Jour 3 & 4 - Job- 3 Test Unitaire

Welcome to Unit Tests - Part 3

Calculette Unitaire !

Toujours des calculs et des tests

Objectif :

À partir des deux interfaces de calculette fournies (PHP et JS), vous devez écrire vous-mêmes les tests unitaires correspondant à la logique de calcul.

📂 Fichiers fournis en annexes

- Calculator_PHP.php (calculette web fonctionnelle en PHP)

- Calculator_JS.html (calculette web fonctionnelle en JavaScript)

- calculator.php (classe PHP `Calculator`)

- calculator.js (fonction JS `calculate()`)

- calculator.css (css pour calculator_JS.html)

- Faites des captures d'écran pour illustrer un readme.md
- Les images seront intégré au readme
- Stocké dans un dossier image du projet
- Puis déposé sur github à chaque étape
- Avec des commits régulier
- Avec un nom et une description explicite et claire

(Pas de nom du genre "mon commit" ou "commit pierre", un nom pro c'est "modification de la connection à la BDD" par exemple, pareil pour les branches "Connection à la BDD" ou "formulaire de contact")

- Créez un dossier `tests/` à la racine du projet
- Pour la calculette PHP :
    - Créez un fichier `CalculatorTest.php` dans `tests/`
    - Écrivez des tests avec PHPUnit
    - À tester :
        - L'addition
        - La soustraction
        - La multiplication
        - La division
        - Le comportement en cas de division par zéro
- Pour la calculette JavaScript :
    - Créez un fichier `calculator.test.js` dans `tests/`
    - Écrivez des tests avec Jest
    - À tester :
        - L'addition
        - La soustraction
        - La multiplication
        - La division
        - Les priorités (`2+3*4`)
        - Les parenthèses (`(2+3)*4`)
        - Le comportement en cas d'expression invalide (`2+bad`)

🧪 Validation

- Les tests doivent passer sans erreur
- Les messages d'erreur doivent être gérés proprement
- Chaque test doit être précis, isolé et justifié

🎁 Bonus

- Ajouter un test pour une chaîne vide (`""`)
- Afficher un résumé de test complet dans le terminal

Compétences visées

- Comprendre et appliquer les principes des tests unitaires en PHP et JavaScript
- Mettre en place une stratégie de tests sur un projet existant
- Utiliser les outils PHPUnit (PHP) et Jest (JS) pour valider la logique métier
- Organiser proprement un projet avec séparation du code et des tests
- Détecter, corriger et prévenir les erreurs de calcul dans un projet web

Rendu

- Faites des captures d'écran à chaque étape pour montrer la commande et le résultat même si c'est un résultat en erreur
- Rendre un "readme.md" avec ces captures d'écran à l'intérieur stockées sur votre github dans un dossier image en annotant chaque capture en expliquant le détail de chaque action et commande de manière structurée en MarkDown dans un readme.md.
- partager votre projet github avec votre formateur
- Un dépôt GitHub nommé `test-unitaire-debutant` contenant :

  - Tous les fichiers du projet

  - Un fichier `README.md` avec captures d'écran et explication en Markdown

  - Des commits réguliers et nommés proprement

Base de connaissances

- Documentation officielle PHP
  - https://www.php.net/manual/fr/
- PHPUnit – Outil de test unitaire PHP
  - https://phpunit.de/documentation.html
- Documentation Node.js
  - https://nodejs.org/fr/docs/
- Jest – Guide officiel pour les tests en JavaScript
  - https://jestjs.io/docs/getting-started
- MDN – JavaScript eval()
  - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/eval
- MDN – Expressions régulières (regex)
  - https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Regular_expressions
- MDN – Formulaires HTML (<form>)
  - https://developer.mozilla.org/fr/docs/Web/HTML/Element/form
- MDN – Événements clavier et DOM (keydown, event)
  - https://developer.mozilla.org/fr/docs/Web/API/KeyboardEvent
- MDN – document.querySelector et manipulation DOM
  - https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector
- W3Schools – Exemple de calculatrice JavaScript simple
  - https://www.w3schools.com/howto/howto_js_calculator.asp

