Disciplina: Desenvolvimento Web 2 (DW2)

Professor: Alex Paulo Lopes Batista

Data de Entrega: 12/12/2024 às 23h59min

Avaliação 2:

Instruções Iniciais:

1. Leia **atentamente** todas as **instruções iniciais** e as **tarefas da Avaliação2(A2)** para construir esta aplicação web com e em caso de dúvidas contate o professor.

2. **A Avaliação2 (A2) pode ser feita em dupla ou individualmente.**

3. **Criar uma pasta** chamada **Avaliação2.**

4. Dentro da **pasta Avaliação2 crie** uma **nova pasta** com o seu **nome ou nome da dupla, exemplo** : “CarlosOliviera“ ou “CarlosOliveira-AndrePereira”.

5. Por fim, vá até a pasta “**Avaliação2”** e faça a **compactação** no **formato de arquivo .ZIP (exemplo: DW2-A2-AlexBatista.zip).**

6. **Deixar de seguir as regras das instruções iniciais de 1 a 5 acima, vai acarretar decréscimo na nota do Avaliação2 (A2).**

OBS: Utilizem os arquivos auxiliares que estarão disponíveis nos materiais distribuídos pelo professor no MSTeams.

Avaliação2 (A2) – Aplicação Web PHP Login e Registro de Usuários:

Introdução:

Leia cada capítulo do conteúdo desse tutorial passo a passo e após análise faça as devidas alterações no projeto da aplicação web PHP login e registro de usuários. Utilize o projeto “login” na sua versão2 que se encontra disponível no MSTeams para criar uma terceira versão já com as alterações contidas nos capítulos dessa atividade.

1. Configuração de Acesso ao Banco de Dados

Nesse capítulo será mostrado o acesso de configuração do banco de dados, através do arquivo “config.php” conforme a listagem 1 abaixo:

Listagem 1 – Arquivo “config.php”:

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "login";

// Criar conexão
$conn = new mysqli($servername, $username, $password, $dbname);

// Verificar conexão
if ($conn->connect_error) {
 die("Falha na conexão com o banco de dados: " . $conn->connect_error);
}
echo "<strong>Conectado com sucesso!!!</strong><br><br>";
 ?>
```

Ao considerar a listagem 1 acima será feita uma análise do código para entender como proceder melhorar esse código-fonte.

Análise do Código

O que o Código Faz

O código config.php estabelece uma conexão com um banco de dados MySQL usando a extensão mysqli. Ele define as credenciais necessárias (servidor, usuário, senha e nome do banco de dados) e verifica se a conexão foi bem-sucedida.

Pontos Positivos

- **Estrutura Básica**: O código está bem estruturado para uma configuração inicial.
- **Verificação de Erros**: A verificação da conexão é uma boa prática, pois permite identificar problemas imediatamente.
-

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para melhorar e incrementar seu código:

1.1. Uso de Constantes para Configuração

Em vez de usar variáveis simples, você pode definir constantes. Isso ajuda a evitar alterações acidentais nas credenciais.

```
<?php

define('DB_SERVER', 'localhost');

define('DB_USERNAME', 'root');

define('DB_PASSWORD', '');

define('DB_DATABASE', 'login');


// Criar conexão

$conn = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_DATABASE);

// Verificar conexão

if ($conn->connect_error) {
```

```
    die("Falha na conexão com o banco de dados: " . $conn->connect_error);
}
?>
```

1.2. Uso de PDO para Conexão

Considere usar PDO (PHP Data Objects) em vez de mysqli. O PDO oferece uma interface mais flexível e suporte a múltiplos bancos de dados.

```
<?php
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);

    // Definindo o modo de erro do PDO para exceções
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(PDOException $e) {
    die("Falha na conexão com o banco de dados: " . $e->getMessage());
}
?>
```

1.3. Configuração em Arquivo Externo

Para maior segurança, considere armazenar suas credenciais em um arquivo separado que não seja acessível publicamente. Isso ajuda a proteger informações sensíveis.

1.4. Tratamento de Erros

Ao invés de usar die(), você pode implementar um sistema de logging para registrar erros sem expor detalhes ao usuário final.

1.5. Comentários e Documentação

Adicione comentários explicativos para facilitar a manutenção do código no futuro.

No próximo capítulo será feita análise do painel de configuração do usuário, através do arquivo “dashboard.php”.

2. Painel de Configuração de Usuário.

Nesse capítulo será mostrado o acesso de configuração do painel de configuração do usuário, através do arquivo “dashboard.php” conforme a listagem 2 abaixo:

Listagem 2 – Arquivo “dashboard.php”:

