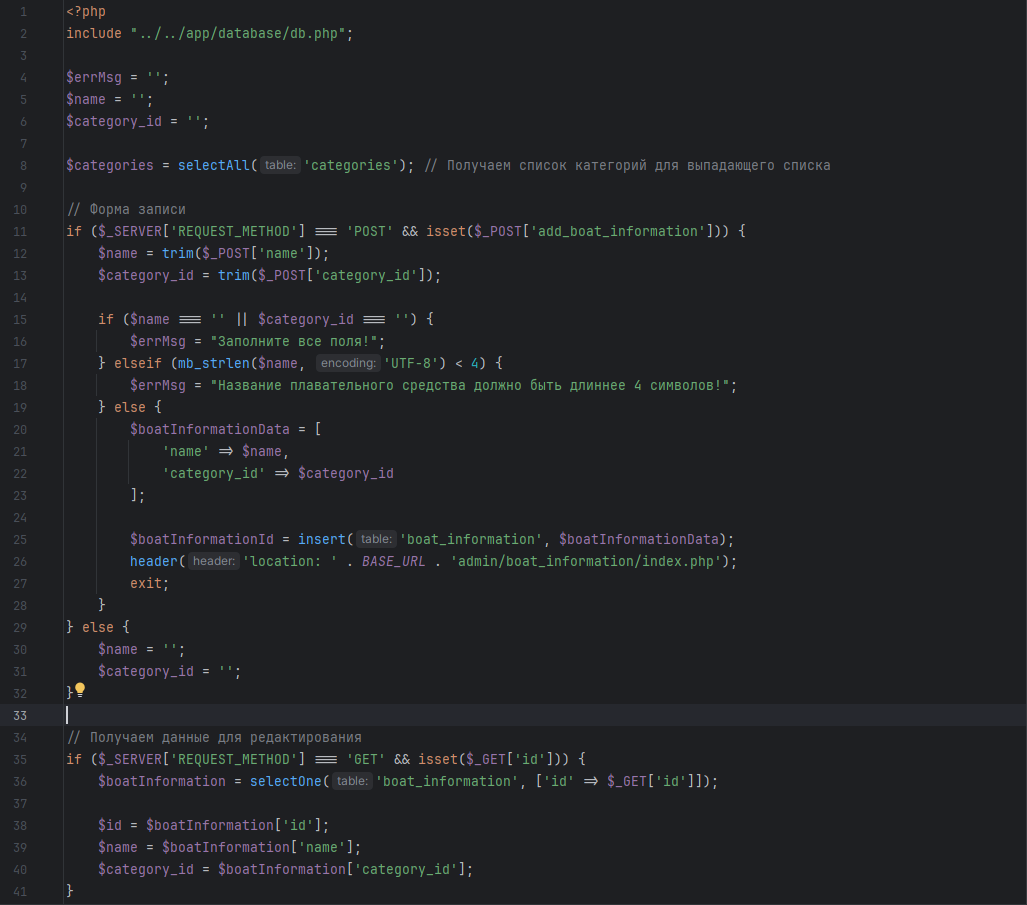


Эта функция выполняет SQL-запрос для выборки данных из двух таблиц базы данных и объединяет результаты с использованием оператора JOIN.

JOIN - это оператор в языке SQL (Structured Query Language), который используется для объединения строк из двух или более таблиц в результате запроса. Оператор JOIN выполняет это объединение на основе указанных условий совпадения между столбцами таблиц.



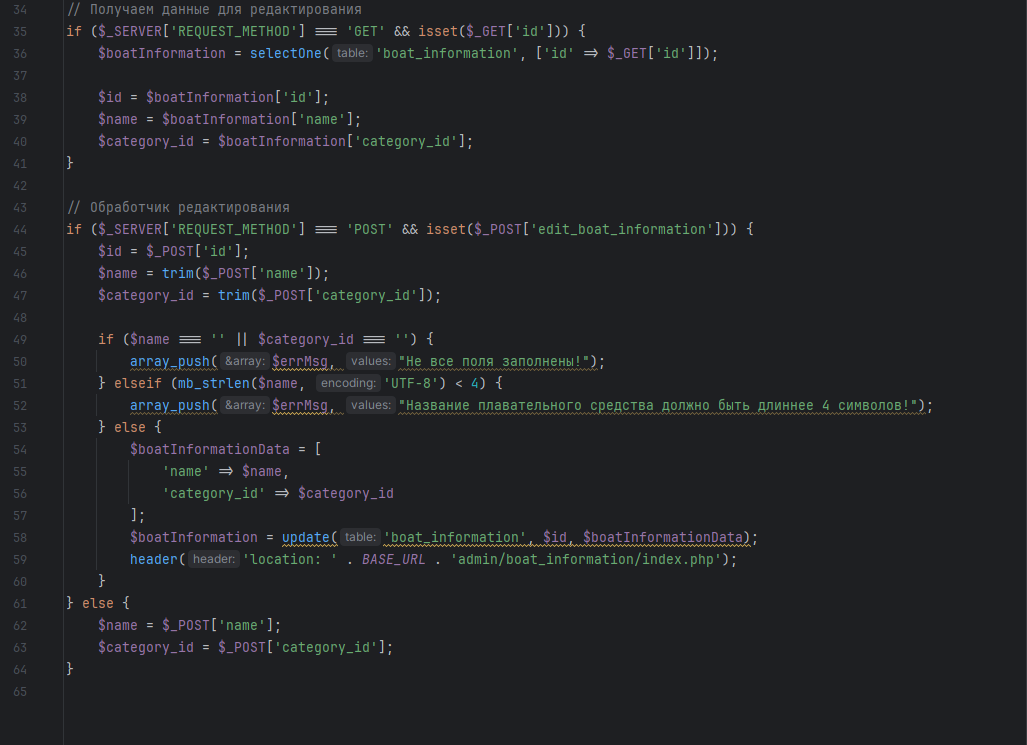
Управление данными о плавательных средствах