Annexes :

Calculator_PHP.php

```
<?php
require_once 'calculator.php';
session_start();

if (!isset($_SESSION['expression'])) $_SESSION['expression'] = '';
if (!isset($_SESSION['history'])) $_SESSION['history'] = [];
if (!isset($_SESSION['raw'])) $_SESSION['raw'] = '';

$result = '';

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

    // Réinitialiser visuellement l'affichage si on tente de continuer après un '='
    if (strpos($_SESSION['expression'], '=') !== false && !in_array($btn, ['=', 'C', 'CE'])) {
        $_SESSION['expression'] = '';
    }

    if ($btn === 'C') {
        $_SESSION['expression'] = '';
        $_SESSION['raw'] = '';
    } elseif ($btn === 'CE') {
        $_SESSION['history'] = [];
    } elseif ($btn === '=') {
        $calculator = new Calculator();
```

```
        try {
            $result = $calculator->calculate($_SESSION['raw']);
            $entry = $_SESSION['raw'] . ' = ' . $result;
            $_SESSION['expression'] = $entry;
            $_SESSION['raw'] = (string)$result;  // conserve le résultat pour continuer à calculer
            $_SESSION['history'][] = $entry;
        } catch (Exception $e) {
            $entry = $_SESSION['raw'] . ' = ' . $e->getMessage();
            $_SESSION['expression'] = $entry;
            $_SESSION['raw'] = '';
            $_SESSION['history'][] = $entry;
        }
    } else {
        if ($_SESSION['raw'] === '' && in_array($btn, ['+', '-', '*', '/'])) {
            $_SESSION['expression'] .= '0' . $btn;
            $_SESSION['raw'] .= '0' . $btn;
        } else {
            $_SESSION['expression'] .= $btn;
            $_SESSION['raw'] .= $btn;
        }
    }
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Calculette Graphique PHP avec Historique</title>
    <style>
        body { font-family: Arial; display: flex; justify-content: center; margin-top: 30px; }
```

```
        .calculator { border: 1px solid #333; padding: 20px; border-radius: 10px; background: #f4f4f4; }
        .display { height: 40px; text-align: right; padding: 10px; font-size: 20px; background: #fff; border: 1px solid #ccc; margin-bottom: 10px; width: 220px; }
        .buttons form { display: grid; grid-template-columns: repeat(4, 1fr); gap: 10px; }
        button { padding: 15px; font-size: 18px; cursor: pointer; }
        .history { margin-top: 20px; font-size: 14px; background: #fff; padding: 10px; border: 1px solid #ccc; height: 100px; overflow-y: auto; }
    </style>
</head>
<body>
    <div class="calculator">
        <div class="display"><?= htmlspecialchars($_SESSION['expression']) ?></div>
        <div class="buttons">
            <form method="post">
                <?php
                $buttons = ['7', '8', '9', '/',
                            '4', '5', '6', '*',
                            '1', '2', '3', '-',
                            '0', '.', 'C', '+',
                            '(', ')', '=', 'CE'];
                foreach ($buttons as $b) {
                    echo '<button type="submit" name="btn" value="' . $b . '">' . $b . '</button>';
                }
                ?>
            </form>
        </div>
        <div class="history">
            <strong>Historique :</strong><br>
            <?php foreach (array_reverse($_SESSION['history']) as $line) {
```

```
            echo htmlspecialchars($line) . "<br>";
        } ?>
    </div>
</div>

<script>
document.addEventListener('keydown', function(event) {
    const validKeys = ['0','1','2','3','4','5','6','7','8','9','.','+','-','*','/','(',')'];
    const form = document.querySelector('.buttons form');
    if (!form) return;

    if (validKeys.includes(event.key)) {
        const btn = document.createElement('input');
        btn.type = 'hidden';
        btn.name = 'btn';
        btn.value = event.key;
        form.appendChild(btn);
        form.submit();
    } else if (event.key === 'Enter') {
        const btn = document.createElement('input');
        btn.type = 'hidden';
        btn.name = 'btn';
        btn.value = '=';
        form.appendChild(btn);
        form.submit();
    } else if (event.key === 'Escape') {
        const btn = document.createElement('input');
        btn.type = 'hidden';
        btn.name = 'btn';
        btn.value = 'C';
```

```
        form.appendChild(btn);
        form.submit();
    }
});
</script>

</body>
</html>
```

