Laravel login and logout session

Setting up the Login and Logout Session in Laravel

Step 1: Create a Laravel Project

First, you need to create a new Laravel project using the following command in the terminal:

```

composer create-project --prefer-dist laravel/laravel project-name

```

Step 2: Setup Database

Next, setup the database credentials in the `.env` file located in the root directory of your project.

```

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=your_database_name

DB_USERNAME=your_database_username

DB_PASSWORD=your_database_password

```

Step 3: Create Authentication Scaffold

Laravel provides an in-built feature to scaffold the authentication views and controllers using the following command:

```

php artisan make:auth

```

This command will generate the required views, controllers, and routes necessary for user authentication.

Step 4: Migrate Database

Now, you need to run the migration command to create the `users` table in the database.

```

php artisan migrate

```

Step 5: Add Authentication Routes

By default, Laravel provides the following authentication routes:

```

Route::get('/login', 'Auth\LoginController@showLoginForm')->name('login');

Route::post('/login', 'Auth\LoginController@login');

Route::post('/logout', 'Auth\LoginController@logout')->name('logout');

```

Step 6: Add Login and Logout Links

You can add the login and logout links in your views using the following code:

```

@if (Route::has('login'))

<div class="top-right links">

@auth

<a href="{{ url('/home') }}">Homea>

<form action="{{ route('logout') }}" method="POST">

@csrf

<button type="submit">Logoutbutton>

form>

@else

<a href="{{ route('login') }}">Logina>

@if (Route::has('register'))

<a href="{{ route('register') }}">Registera>

@endif

@endauth

div>

@endif

```

Step 7: Protect Routes

To protect your routes, you can use the `auth` middleware in your routes. For example:

```

Route::get('/profile', 'UserController@profile')->middleware('auth');

```

This will ensure that only authenticated users can access the `profile` route.

Step 8: Test the Login and Logout Functionality

Finally, you can test the login and logout functionality of your Laravel application. You can visit the login page at `http://your-app-url/login` and enter the email and password of your registered user to log in. Once you are logged in, you can access the protected routes. To log out, you can click the "Logout" button in the navigation bar.

Laravel CRUD Operation

Here are the steps to execute CRUD operations in Laravel:

Setting Up the Database

Before we can create CRUD operations in Laravel, we need to set up the database. This involves creating a schema and migrations.

1. Create a new database schema in your preferred database management tool.

2. Create a new migration by running the following command in your terminal:

```

php artisan make:migration create_[table_name]_table

```

Replace `[table_name]` with the name of the table you want to create.

3. In the newly created migration file, define the table schema using the `Schema` facade. For example, to create a `users` table with `name` and `email` fields, you could use the following code:

```php

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->string('email')->unique();

$table->timestamps();
```

});

}

```

4. Run the migration to create the table in the database by running the following command:

```

php artisan migrate

```

## Creating the Model

Next, we need to create a model that will interact with the database table we just created.

1. Create a new model by running the following command in your terminal:

```

php artisan make:model [ModelName]

```

Replace `[ModelName]` with the name of your model.

2. In the newly created model file, define the model's properties, including the table name, fillable fields, and any relationships with other models. For example, to create a `User` model, you could use the following code:

```php

class User extends Model

{

protected $fillable = ['name', 'email'];

}

```

Creating the Controller

Now that we have a model, we need to create a controller to handle the CRUD operations.

1. Create a new controller by running the following command in your terminal:

```

php artisan make:controller [ControllerName]

```

Replace `[ControllerName]` with the name of your controller.

2. In the newly created controller file, define the controller's methods for each CRUD operation. For example, to create a `UserController` with methods for creating, reading, updating, and deleting users, you could use the following code:

```php

class UserController extends Controller

{

public function index()

{

$users = User::all();

return view('users.index', compact('users'));

}

```
public function create()
{
return view('users.create');
}

public function store(Request $request)
{
$user = User::create($request->all());
return redirect()->route('users.index');
}

public function edit(User $user)
{
return view('users.edit', compact('user'));
}

public function update(Request $request, User $user)
{
$user->update($request->all());
return redirect()->route('users.index');
}
```

```
public function destroy(User $user)

{

$user->delete();

return redirect()->route('users.index');

}

}
```

Creating the Views

Finally, we need to create the views for our CRUD operations.