```
<?php
   session_start();
   if (!isset($_SESSION['usuario_id'])) {
      header('location: login.php');
   }
   echo "Bem-vindo ao Dashboard!<br>";
   echo "Olá ".$_SESSION['usuario_nome']." você está logado !!!<br>";
   echo "Autorizado: ".$_SESSION['tipo_descricao'] ;
?>
<br>
<a href="logout.php">Logout</a>
```

Considere a listagem 2 acima será feita uma análise do código para entender como proceder melhorar esse código-fonte.

Análise do Código

O que o Código Faz

1. **Verificação de Sessão:** O código verifica se o usuário está autenticado, checando se a variável de sessão usuario_id está definida. Se não estiver, redireciona para a página de login.

2. **Exibição de Mensagens:** Se o usuário estiver autenticado, exibe uma mensagem de boas-vindas com o nome do usuário e seu tipo de autorização.
3. **Link para Logout:** Fornece um link para a página de logout.

Pontos Positivos

- **Segurança Básica:** A verificação da sessão é uma prática importante para proteger páginas que requerem autenticação.
- **Mensagem Personalizada:** A saudação personalizada melhora a experiência do usuário.

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para aprimorar seu código:

2.1. Sanitização da Saída

Para evitar vulnerabilidades como XSS (Cross-Site Scripting), é importante sanitizar as saídas que são exibidas na página. Você pode usar a função htmlspecialchars():

```
echo "Olá " . htmlspecialchars($_SESSION['usuario_nome']) . ", você está logado !!!<br>";
```

2.2. Melhorar a Estrutura do HTML

Considere estruturar melhor o HTML, utilizando tags apropriadas para melhorar a legibilidade e acessibilidade:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <title>Dashboard</title>
</head>
```

```
<body>
    <h1>Bem-vindo ao Dashboard!</h1>
    <p>Olá <?php echo htmlspecialchars($_SESSION['usuario_nome']); ?>, você está logado !!!</p>
    <p>Autorizado: <?php echo htmlspecialchars($_SESSION['tipo_descricao']); ?></p>
    <br>
    <a href="logout.php">Logout</a>
</body>
</html>
```

2.3. Implementação de Logout

Certifique-se de que sua página logout.php destrua a sessão corretamente e redirecione o usuário para a página de login:

```
<?php
session_start();
session_destroy(); // Destrói todas as informações da sessão
header('location: login.php'); // Redireciona para a página de login
exit();
?>
```

2.4. Adição de Estilo

Você pode adicionar CSS para melhorar a aparência do dashboard, tornando-o mais atraente visualmente.

2.5. Feedback ao Usuário

Considere adicionar mensagens de feedback ou alertas (por exemplo, usando JavaScript ou PHP) caso algo não funcione como esperado.

No próximo capítulo será feita análise da página inicial, através do arquivo “index.php”.

3. Página inicial da Aplicação Web Login.

Nesse capítulo será mostrada a página inicial da aplicação web login, através do arquivo “index.php” conforme a listagem 3 abaixo:

Listagem 3 – Arquivo “index.php”:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Página Inicial</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f4f4f4;
        }
        .container {
            text-align: center;
        }
        .button {
            display: inline-block;
            padding: 10px 20px;
            margin: 10px;
            font-size: 16px;
            color: white;
```

```
        background-color: #007BFF; /* Cor azul */
        text-decoration: none; /* Remove o sublinhado */
        border-radius: 5px; /* Bordas arredondadas */
        transition: background-color 0.3s; /* Efeito de transição */
    }
    .button:hover {
        background-color: #0056b3; /* Cor ao passar o mouse */
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Bem-vindo ao Sistema Login</h1>
    <a href="login.php" class="button">Login</a>
    <a href="registro.php" class="button">Registrar</a>
  </div>
</body>
</html>
```

A seguir tem-se a análise desse código-fonte do arquivo “index.php”.

O que o Código Faz

1. **Estrutura HTML**: O código cria uma página HTML básica com um título e duas opções de navegação: "Login" e "Registrar".
2. **Estilização CSS**: O estilo é aplicado diretamente na página, utilizando flexbox para centralizar os elementos e estilizar os botões.

Pontos Positivos

- **Design Responsivo**: O uso de viewport e flexbox ajuda a manter a página responsiva em diferentes tamanhos de tela.
- **Interatividade Visual**: A transição de cor nos botões ao passar o mouse melhora a experiência do usuário.

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para aprimorar seu código:

3.1. Separar CSS em um Arquivo Externo

Para facilitar a manutenção e reutilização, considere mover o CSS para um arquivo externo, como styles.css. Isso torna o HTML mais limpo e melhora o carregamento da página:

```
<link rel="stylesheet" href="styles.css">
```

3.2. Melhorar Acessibilidade

Adicione atributos aria ou use elementos HTML semânticos para melhorar a acessibilidade da sua página:

```
<h1 aria-label="Título da Página">Bem-vindo ao Sistema Login</h1>
```

3.3. Incluir Meta Tags Adicionais

Considere adicionar meta tags para SEO (Search Engine Optimization) e descrição da página:

```
<meta name="description" content="Sistema de login para gerenciar usuários.">
```

3.4. Adicionar um Rodapé

Um rodapé pode ser útil para incluir informações adicionais, como direitos autorais ou links úteis.