Calculator_js.php

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Calculatrice JavaScript</title>
    <link rel="stylesheet" href="calculator.css">
</head>
<body>
    <div class="calculator">
        <input id="result" type="text" class="calc_resultat" readonly>
        <div class="buttons">
            <button onclick="clearResult()">C</button>
            <button onclick="appendValue('7')">7</button>
            <button onclick="appendValue('8')">8</button>
            <button onclick="appendValue('9')">9</button>
            <button onclick="appendOperator('+')">+</button>
            <button onclick="appendValue('4')">4</button>
            <button onclick="appendValue('5')">5</button>
            <button onclick="appendValue('6')">6</button>
            <button onclick="appendOperator('-')">-</button>
            <button onclick="appendValue('1')">1</button>
            <button onclick="appendValue('2')">2</button>
            <button onclick="appendValue('3')">3</button>
            <button onclick="appendOperator('*')">*</button>
            <button onclick="appendValue('0')">0</button>
            <button onclick="appendValue('.')">.</button>
            <button onclick="calculate()">=</button>
            <button onclick="appendOperator('/')">/</button>
```

```
        </div>
    </div>
    <script src="calculator.js"></script>
</body>
</html>
```

Calculator.php

```
<?php
class Calculator {
    public function calculate($expression) {
        $expression = str_replace(['×', '÷', '-', '–', '—'], ['*', '/', '-', '-', '-'], $expression);
        $expression = trim($expression);

        try {
            $result = eval("return $expression;");
        } catch (Throwable $e) {
            throw new RuntimeException("Erreur de calcul");
        }

        if ($result === false) {
            throw new RuntimeException("Erreur de calcul");
        }

        return $result;
    }
}
?>
```

Calculator.js

```
let currentInput = "";

function updateDisplay() {
    if (typeof document !== 'undefined') {
        document.getElementById('result').value = currentInput;
    }
}

function appendValue(value) {
    currentInput += value;
    updateDisplay();
}

function appendOperator(operator) {
    if (!/[+\-*/]$/.test(currentInput)) {
        currentInput += operator;
        updateDisplay();
    }
}

function clearResult() {
    currentInput = "";
    updateDisplay();
}

function calculate() {
    try {
```

```
        const result = eval(currentInput);
        currentInput = result.toString();
    } catch (e) {
        currentInput = "Erreur";
    }
    updateDisplay();
}

// Fonction exportable pour Jest
function evaluateExpression(expression) {
    if (!/^[0-9+\-*/().\s]+$/.test(expression)) {
        throw new Error("Expression invalide");
    }
    return eval(expression);
}

if (typeof module !== 'undefined') {
    module.exports = {
        calculate: evaluateExpression
    };
}
```

Calculator.css

```
.calculator {
    width: 240px;
    margin: 50px auto;
    padding: 20px;
    border: 2px solid #0ff;
    border-radius: 10px;
    background-color: #000;
}
.calc_resultat {
    width: 100%;
    height: 40px;
    font-size: 1.5em;
    margin-bottom: 10px;
    text-align: right;
    padding: 5px;
    border-radius: 5px;
}
.buttons {
    display: grid;
    grid-template-columns: repeat(4, auto);
    gap: 5px;
}
button {
    padding: 5px 8px;
    font-size: 1em;
    background-color: #000;
    color: #0ff;
    border: 1px solid #0ff;
```

```
    border-radius: 5px;
    cursor: pointer;
}
```