1. Create a directory in your `resources/views` directory called `users`.

2. Create a view file for each CRUD operation. For example, to create an `index` view that displays a list of users, you could use the following code:

```html
<table>

<thead>

<tr>

<th>Nameth>

<th>Emailth>

<th></th>

tr>
```

thead>

<tbody>

@foreach ($users as $user)

<tr>

<td>{{ $user->name }}</td>

<td>{{ $user->email }}</td>

<td>

<a href="{{ route('users.edit', $user) }}">Edita>

<form action="{{ route('users.destroy', $user) }}" method="POST">

@csrf

@method('DELETE')

<button>Deletebutton>

form>

td>

tr>

@endforeach

tbody>

table>

```

And to create a `create` view that displays a form for creating a new user, you could use the following code:
```

```html
<form action="{{ route('users.store') }}" method="POST">
@csrf
<div>
<label for="name">Name:</label>
<input type="text" name="name" id="name">
div>
<div>
<label for="email">Email:</label>
<input type="email" name="email" id="email">
div>
<button>Create Userbutton>
form>
```

And that's it! With these steps, you should now be able to execute CRUD operations in Laravel.

Connecting PHP to MySQL Database Tutorial

PHP MySQL Connectivity Tutorial

When building dynamic web applications, it is common to use PHP and MySQL together. PHP is a popular server-side scripting language, while MySQL is a widely-used relational database management system. In this tutorial, we will go over the steps to connect PHP to a MySQL database.

Step 1: Install the Required Software

Before we can begin, we need to make sure that PHP and MySQL are installed on our system. You can download and install PHP from the official website (https://www.php.net/downloads.php), while MySQL can be downloaded from the MySQL website (**https://dev.mysql.com/downloads/**).

Step 2: Create a Database in MySQL

Next, we will create a database that PHP can connect to. You can use the MySQL command-line tool to create a database. Here's an example:

```

mysql -u root -p

```

This command will prompt you to enter the MySQL root user password. Once you have entered the password, you will see the MySQL command prompt. Here's an example of how to create a database:

```

CREATE DATABASE example_db;

```

Step 3: Connect PHP to MySQL

To connect PHP to MySQL, we will use the `mysqli` extension. Here's an example of how to connect to the database we created in the previous step:

```

php

$servername = "localhost";

$username = "root"; // replace with your MySQL username

$password = "your_password"; // replace with your MySQL password

$dbname = "example_db"; // replace with your MySQL database name

// Create connection

$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection

if (!$conn) {

die("Connection failed: " . mysqli_connect_error());

}

echo "Connected successfully";

?>

```

In the above example, we first set the servername, username, password, and database name variables. We then use the `mysqli_connect` function to connect to the database. If the connection fails, we use the `mysqli_connect_error` function to display an error message.

Step 4: Perform Database Operations

Once we have connected to the database, we can perform various operations such as inserting data, updating data, and selecting data. Here's an example of how to insert data into a table:

```

$sql = "INSERT INTO example_table (name, email, phone) VALUES ('John Doe', 'johndoe@example.com', '123-456-7890')";

if (mysqli_query($conn, $sql)) {

echo "New record created successfully";

} else {

echo "Error: " . $sql . "<br>" . mysqli_error($conn);

}

```

In the above example, we use the `mysqli_query` function to execute an SQL statement. If the statement is executed successfully, we display a success message. Otherwise, we display an error message.

Closing the Connection

After you're done using the database, you should close the connection to free up resources. You can do this by calling the `close()` method on the `$conn` object:

```php

$conn->close();

```

Conclusion

In this tutorial, we went over the steps to connect PHP to a MySQL database. We first installed the required software, created a database in MySQL, connected

PHP to the database using the `mysqli` extension, and performed database operations such as inserting data. With this knowledge, you should be able to build dynamic web applications that use PHP and MySQL together.

PHP MySQL: Basic Database Operations with Examples

PHP MySQL Explained with Examples

PHP is a server-side scripting language used for web development, and MySQL is a popular open-source relational database management system used for storing and retrieving data. In this guide, we will explain how to use PHP and MySQL together with examples.