```
<footer>
  <p>&copy; 2024 Seu Nome ou Empresa. Todos os direitos reservados.</p>
</footer>
```

3.5. Feedback Visual no Registro

Se você tiver uma página de registro (registro.php), considere adicionar uma mensagem que informe ao usuário se o registro foi bem-sucedido ou não.

3.6. Validação de Formulário

Caso você implemente formulários na página de registro, não se esqueça de incluir validação tanto no lado do cliente (JavaScript) quanto no lado do servidor (PHP) para garantir a integridade dos dados.

3.7 Exemplo Atualizado

Aqui está uma versão atualizada do seu index.php com algumas das sugestões implementadas:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="Sistema de login para gerenciar usuários.">
    <title>Página Inicial</title>
    <link rel="stylesheet" href="styles.css"> <!-- CSS externo -->
</head>
<body>
    <div class="container">
        <h1>Bem-vindo ao Sistema Login</h1>
        <a href="login.php" class="button">Login</a>
        <a href="registro.php" class="button">Registrar</a>
    </div>
    <footer>
        <p>&copy; 2024 Seu Nome ou Empresa. Todos os direitos reservados.</p>
    </footer>
</body>
```

```
</html>
```

No próximo capítulo será feita análise do acesso do usuário na aplicação web login, através do arquivo “login.php”.

4. Acesso a Entrada do Usuário na Aplicação Web Login.

Nesse capítulo será mostrado o acesso, entrada ou ingresso do usuário na aplicação web login, através do arquivo “login.php” conforme a listagem 4 abaixo:

Listagem 4 – Arquivo “login.php”:

```
<?php
    session_start();
    include('config.php');

    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $email = $_POST['email'];
        $senha = $_POST['senha'];

        // Verificar credenciais
        $sql = "SELECT * FROM usuario WHERE email='$email'";
        $result = $conn->query($sql);

        if ($result->num_rows > 0) {
            $usuario = $result->fetch_assoc();
            if (password_verify($senha, $usuario['senha'])) {
                $_SESSION['usuario_id'] = $usuario['id'];
                $_SESSION['usuario_nome'] = $usuario['nome'];
                $_SESSION['tipo_id'] = $usuario['tipo_id'];
```

```
            // Verifica o Id do Tipo do Usuário e retorna sua Descrição.
            if($_SESSION['tipo_id'] == 1) {
                $_SESSION['tipo_descricao'] = "Administrador";
            } else if($_SESSION['tipo_id'] == 2) {
                $_SESSION['tipo_descricao'] = "Supevisor";
            } else {
                $_SESSION['tipo_descricao'] = "Usuário Comum";
            }
            setcookie("usuario", $usuario['nome'], time() + (86400 * 30), "/"); // 30 dias
            header('location: dashboard.php');
        } else {
            echo "Senha incorreta.";
        }
    } else {
        echo "Usuário não encontrado.";
    }
  }
?>

<form method="post">
   Email: <input type="email" name="email" required><br>
   Senha: <input type="password" name="senha" required><br>
   <input type="submit" value="Login">
</form>
```

Veja a análise desse código-fonte do arquivo “login.php”.

Análise do Código

O que o Código Faz

1. **Início da Sessão**: O código inicia uma sessão para armazenar informações do usuário após o login.
2. **Inclusão do Arquivo de Configuração**: O arquivo config.php é incluído para estabelecer a conexão com o banco de dados.
3. **Verificação do Método de Requisição**: O código verifica se a requisição é do tipo POST, indicando que o formulário foi enviado.
4. **Consulta ao Banco de Dados**: Ele busca um usuário no banco de dados com base no email fornecido.
5. **Verificação da Senha**: Se o usuário for encontrado, a senha é verificada usando password_verify().
6. **Armazenamento em Sessão**: Se as credenciais estiverem corretas, as informações do usuário são armazenadas na sessão e um cookie é definido.
7. **Redirecionamento**: O usuário é redirecionado para o dashboard.php após um login bem-sucedido.

Pontos Positivos

- **Uso de password_verify()**: Isso garante que as senhas sejam verificadas de forma segura.
- **Gerenciamento de Sessões**: A utilização de sessões para manter o estado do usuário é uma boa prática.

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para aprimorar seu código:

4.1. Proteção Contra SQL Injection

Atualmente, o código está vulnerável a ataques de SQL Injection. Utilize prepared statements para evitar esse tipo de vulnerabilidade:

```
$stmt = $conn->prepare("SELECT * FROM usuario WHERE email = ?");

$stmt->bind_param("s", $email);

$stmt->execute();

$result = $stmt->get_result();
```

4.2. Sanitização de Saída

Assim como em outras partes do seu código, é importante sanitizar as saídas para evitar XSS:

```
echo htmlspecialchars("Senha incorreta.");
```

4.3. Feedback ao Usuário

Em vez de apenas exibir mensagens simples, considere usar um sistema de feedback mais amigável, talvez com um alerta ou redirecionamento.

