UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática
UFPE

Universidade Federal de Pernambuco
Centro de Informática - CIn

Documento de Teste de Unidade de Software

Sistema de Gerenciamento de Estacionamento

Versão 1.0

Carlos Kaynan de Sousa
Humberto Hugo Silva Campos
Maria Eduarda Lima Gomes

Recife, 20 de Setembro de 2023

Versão	Data	Descrição	Autores
1.0	18/09/2023 19/09/2023 20/09/2023	Sistema de Estacionamento versão 1.0	Carlos Kaynan de Sousa Humberto Hugo Silva Campos Maria Eduarda Lima Gomes

1. Introdução

Este documento descreve os testes de unidade para a função Cadastrar_cliente, que é responsável por cadastrar o cliente no banco de dados do sistema, a fim de que ele possa logar em qualquer dispositivo e marcar sua reserva. O objetivo dos testes é garantir que a função do cadastro e autenticação dos clientes seja feita corretamente.

2. Tabela de cadastro no front-end.

2.1 cadastro de usuário

A classe ‘input-group’ recebe dados dos usuários com os campos “nome”, “e-mail” e “senha”, dados que serão salvos no banco para futuros login do usuário. A classe em HTML é definida da seguinte forma:

```
<div class="input-group">
    <div class="input-box">
        <label for="firstname">Digite seu Nome</label>
        <input id="firstname" type="text" name="firstname" placeholder="Digite seu nome" required>
    </div>

    <div class="input-box">
        <label for="email">E-Mail</label>
        <input id="email" type="email" name="email" placeholder="Digite seu e-mail" required>
    </div>

    <div class="input-box">
        <label for="password">Senha</label>
        <input id="password" type="password" name="password" placeholder="Digite sua senha" required>
    </div>

</div>
```

A função em JS é definida da seguinte forma:

```
const usuarioCadastro = async (req, res) => {
    const { nome, email, senha } = req.body;

    if (!nome) {
        return res.status(401).json({ mensagem: 'O campo nome não foi informado!'});
    };

    if (!email) {
        return res.status(401).json({ mensagem: 'O campo email não foi informado!'});
    };

    if (!senha) {
        return res.status(401).json({ mensagem: 'O campo senha não foi informado!'});
    };

    try {
        const queryClienteEmail = 'select * from usuarios where email = $1'
        const emailExistente = await pool.query(queryClienteEmail, [email]);

        if (emailExistente.rowCount > 0) {
            return res
                .status(400).json({ mensagem: 'E-mail informado já está cadastrado' });
        };

        const senhaCriptografada = await bcrypt.hash(senha, 10);

        const novoUsuario = await pool.query(
            'insert into usuarios (nome, email, senha) values ($1, $2, $3) returning *',
            [nome, email, senhaCriptografada]
        );

        return res.status(201).json(novoUsuario.rows[0]);

    } catch (error) {
        return res.status(500).json({ mensagem: 'Erro interno do servidor' });
    };
};

module.exports = usuarioCadastro;
```

2.2 Cadastro de veículo

A classe ‘input-group’ recebe dados dos veículos que o usuário quer cadastrar, com os campos “Placa”, “Marca”, “modelo”, “cor” e “tipo”, dados que serão salvos no banco para futuras solicitações de vagas no estacionamento. A classe em HTML é definida da seguinte forma:

```
<div class="input-group">
    <div class="input-box">
        <label for="firstname">Placa</label>
        <input id="firstname" type="text" name="firstname" placeholder="Digite a placa do seu carro" required>
    </div>

    <div class="input-box">
        <label for="marca">Marca</label>
        <input id="marca" type="text" name="marca" placeholder="Digite a marca do carro" required>
    </div>
    <div class="input-box">
        <label for="modelo">Modelo</label>
        <input id="modelo" type="text" name="modelo" placeholder="Digite o modelo do voículo" required>
    </div>

    <div class="input-box">
        <label for="cor">Cor</label>
        <input id="cor" type="text" name="cor" placeholder="Digite a cor do veículo" required>
    </div>

</div>

<div class="gender-inputs">
    <div class="gender-title">
        <h6>Tipo de veículo</h6>
    </div>

    <div class="gender-group">
        <div class="gender-input">
            <input id="carro" type="radio" name="carro">
            <label for="carro">Carro</label>
        </div>

        <div class="gender-input">
            <input id="moto" type="radio" name="moto">
            <label for="moto">Moto</label>
        </div>
```