Connecting to MySQL Database

To connect to a MySQL database using PHP, you need to use the `mysqli_connect()` function. Here is an example:

```php

// MySQL database credentials

$host = "localhost";

$username = "myusername";

$password = "mypassword";

$database = "mydatabase";

// Create connection

$conn = mysqli_connect($host, $username, $password, $database);

// Check connection

if (!$conn) {
```

```
die("Connection failed: " . mysqli_connect_error());

}

echo "Connected successfully";
```

Inserting Data into MySQL Database

To insert data into a MySQL database using PHP, you need to use the `mysqli_query()` function. Here is an example:

```php
// Insert data into table

$sql = "INSERT INTO mytable (name, email, phone) VALUES ('John Doe', 'john@example.com', '1234567890')";

if (mysqli_query($conn, $sql)) {

echo "Data inserted successfully";

} else {

echo "Error: " . mysqli_error($conn);

}
```

Retrieving Data from MySQL Database

To retrieve data from a MySQL database using PHP, you need to use the `mysqli_query()` function to execute a SELECT query. Here is an example:

```php
// Retrieve data from table

$sql = "SELECT * FROM mytable";

$result = mysqli_query($conn, $sql);
```

```
if (mysqli_num_rows($result) > 0) {

// Output data of each row

while($row = mysqli_fetch_assoc($result)) {

echo "Name: " . $row["name"]. " - Email: " . $row["email"]. " - Phone: " . $row["phone"]. "<br>";

}

} else {

echo "0 results";

}
```

Updating Data in MySQL Database

To update data in a MySQL database using PHP, you need to use the `mysqli_query()` function to execute an UPDATE query. Here is an example:

```php
// Update data in table

$sql = "UPDATE mytable SET phone='0987654321' WHERE name='John Doe'";

if (mysqli_query($conn, $sql)) {

echo "Data updated successfully";

} else {

echo "Error updating data: " . mysqli_error($conn);

}
```

Deleting Data from MySQL Database

To delete data from a MySQL database using PHP, you need to use the `mysqli_query()` function to execute a DELETE query. Here is an example:

```php

// Delete data from table

$sql = "DELETE FROM mytable WHERE name='John Doe'";

if (mysqli_query($conn, $sql)) {

echo "Data deleted successfully";

} else {

echo "Error deleting data: " . mysqli_error($conn);

}

```

That's it! You now have a basic understanding of how to use PHP and MySQL together to create, read, update, and delete data from a database.

Connecting PHP with MongoDB for CRUD Operations

PHP and MongoDB Connectivity

MongoDB is a NoSQL database that is used to store and retrieve data. PHP is a popular programming language that is used to create dynamic web applications. In this tutorial, we will go over how to connect PHP with MongoDB to perform CRUD (Create, Read, Update, Delete) operations.

Prerequisites

Before we start, you will need to have the following software installed on your computer:

- PHP

- MongoDB

- MongoDB PHP driver

Connecting to MongoDB

To connect to MongoDB from PHP, we will need to use the MongoDB PHP driver. You can download the driver from the MongoDB website or install it using Composer.

```php

php

// connect to mongodb

$mongoClient = new MongoDB\Client("mongodb://localhost:27017");

```

In the above code, we are creating a new instance of the `MongoDB\Client` class and passing in the connection string for our MongoDB server. By default, MongoDB runs on port 27017.

Selecting a Database

Once we have connected to MongoDB, we need to select a database to work with.

```php

php

// select database

$database = $mongoClient->myDatabase;

```

In the above code, we are selecting a database called `myDatabase`. If the database does not exist, MongoDB will create it for us.

Selecting a Collection

A collection is like a table in a relational database. We can select a collection to work with using the `selectCollection()` method.

```php

php

// select collection

$collection = $database->myCollection;

```

In the above code, we are selecting a collection called `myCollection`. If the collection does not exist, MongoDB will create it for us.