4.4. Validação Adicional

Adicione validação adicional ao email e à senha antes de processar a requisição, garantindo que os dados estejam no formato correto.

4.5. Melhorar a Interface do Formulário

Considere adicionar estilos ao formulário para melhorar a experiência do usuário:

```
<form method="post" style="max-width: 300px; margin: auto;">

  <label for="email">Email:</label>

  <input type="email" name="email" required><br>

  <label for="senha">Senha:</label>

  <input type="password" name="senha" required><br>

  <input type="submit" value="Login">

</form>
```

4.6. Implementar Proteção contra Brute Force

Considere implementar um mecanismo que limite tentativas de login (por exemplo, bloqueio temporário após várias tentativas falhas). Normalmente, implementa-se um mecanismo de controle de login com um número de tentativas que variam de 3 a 5 entradas de dados, como email e senha, por exemplo.

4.7 Exemplo Atualizado

Aqui está uma versão atualizada do seu login.php, incorporando algumas das sugestões mencionadas:

```
<?php

session_start();

include('config.php');


if ($_SERVER['REQUEST_METHOD'] == 'POST') {

   $email = $_POST['email'];

   $senha = $_POST['senha'];


   // Verificar credenciais usando prepared statements

   $stmt = $conn->prepare("SELECT * FROM usuario WHERE email = ?");

   $stmt->bind_param("s", $email);

   $stmt->execute();

   $result = $stmt->get_result();


   if ($result->num_rows > 0) {

      $usuario = $result->fetch_assoc();

      if (password_verify($senha, $usuario['senha'])) {

         $_SESSION['usuario_id'] = $usuario['id'];

         $_SESSION['usuario_nome'] = $usuario['nome'];

         $_SESSION['tipo_id'] = $usuario['tipo_id'];
```

```
        // Verifica o Id do Tipo do Usuário e retorna sua Descrição.
        switch ($_SESSION['tipo_id']) {
            case 1:
                $_SESSION['tipo_descricao'] = "Administrador";
                break;
            case 2:
                $_SESSION['tipo_descricao'] = "Supervisor";
                break;
            default:
                $_SESSION['tipo_descricao'] = "Usuário Comum";
        }
          // sobre o valor de 86.400
         // => 60 segundos * 60 minutos * 24 horas = 86.400 segundos em um dia.
        setcookie("usuario", htmlspecialchars($usuario['nome']), time() + (86400 * 30), "/"); // 30 dias
        header('location: dashboard.php');
        exit();
      } else {
        echo htmlspecialchars("Senha incorreta.");
      }
   } else {
      echo htmlspecialchars("Usuário não encontrado.");
   }
}
?>

<form method="post" style="max-width: 300px; margin: auto;">
   <label for="email">Email:</label>
   <input type="email" name="email" required><br>
   <label for="senha">Senha:</label>
```

```
    <input type="password" name="senha" required><br>
    <input type="submit" value="Login">
</form>
```

No próximo capítulo será feita análise do acesso do usuário na aplicação web login, através do arquivo “logout.php”.

5. Saída do Usuário da Aplicação Web Login.

Nesse capítulo será mostrada a saída do usuário na aplicação web login, através do arquivo “logout.php” conforme a listagem 5 abaixo:

Listagem 5 – Arquivo “logout.php”:

```
<?php
    session_start();
    session_destroy();
    setcookie("usuario", "", time() - 3600, "/"); // Remove cookie
    header('location: login.php');
?>
```

Veja a análise do código-fonte do arquivo “logout.php”.

Análise do Código

O que o Código Faz

1. **Início da Sessão**: O código inicia a sessão para garantir que as informações da sessão possam ser acessadas.
2. **Destruição da Sessão**: A função session_destroy() é chamada para encerrar a sessão atual, removendo todas as variáveis de sessão.
3. **Remoção do Cookie**: O cookie associado ao usuário é removido definindo sua expiração para um tempo no passado.
4. **Redirecionamento**: O usuário é redirecionado para a página de login.

Pontos Positivos

- **Simples e Eficiente**: O código é direto e cumpre sua função de forma eficaz.
- **Remoção do Cookie**: A remoção do cookie é uma boa prática para garantir que as informações do usuário não permaneçam no navegador.

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para aprimorar seu código:

5.1. Verificação da Sessão

Antes de destruir a sessão, você pode verificar se a sessão já está ativa, embora isso não seja estritamente necessário, pode ser uma boa prática em sistemas mais complexos.