Função Cadastro de veículo em Node.JS é definida da seguinte forma:

```
const veiculoCadastro = async (req, res) => {
    const { placa, marca, modelo, cor } = req.body;

    if (!placa) {
        return res.status(401).json({ mensagem: 'O campo placa não foi informado!'});
    };

    if (!marca) {
        return res.status(401).json({ mensagem: 'O campo marca não foi informado!'});
    };

    if (!modelo) {
        return res.status(401).json({ mensagem: 'O campo modelo não foi informado!'});
    };

    if (!cor) {
        return res.status(401).json({ mensagem: 'O campo cor não foi informado!'});
    };

    try {
        const placaVeiculo = 'select * from veiculo where placa = $1'
        const placaExistente = await pool.query(placaVeiculo, [placa]);

        if (placaExistente.rowCount > 0) {
            return res.status(400).json({ mensagem: 'Carro informado já está cadastrado' });
        };

        const novoVeiculo = await pool.query(
            'insert into veiculo (placa, marca, modelo, cor) values ($1, $2, $3, $4) returning *',
            [placa, marca, modelo, cor]
        );

        return res.status(201).json(novoVeiculo.rows[0]);

    } catch (error) {
        return res.status(500).json({ mensagem: 'Erro interno do servidor' });
    };
};
```

2.3 Casos de Testes de Cadastro

Caso de Teste 1: Lista Vazia

- Entrada: [Nome, E-mail, Senha]
- Saída: mensagem de usuário cadastrado no sistema

Caso de teste 2: Lista Vazia

- Entrada: [Placa, Marca, Modelo, Cor, Tipo]
- Saída: mensagem de carro cadastrado no sistema

3. Tela de login do usuário

A classe ‘login-container’ recebe dados dos usuários previamente cadastrados para fazer login no site, com os campos “Nome de usuário” e

“Senha”, dados que serão recuperados do banco para autenticar e concluir o login do usuário. A classe em HTML é definida da seguinte forma:

```html
<div class="login-container">
    <form>
        <h1>Login</h1>
        <div class="form-group">
            <label for="username">Nome de Usuário:</label>
            <input type="text" id="username" name="username" required>
        </div>
        <div class="form-group">
            <label for="password">Senha:</label>
            <input type="password" id="password" name="password" required>
        </div>
        <div class="form-group">
            <button type="submit"><a href="home.html">Entrar</a></button>
        </div>
    </form>
</div>
```

Função de login em Node.JS da seguinte forma:

```javascript
const usuarioLogin = async (req, res) => {
    const { email, senha } = req.body;

    if (!email) {
        return res.status(401).json({ mensagem: 'O campo email não foi informado!'});
    };

    if (!senha) {
        return res.status(401).json({ mensagem: 'O campo senha não foi informado!'});
    };

    try {
        const usuario = await pool.query(
            'select * from usuarios where email = $1',
            [email]
        );

        if (usuario.rowCount < 1) {
            return res.status(404).json({ mensagem: 'Email invalido' });
        };

        const senhaValida = await bcrypt.compare(senha, usuario.rows[0].senha);

        if (!senhaValida) {
            return res.status(400).json({ mensagem: 'Senha invalida' })
        };

        const token = jwt.sign({ id: usuario.rows[0].id }, senhaJwt, {
            expiresIn: '8h',
        });

        const { senha: _, ...usuarioLogado } = usuario.rows[0];

        return res.json({ usuario: usuarioLogado, token });

    } catch (error) {
        return res.status(500).json({ mensagem: 'Erro interno do servidor' });
    };

```

3.1 Casos de Testes de Cadastro

Caso de Teste 1: Lista Vazia

- Entrada: [Nome, Senha]
- Saída: Usuário logado no sistema

4. Tela de Check-in e Checkout do estacionamento

A classe ‘Checkin-container’ recebe a informação da placa do veículo e o horário desejado para marcar a vaga, com os campos “Placa” e “Horário”, o campo horário deve ser escrito em HH:MM - DD/MM/AAAA,esses dados serão inseridos no banco para marcar a reserva do usuário. A classe em HTML é definida da seguinte forma:

```html
<div class="login-container">
    <form>
        <h1>Check-in</h1>
        <div class="form-group">
            <label for="placa">Placa do carro:</label>
            <input type="text" id="placa" name="placa" required>
        </div>
        <div class="form-group">
            <label for="horario">Horário:</label>
            <input type="text" id="horario" name="horario" placeholder="ex: HH:MM - DD/MM/AAAA" required>
        </div>
        <div class="form-group">
            <button type="submit"><a href="home.html">Marcar horário</a></button>
        </div>
    </form>
</div>
```

Função de Check-in em Node.JS é definida da seguinte forma:

```
const veiculoReserva = async (req, res) => {
    const { placa, horario } = req.body;

    if (!placa) {
        return res.status(401).json({ mensagem: 'O campo placa não foi informado!'});
    };

    if (!horario) {
        return res.status(401).json({ mensagem: 'O campo horario não foi informado!'});
    };

    try {
        const veiculo = await pool.query(
            'select * from veiculo where placa = $1',
            [placa]
        );

        if (veiculo.rowCount < 1) {
            return res.status(404).json({ mensagem: 'Placa invalida' });
        };

        const novoVeiculo = await pool.query(
            'insert into reserva (placa, horario) values ($1, $2) returning *',
            [placa, horario]
        );

        return res.status(201).json(novoVeiculo.rows[0]);

    } catch (error) {
        return res.status(500).json({ mensagem: 'Erro interno do servidor' });
    };
};
```