Представляет собой часть функционала для управления информацией о плавательных средствах.

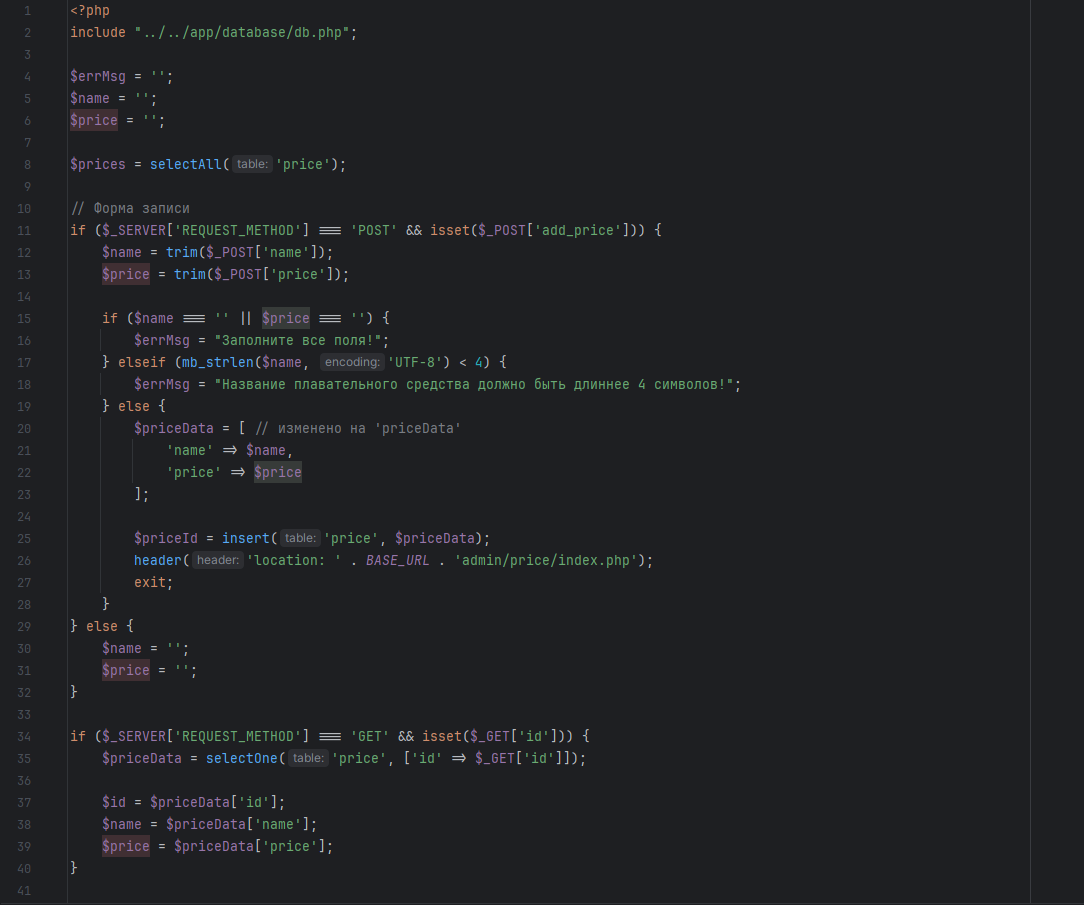
if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['add\_boat\_information'])) { // ... } В этом блоке кода проверяется, была ли отправлена форма с именем add\_boat\_information. Если форма отправлена, извлекаются данные из формы (название и идентификатор категории), проводится валидация данных, и если данные валидны, они добавляются в базу данных.

if ($\_SERVER['REQUEST\_METHOD'] === 'GET' && isset($\_GET['id'])) { // ... } Если запрос приходит методом GET и содержит параметр id, код извлекает информацию о плавательном средстве с указанным идентификатором для последующего редактирования.