5.2. Feedback ao Usuário

Após o logout, você pode fornecer uma mensagem de feedback ao usuário informando que ele foi desconectado com sucesso. Isso pode ser feito utilizando uma variável de sessão ou redirecionando para uma página que exiba essa mensagem.

5.3. Redirecionamento com Mensagem

Se você optar por fornecer feedback, considere redirecionar o usuário para login.php com uma mensagem usando a URL (por exemplo, login.php?msg=logout), e então exibir essa mensagem na página de login.

5.4. Limpeza Adicional

Embora session_destroy() remova as variáveis de sessão, você também pode usar session_unset() antes de destruí-la para garantir que todas as variáveis sejam removidas.

5.5. Exemplo Atualizado

Aqui está uma versão atualizada do seu logout.php, incorporando algumas das sugestões mencionadas:

```
<?php
session_start();
session_unset(); // Limpa todas as variáveis de sessão
session_destroy(); // Destrói a sessão

// Remove o cookie: a partir do tempo atual com diferença de uma hora (60 segundos * 60 minutos = 1 hora).
setcookie("usuario", "", time() - 3600, "/");

// Redireciona para a página de login com uma mensagem
header('location: login.php?msg=logout');
exit();
?>
```

5.6 Exibição da Mensagem no login.php

Para exibir uma mensagem após o logout na página de login, você pode adicionar o seguinte código no início do seu login.php:

```
if (isset($_GET['msg']) && $_GET['msg'] == 'logout') {
   echo "<p>Você foi desconectado com sucesso.</p>";
}
```

No capítulo a seguir será feita análise do registro do usuário na aplicação web login, através do arquivo “registro.php”.

6. Registro do Usuário da Aplicação Web Login.

Nesse capítulo será exibido o registro do usuário na aplicação web login, através do arquivo “registro.php” conforme a listagem 6 abaixo:

Listagem 6 – Arquivo “registro.php”:

```
<?php
   session_start();
   include('config.php');

   if ($_SERVER['REQUEST_METHOD'] == 'POST') {
      $nome = $_POST['nome'];
      $email = $_POST['email'];
      $telefone = $_POST['telefone'];
      $senha = password_hash($_POST['senha'], PASSWORD_DEFAULT);
      $tipo_id = $_POST['tipo_id'];

      // Inserir no banco de dados
      $sql = "INSERT INTO usuario (nome, email, telefone, senha, tipo_id) VALUES ('$nome', '$email', '$telefone', '$senha', '$tipo_id')";

      if ($conn->query($sql) === TRUE) {
         $_SESSION['msg'] = "Registro criado com sucesso!";
         header('location: login.php');
      } else {
         echo "Erro: " . $sql . "<br>" . $conn->error;
      }
   }
?>

<form method="post">
   Nome: <input type="text" name="nome" required><br>
   Email: <input type="email" name="email" required><br>
```

```
    Telefone: <input type="text" name="telefone"><br>
    Senha: <input type="password" name="senha" required><br>
    Tipo de Usuário:
    <select name="tipo_id">
        <option value="1">Admin</option>
        <option value="2">Supervisor</option>
        <option value="3">Usuário Comum</option>
    </select><br>
    <input type="submit" value="Registrar">
</form>
```

A seguir é feita uma análise do código-fonte do arquivo “registro.php”.

Análise do Código

O que o Código Faz

1. **Início da Sessão**: O código inicia uma sessão para armazenar mensagens ou informações relevantes.
2. **Inclusão do Arquivo de Configuração**: O arquivo config.php é incluído para estabelecer a conexão com o banco de dados.
3. **Verificação do Método de Requisição**: O código verifica se a requisição é do tipo POST, indicando que o formulário foi enviado.
4. **Coleta de Dados**: Os dados do formulário (nome, email, telefone, senha e tipo de usuário) são coletados.
5. **Hash da Senha**: A senha é criptografada usando password_hash().
6. **Inserção no Banco de Dados**: Os dados são inseridos na tabela usuario.
7. **Feedback ao Usuário**: Se a inserção for bem-sucedida, o usuário é redirecionado para a página de login com uma mensagem de sucesso.

Pontos Positivos

- **Uso de Hash para Senhas**: A utilização de password_hash() é uma boa prática para garantir a segurança das senhas.
- **Estrutura Simples**: O código é fácil de entender e seguir.