Inserting Documents

To insert a new document into a collection, we can use the `insertOne()` method.

```php

php

// insert document

$insertResult = $collection->insertOne([

'name' => 'John Doe',

'email' => 'john.doe@example.com',

'age' => 25

]);

// print inserted document id

echo $insertResult->getInsertedId();

```

In the above code, we are inserting a new document into the `myCollection` collection. The `insertOne()` method takes an array of key-value pairs, where the keys are the names of the fields and the values are the values to be stored in those fields. The `getInsertedId()` method returns the _id of the inserted document.

Reading Documents

To read documents from a collection, we can use the `find()` method.

```php

php

// find documents

$documents = $collection->find();
```

```
// print documents

foreach ($documents as $document) {

echo $document['name'] . ' (' . $document['email'] . ')>';

}
```

In the above code, we are retrieving all documents from the `myCollection` collection using the `find()` method. The `find()` method returns a cursor object that we can iterate over to retrieve each document. We can access the fields of each document using array notation.

Updating Documents

To update a document in a collection, we can use the `updateOne()` method.

```php
php

// update document

$updateResult = $collection->updateOne(

['name' => 'John Doe'],

['$set' => ['age' => 30]]

);

// print number of updated documents

echo $updateResult->getModifiedCount();
```

In the above code, we are updating the `age` field of the document with the name `John Doe` to `30`. The `updateOne()` method takes two arguments: a filter to

select the document to update, and an update document that specifies the changes to be made.

Deleting Documents

To delete a document from a collection, we can use the `deleteOne()` method.

```php

php

// delete document

$deleteResult = $collection->deleteOne(['name' => 'John Doe']);

// print number of deleted documents

echo $deleteResult->getDeletedCount();

```

In the above code, we are deleting the document with the name `John Doe` from the `myCollection` collection. The `deleteOne()` method takes a filter to select the document to delete.

Conclusion

In this tutorial, we have gone over how to connect PHP with MongoDB to perform CRUD operations. We covered connecting to MongoDB, selecting a database and collection, inserting documents, reading documents, updating documents, and deleting documents. With this knowledge, you should be able to start building web applications that use MongoDB as a backend database.

Using PHP with MongoDB: Basic Operations

Here is an example of using PHP with MongoDB:

Prerequisites

Before we start, make sure you have the following installed:

- PHP (version 5.4 or higher)

- MongoDB PHP driver

You can install the MongoDB PHP driver via PECL:

```

pecl install mongodb

```

Connecting to MongoDB

To connect to MongoDB, you can use the `MongoDB\Driver\Manager` class. Here is an example of connecting to a MongoDB database on the local machine:

```php

$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");

```

Inserting Data

To insert a document into a collection, use the `MongoDB\Driver\BulkWrite` class. Here is an example of inserting a document into a collection:

```php
```

```
$bulk = new MongoDB\Driver\BulkWrite;

$doc = ['_id' => new MongoDB\BSON\ObjectID, 'name' => 'John Doe', 'age' => 30];

$bulk->insert($doc);

$result = $manager->executeBulkWrite('test.users', $bulk);
```

Querying Data

To query data from a collection, use the `MongoDB\Driver\Query` class. Here is an example of querying all documents from a collection:

```php
$query = new MongoDB\Driver\Query([]);

$cursor = $manager->executeQuery('test.users', $query);

foreach ($cursor as $document) {

var_dump($document);

}
```

Updating Data

To update a document in a collection, use the `MongoDB\Driver\BulkWrite` class. Here is an example of updating a document in a collection:

```php
$bulk = new MongoDB\Driver\BulkWrite;

$filter = ['name' => 'John Doe'];
```

```
$update = ['$set' => ['age' => 31]];

$bulk->update($filter, $update);

$result = $manager->executeBulkWrite('test.users', $bulk);
```

Deleting Data

To delete a document from a collection, use the `MongoDB\Driver\BulkWrite` class. Here is an example of deleting a document from a collection:

```php
$bulk = new MongoDB\Driver\BulkWrite;

$filter = ['name' => 'John Doe'];

$bulk->delete($filter);

$result = $manager->executeBulkWrite('test.users', $bulk);
```

And that's it! This is a basic example of using PHP with MongoDB. There are many other features and capabilities of MongoDB, so be sure to check out the official documentation for more information.