Обработчик редактирования: if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['edit\_boat\_information'])) { // ... } В этом блоке кода проверяется, была ли отправлена форма с именем edit\_boat\_information. Если форма отправлена, извлекаются данные из формы (идентификатор, название и идентификатор категории), проводится валидация данных, и если данные валидны, информация о плавательном средстве обновляется в базе данных.

CRUD-функционал для администрирования цен веб-приложения.

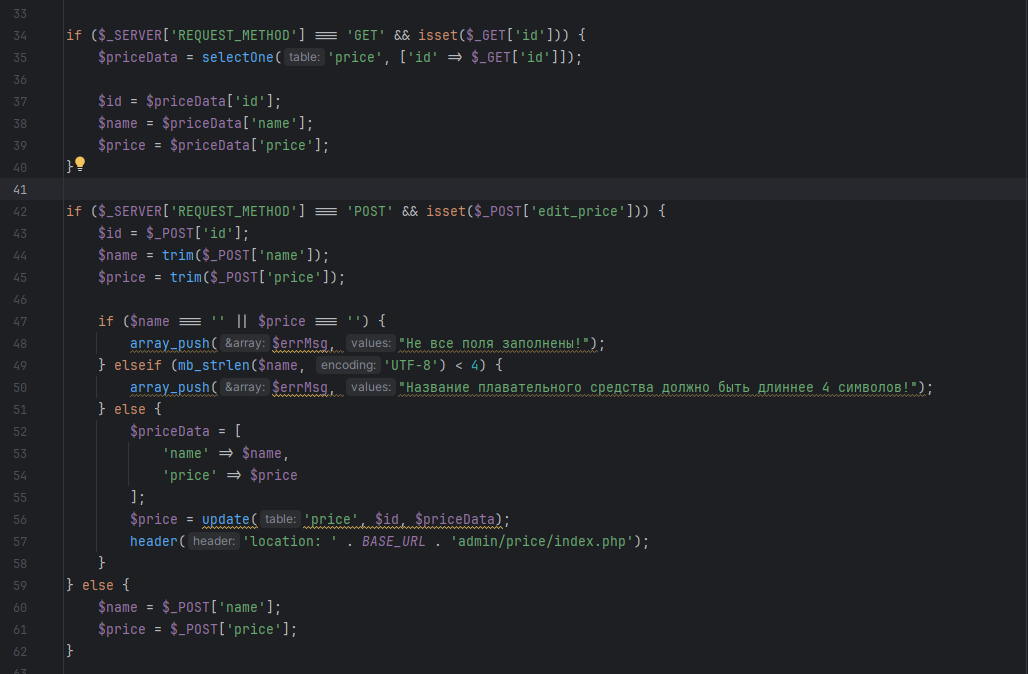


Этот код реализует веб-страницу для управления ценами на какие-то товары или услуги.

$errMsg = ''; $name = ''; $price = ''; Здесь объявляются переменные для хранения сообщений об ошибках ($errMsg), названия товара/услуги ($name) и цены ($price).

$prices = selectAll('price'); Извлекаются все цены из таблицы 'price' и сохраняются в массиве $prices.

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['add\_price'])) { // ... (получение данных из формы и их валидация) $priceData = [ 'name' => $name, 'price' => $price ]; $priceId = insert('price', $priceData); header('location: ' . BASE\_URL . 'admin/price/index.php'); exit; } Если форма отправлена методом POST и установлен соответствующий параметр (add\_price), данные из формы сохраняются в массив $priceData, который передается функции insert для добавления новой записи в таблицу 'price'.



if ($\_SERVER['REQUEST\_METHOD'] === 'GET' && isset($\_GET['id'])) { $priceData = selectOne('price', ['id' => $\_GET['id']]); $id = $priceData['id']; $name = $priceData['name']; $price = $priceData['price']; } Если запрос выполнен методом GET и установлен параметр 'id', то данные конкретной записи извлекаются из таблицы 'price' и используются для предзаполнения формы редактирования.

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['edit\_price'])) { // ... (получение данных из формы и их валидация) $priceData = [ 'name' => $name, 'price' => $price ]; $price = update('price', $id, $priceData); header('location: ' . BASE\_URL . 'admin/price/index.php'); } Если форма отправлена методом POST и установлен соответствующий параметр (edit\_price), данные из формы сохраняются в массив $priceData, который передается функции update для изменения существующей записи в таблице 'price'.

} else { $name = $\_POST['name']; $price = $\_POST['price']; } Если форма не отправлена или не редактировалась, то переменные $name и $price получают значения из формы для отображения в случае ошибок или для предзаполнения при повторном вводе.

Материал:

<https://www.tutorialrepublic.com/php-tutorial/php-mysql-crud-application.php>

https://www.simplilearn.com/tutorials/php-tutorial/pdo-in-php

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

<https://cyberleninka.ru/article/n/web-development-s-ispolzovaniem-php-i-freymvorka-laravel>

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

<https://metanit.com/php/>

<https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics>

Мой GitHub:

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