Sugestões de Melhorias e Implementações

Aqui estão algumas sugestões para aprimorar seu código:

6.1. Proteção Contra SQL Injection

Assim como no login, utilize prepared statements para evitar SQL Injection:

```
$stmt = $conn->prepare("INSERT INTO usuario (nome, email, telefone, senha, tipo_id) VALUES (?, ?, ?, ?, ?)");

$stmt->bind_param("ssssi", $nome, $email, $telefone, $senha, $tipo_id);
```

6.2. Validação dos Dados

Adicione validações para os dados recebidos antes de inseri-los no banco de dados. Por exemplo, verifique se o email já está cadastrado:

```
$stmt = $conn->prepare("SELECT * FROM usuario WHERE email = ?");

$stmt->bind_param("s", $email);

$stmt->execute();

$result = $stmt->get_result();


if ($result->num_rows > 0) {

   echo "Email já cadastrado.";

   exit();

}
```

6.3. Sanitização da Saída

Ao exibir mensagens ou erros, utilize htmlspecialchars() para evitar XSS:

```
echo htmlspecialchars("Erro: " . $conn->error);
```

6.4. Mensagens de Sucesso

Em vez de redirecionar imediatamente após o registro, considere exibir uma mensagem na página de login informando que o registro foi bem-sucedido.

6.5. Melhorar a Interface do Formulário

Adicione estilos ao formulário para melhorar a experiência do usuário e use elementos HTML semânticos.

6.6. Adicionar Campos Opcionais

Se o telefone não for um campo obrigatório, considere adicionar uma validação que permita que ele seja opcional.

6. 7. Exemplo Atualizado

Aqui está uma versão atualizada do seu registro.php, incorporando algumas das sugestões mencionadas:

```
<?php

session_start();

include('config.php');


if ($_SERVER['REQUEST_METHOD'] == 'POST') {

   $nome = $_POST['nome'];

   $email = $_POST['email'];

   $telefone = $_POST['telefone'];

   $senha = password_hash($_POST['senha'], PASSWORD_DEFAULT);

   $tipo_id = $_POST['tipo_id'];

   // Verificar se o email já está cadastrado

   $stmt = $conn->prepare("SELECT * FROM usuario WHERE email = ?");

   $stmt->bind_param("s", $email);

   $stmt->execute();

   $result = $stmt->get_result();


   if ($result->num_rows > 0) {
```

```
        echo htmlspecialchars("Email já cadastrado.");
        exit();
    }

    // Inserir no banco de dados usando prepared statement
    $stmt = $conn->prepare("INSERT INTO usuario (nome, email, telefone, senha, tipo_id) VALUES (?, ?, ?, ?, ?)");
    $stmt->bind_param("ssssi", $nome, $email, $telefone, $senha, $tipo_id);

    if ($stmt->execute() === TRUE) {
        $_SESSION['msg'] = "Registro criado com sucesso!";
        header('location: login.php');
        exit();
    } else {
        echo htmlspecialchars("Erro: " . $stmt->error);
    }
}
?>

<form method="post" style="max-width: 300px; margin: auto;">
    <label for="nome">Nome:</label>
    <input type="text" name="nome" required><br>

    <label for="email">Email:</label>
    <input type="email" name="email" required><br>

    <label for="telefone">Telefone:</label>
    <input type="text" name="telefone"><br>

    <label for="senha">Senha:</label>
    <input type="password" name="senha" required><br>
```

```
    <label for="tipo_id">Tipo de Usuário:</label>
    <select name="tipo_id">
        <option value="1">Admin</option>
        <option value="2">Supervisor</option>
        <option value="3">Usuário Comum</option>
    </select><br>

    <input type="submit" value="Registrar">
</form>
```

No capítulo a seguir mostra como implementar temas com uso do framework Bootstrap em uma aplicação web.

7. Implementação de Temas com uso do Framework Bootstrap para Aplicações Web.

Nesse capítulo será feita a implementação de temas com uso do framework Bootstrap. Para implementar as funcionalidades solicitadas utilizando Bootstrap, será implementado na aplicação web dois temas, sendo: um tema claro e um tema escuro, além de três temas coloridos específicos para diferentes tipos de usuários. Abaixo, apresenta-se a estrutura básica para os arquivos e o código necessário para atender aos requisitos da aplicação web.

Estrutura do Projeto

1. **index.php** - Página inicial.
2. **login.php** - Página de login.
3. **registro.php** - Página de registro.
4. **dashboard.php** - Dashboard para usuários comuns.

5. **dashboard_super.php** - Dashboard para supervisores.
6. **dashboard_admin.php** - Dashboard para administradores.
7. **noticias.php** - Página de notícias.
8. **contato.php** - Página de contato.
9. **sobre.php** - Página sobre.

7.1. Implementação dos Temas com Bootstrap

Primeiro, adicione o Bootstrap ao seu projeto. Você pode usar o CDN ou instalar via npm. Para este exemplo, usaremos o CDN. As seguintes modificações serão feitas no arquivo "index.php" conforme a listagem 7:

Listagem 7 – Arquivo "index.php": atualizado com Framework Bootstrap:

```
<!DOCTYPE html>
<html lang="pt-BR" data-bs-theme="light"> <!-- Tema padrão -->
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/5.3.0/css/bootstrap.min.css"
rel="stylesheet">
  <title>Página Inicial</title>
  <style>
    /* Customização das cores */
    body[data-bs-theme='admin'] {
      background-color: #dc3545; /* Vermelho */
      color: white;
    }
    body[data-bs-theme='supervisor'] {
      background-color: #28a745; /* Verde */
      color: white;
    }
    body[data-bs-theme='common'] {
      background-color: #007bff; /* Azul */
```

```
        color: white;
    }
  </style>
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="index.php">Home</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
          <li class="nav-item"><a class="nav-link" href="noticias.php">Notícias</a></li>
          <li class="nav-item"><a class="nav-link" href="contato.php">Contato</a></li>
          <li class="nav-item"><a class="nav-link" href="sobre.php">Sobre</a></li>
          <?php if (isset($_SESSION['tipo_id'])): ?>
            <?php if ($_SESSION['tipo_id'] == 2): ?>
              <li class="nav-item">
                  <a class="nav-link" href="dashboard_super.php">Supervisor</a>
              </li>
            <?php elseif ($_SESSION['tipo_id'] == 1): ?>
              <li class="nav-item">
                  <a class="nav-link" href="dashboard_admin.php">Administrador</a>
              </li>
            <?php endif; ?>
          <?php endif; ?>
        </ul>
      </div>
    </div>
  </nav>
```

```
    <!-- Conteúdo Principal -->
    <div class="container mt-5">
        <!-- O conteúdo da página vai aqui -->
    </div>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/5.3.0/js/bootstrap.min.js"></script>
</body>
</html>
```

7.2. Páginas Adicionais

a) noticias.php

```
<?php include('header.php'); ?>
<div class="container mt-5">
    <h1>Você está na página de Notícias</h1>
</div>
<?php include('footer.php'); ?>
```

b) contato.php

```
<?php include('header.php'); ?>
<div class="container mt-5">
    <h1>Você está na página de Contatos</h1>
</div>
<?php include('footer.php'); ?>
```

c) sobre.php

```
<?php include('header.php'); ?>
<div class="container mt-5">
    <h1>Você está na página de Sobre</h1>
```

```
</div>

<?php include('footer.php'); ?>
```

7.3. Dashboards Específicos

a) dashboard_super.php

```
<?php session_start(); include('config.php'); if (!isset($_SESSION['usuario_id']) ||
$_SESSION['tipo_id'] != 2) { header('location: login.php'); exit(); } ?>

<!DOCTYPE html>

<html lang="pt-BR" data-bs-theme="supervisor">

<head>

  <!-- Include Bootstrap and other necessary headers -->

</head>

<body>

  <!-- Include Navbar -->

  <div class="container mt-5">

    <h1>Bem-vindo ao Dashboard do Supervisor!</h1>

    <!-- Conteúdo específico do Supervisor -->

  </div>

</body>

</html>
```

b) dashboard_admin.php

```
<?php session_start(); include('config.php'); if (!isset($_SESSION['usuario_id']) ||
$_SESSION['tipo_id'] != 1) { header('location: login.php'); exit(); } ?>

<!DOCTYPE html>

<html lang="pt-BR" data-bs-theme="admin">

<head>

  <!-- Include Bootstrap and other necessary headers -->

</head>

<body>
```

```
    <!-- Include Navbar -->
    <div class="container mt-5">
        <h1>Bem-vindo ao Dashboard do Administrador!</h1>
        <!-- Conteúdo específico do Administrador -->
    </div>
</body>
</html>
```

7.4. Implementação do Tema Claro e Escuro

Para permitir que os usuários alternem entre temas claro e escuro, você pode adicionar um botão que muda o atributo data-bs-theme no elemento <html>:

```
<button id="theme-toggle" class="btn btn-secondary">Alternar Tema</button>

<script>
document.getElementById('theme-toggle').addEventListener('click', function() {
    const currentTheme = document.documentElement.getAttribute('data-bs-theme');
    const newTheme = currentTheme === 'dark' ? 'light' : 'dark';
    document.documentElement.setAttribute('data-bs-theme', newTheme);
});
</script>
```

7.5 Conclusão

Com essa estrutura, você terá um site que atende aos requisitos mencionados, incluindo a implementação de temas utilizando Bootstrap e a criação de páginas adicionais com um menu responsivo.

No próximo capítulo será feita a implementação do controle de sessão e uso de cookies do usuário para permitir a troca de temas do framework Bootstrap em uma aplicação web.

8. Controle de Sessão e Cookies da Aplicações Web para troca de temas do Framework Bootstrap.

Nesse capítulo será criado cookies e variáveis de sessão para armazenar e alterar a escolha dos temas pelos usuários de forma que ele possa escolher entre os temas: claro, escuro ou colorido de acordo com o tipo de usuário que estiver logado no sistema de forma que a cada login ele possa ter o mesmo tema ou outro tema definido pelo usuário.

Para implementar a funcionalidade de escolha de temas (claro, escuro e colorido) que persista entre as sessões dos usuários, você pode usar cookies e variáveis de sessão. Abaixo, é apresentado um guia passo a passo para implementar essa funcionalidade na aplicação web.