4.1 Casos de Testes de Check-in

Caso de Teste 1: Lista Vazia

- Entrada: [Placa, Horário]
- Saída: Mensagem de confirmação de agendamento

A classe ‘Checkout-container’ recebe a informação da placa do veículo pra confirmar a saída do mesmo . A classe em HTML é definida da seguinte forma:

```html
<div class="checkout-container">
    <h1>Digite a Placa do seu carro</h1>
    <form>
        <div class="form-group">
            <label for="placa">Placa:</label>
            <input type="text" id="placa" name="placa" required>
        </div>
        <div class="form-group">
            <button type="submit">Checkout</button>
        </div>
    </form>
</div>
```

Função de Checkout em Node.JS é definida da seguinte forma:

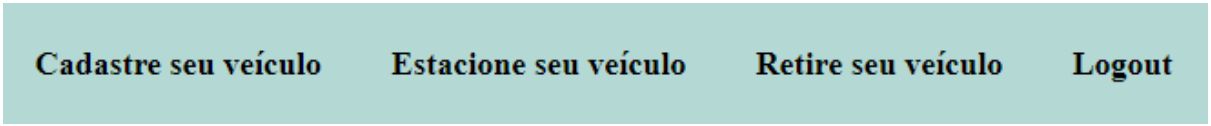
```javascript
const horarioAtual = new Date();

const checkout = async (req, res) => {
    const { placa } = req.body;

    if (!placa) {
        return res.status(401).json({ mensagem: 'O campo placa não foi informado!'});
    };

    try {
        const placaVeiculo = 'select * from reserva where placa = $1'
        const placaExistente = await pool.query(placaVeiculo, [placa]);

        const horarioConsulta = 'select horario from reserva where placa = $1'
        const horarioEntrada = await pool.query(horarioConsulta, [placa]);


        if (placaExistente.rowCount < 1) {
            return res.status(400).json({ mensagem: 'Carro informado não está estacionado' });
        };

        const [dia, mes, ano, horas, minutos] = horarioEntrada.rows[0].horario.match(/\d+/g);

        const dataHoraDesejada = new Date(ano, mes - 1, dia, horas, minutos);

        const valorCobrado =  Math.floor(((horarioAtual - dataHoraDesejada) / 36e5) * 5);

        await pool.query('delete from reserva where placa = $1', [placa]);

        return res.status(204).json(valorCobrado);

    } catch (error) {
        return res.status(500).json('Erro interno do servidor');
    };

};

module.exports = checkout;
```

4.2 Casos de Testes de Checkout

Caso de Teste 1: Lista Vazia

- Entrada: [Placa]
- Saída: Ticket com preço a pagar pelo tempo solicitado

5.0 Botões de navegação da Homepage



O usuário clica no botão de navegação para acessar a página desejada. Os botões de navegação estão funcionando, exceto o botão de “Logout”, devido ao fato de não conseguirmos fazer com que o usuário fique logado.

Classe HTML com botões de navegação:

```
<ul>
    <li><a href="cadastro_car.html">Cadastre seu veículo</a></li>
    <li><a href="checkin.html">Estacione seu veículo</a></li>
    <li><a href="checkout.html">Retire seu veículo</a></li>
    <li><a href="#">Logout</a></li>
</ul>
```

5.1 Casos de Testes dos Botões de Navegação

Caso de Teste 1:

- Entrada: Botão [cadastre seu veículo]
- saída: È iniciada página de cadastro de veículo

Caso de Teste 2:

- Entrada: Botão [estacione seu veículo]
- saída: È iniciada página de check-in

Caso de Teste 3:

- Entrada: Botão [retire seu veículo]
- saída: È iniciada página checkout

6. Unidade não funcionais

Não conseguimos integrar o botão de “cadastrar usuário” na tela de Home, devido a falta de tempo e desligamento repentino do integrante do grupo que estava a frente do Frontend.

7. Resultados esperados

Espera-se que todos os casos de testes sejam bem-sucedidos, com o cadastro e efetivação da vaga no estacionamento, correspondendo a entrada e saída do cliente no estacionamento de forma fácil e rápida.

8. conclusão

Este documento de teste de unidade de software descreve os testes realizados na funções cadastrar_usuario, cadastrar_veiculo, login, Check-in e checkout. para garantir que ela funcione conforme o esperado. Testes de unidade são essenciais para verificar a funcionalidade correta das unidades individuais de código, ajudando a manter a qualidade do software.