8.1. Modificações no Código de Login

Primeiro, vamos garantir que, ao fazer login, o tema escolhido pelo usuário seja armazenado na sessão e que um cookie seja definido para lembrar essa escolha.

Exemplo de login.php Atualizado

Adicione um campo para que o usuário possa escolher o tema durante o login:

```
<form method="post">
    Email: <input type="email" name="email" required><br>
    Senha: <input type="password" name="senha" required><br>
    Tema:
    <select name="tema">
        <option value="light">Claro</option>
        <option value="dark">Escuro</option>
        <option value="admin">Vermelho (Admin)</option>
```

```
    <option value="supervisor">Verde (Supervisor)</option>
    <option value="common">Azul (Comum)</option>
  </select><br>
  <input type="submit" value="Login">
</form>
```

No processamento do login, armazene a escolha do tema na sessão e no cookie:

```
if (password_verify($senha, $usuario['senha'])) {
  $_SESSION['usuario_id'] = $usuario['id'];
  $_SESSION['usuario_nome'] = $usuario['nome'];
  $_SESSION['tipo_id'] = $usuario['tipo_id'];

  // Armazenar o tema escolhido
  $temaEscolhido = $_POST['tema'];
  $_SESSION['tema'] = $temaEscolhido;
  setcookie("tema", $temaEscolhido, time() + (86400 * 30), "/"); // 30 dias

  // Definir a descrição do tipo de usuário
  if($_SESSION['tipo_id'] == 1) {
    $_SESSION['tipo_descricao'] = "Administrador";
  } else if($_SESSION['tipo_id'] == 2) {
    $_SESSION['tipo_descricao'] = "Supervisor";
  } else {
    $_SESSION['tipo_descricao'] = "Usuário Comum";
  }

  header('location: dashboard.php');
}
```

8.2. Modificações nas Páginas do Dashboard

Em cada página do dashboard (e nas outras páginas), você deve verificar se existe um tema armazenado na sessão ou no cookie e aplicar esse tema.

Exemplo de dashboard.php Atualizado

```
<?php
session_start();
include('config.php');

// Verificar se o usuário está logado
if (!isset($_SESSION['usuario_id'])) {
    header('location: login.php');
    exit();
}

// Definir o tema
$tema = isset($_SESSION['tema']) ? $_SESSION['tema'] : 'light'; // Tema padrão
if (isset($_COOKIE['tema'])) {
    $tema = $_COOKIE['tema']; // Se houver cookie, usar o tema do cookie
}
?>

<!DOCTYPE html>
<html lang="pt-BR" data-bs-theme="<?php echo htmlspecialchars($tema); ?>">
<head>
    <!-- Incluir de código do cabeçaho e link do framework Bootstrap e o que for necessário -->
</head>
<body>
    <!-- Incluir Navbar -->
```

```
    <div class="container mt-5">
        <h1>Bem-vindo ao Dashboard!</h1>
        <!-- Conteúdo do Dashboard -->

        <!-- Botão para alternar temas -->
        <button id="theme-toggle" class="btn btn-secondary">Alternar Tema</button>

        <script>
            document.getElementById('theme-toggle').addEventListener('click', function() {
                const currentTheme = document.documentElement.getAttribute('data-bs-theme');
                const newTheme = currentTheme === 'dark' ? 'light' : 'dark';
                document.documentElement.setAttribute('data-bs-theme', newTheme);

                // Atualizar a escolha do tema no cookie
                document.cookie = "tema=" + newTheme + "; path=/; max-age=2592000"; // 30 dias
            });
        </script>

    </div>
</body>
</html>
```

8.3. Aplicação em Outras Páginas

Repita a lógica acima em todas as páginas onde você deseja aplicar a escolha de temas. Isso inclui dashboard_super.php, dashboard_admin.php, index.php, noticias.php, contato.php, e sobre.php.

8.4. Configuração dos Temas Coloridos

Para aplicar os temas coloridos com base no tipo de usuário, você pode modificar a lógica de definição do tema nas páginas:

```
// Definir o tema baseado no tipo de usuário
if (isset($_SESSION['tipo_id'])) {
    switch ($_SESSION['tipo_id']) {
        case 1:
            $tema = 'admin'; // Vermelho para Administrador
            break;
        case 2:
            $tema = 'supervisor'; // Verde para Supervisor
            break;
        default:
            $tema = 'common'; // Azul para Usuário Comum
            break;
    }
}
```

8.5 Conclusão

Com essas implementações, os usuários poderão escolher seu tema preferido durante o login, e essa escolha será persistida entre as sessões usando cookies. Além disso, ao fazer login, o sistema aplicará automaticamente um tema apropriado com base no tipo de usuário.

Pronto, agora a aplicação Web foi finalizada com sucesso!!!