Coding Standards

As part of our development team, I am establishing the following coding standards for frontend (HTML, CSS, JavaScript) and backend (PHP) to ensure consistency, readability, and maintainability in our projects. These guidelines will help us write cleaner, more efficient, and more secure code.

Frontend Coding Standards:

HTML Coding Standards:

File Names:

- We use lowercase letters and hyphens (-) to separate words.
- We void spaces, underscores, or camelCase.

Example:

index.html
contact-us.html
about.html

Element Naming:

- We use lowercase for element names (e.g., ,).
- We always close all elements properly.
- We use semantic HTML tags where appropriate (e.g., <header>, <section> instead of `<div>`).

Example:

```
<header>
    <h1>Welcome</h1>
</header>
```

Class & ID Naming

- We use lowercase with hyphens (-) for multi-word class and ID names.
- We use classes for styling and IDs for unique elements.

Example:

```
<div class="container"></div>
<button id="submit-btn"></button>
```

Attribute Formatting

- We use double quotes for attribute values.
- We use boolean attributes without values (e.g., checked, disabled).

Example:

```
<input type="checkbox" checked>
<img src="image.jpg" alt="Description">
```

Indentation & Spacing

- We use 2 spaces per indentation level.
- We avoid unnecessary blank lines.

Example:

```

<nav>
  <ul>
    <li><a href="#">Home</a></li>
  </ul>
</nav>
```

Comments

- We use comments to explain sections of your code, especially for complex parts.
- We keep comments concise and relevant.

Example:

```
<!-- Main navigation section -->
<nav>
  <ul>
    <li><a href="#">Home</a></li>
  </ul>
</nav>
```

CSS Coding Standards:

File Names:

- We use lowercase letters and hyphens (-).
- We keep file names short and meaningful.

Example:

styles.css
responsive.css
header.css

Selector Naming:

- We use lowercase with hyphens (-) for class names.
- We use meaningful, semantic names.

Example:

```
.container {}
.nav-bar {}
.btn-primary {}
```

Property Formatting & Order:

- We write one property per line.
- We use a space after colons (`:`).

- We use hex codes for colors.
- We follow logical property ordering (position, box-model, typography, visual effects).

Example:

```
.button {
  position: relative;
  display: inline-block;
  width: 100px;
  padding: 10px;
  background-color: #ff6600;
  color: white;
}
```

Variables:

- We define CSS variables inside :root.

Example:

```
:root {
  --primary-color: #ff6600;
  --font-size-base: 16px;
}
```

Keyframes Naming:

- We use lowercase letters and hyphens (-).
- We use a descriptive action-based name.

Example:

```
@keyframes fade-in {
  from { opacity: 0; }
  to { opacity: 1; }
}
```

Indentation & Formatting:

- We use 2 spaces for indentation.
- We keep CSS properties alphabetically ordered where possible.
- We avoid unnecessary units (e.g., margin : 0 ; instead of margin : 0px ;).

Example:

```
.card {
  background-color: #fff;
  border-radius: 8px;
  padding: 20px;
}
```

JavaScript Coding Standards:

- We follow the ECMAScript 6+ (ES6+) standard.
- We use meaningful variable and function names.
- We keep the code DRY (Don't Repeat Yourself).
- We write modular, reusable, and maintainable code.
- We avoid global variables.
- We ensure code readability and consistency.

Formatting

- We use 2 spaces for indentation.
- We limit lines to 80-100 characters.
- We use camelCase for variable and function names.
- We use PascalCase for class names.
- We use kebab-case for filenames .
- We use semicolons (;) consistently.

Functions

- We use arrow functions where appropriate.
- We keep functions small and focused.
- We use default parameters when applicable.

Coding Segment:

```
// hello-service.js

class HelloService {
  sayHello() {
    return 'Hello';
  }
}

export default HelloService;
```

Backend Coding Standards:

1. General PHP Standards

- We ollow PSR-12 coding standards.
- Use strict types where applicable: `declare(strict_types=1);`
- Use meaningful variable and function names.
- Keep code modular and reusable.
- Use OOP principles when possible.

2. Code Formatting

- Use 4 spaces for indentation.
- Use camelCase for variable and function names.
- Use PascalCase for class names.
- Use snake_case for database field names.
- End PHP files with a single newline character.

Coding Segment:

```
<?php



declare(strict_types=1);
```

```php
class User {
    private string $name;

    public function __construct(string $name) {
        $this->name = $name;
    }

    public function getName(): string {
        return $this->name;
    }
}
```

3. Security Best Practices

- We always sanitize user inputs.
- We use prepared statements for SQL queries (PDO or `MySQLi` with parameterized queries).
- We avoid using `eval()`.
- Escape output to prevent XSS attacks.
- Use environment variables (`.env`) for sensitive data like database credentials.

Coding Segment:

```php
// Prevent SQL Injection

$statement = $pdo->prepare("SELECT * FROM users WHERE email =
:userEmail");

$statement->execute(['userEmail' => $userEmail]);

$userData = $statement->fetch();
```

```
// Prevent XSS attacks

$sanitizedUsername = htmlspecialchars($_POST['username'], ENT_QUOTES,
'UTF-8');

echo "Welcome, $sanitizedUsername!";
```

4. Error Handling

- Use `try...catch` blocks for handling exceptions.
- Log errors instead of displaying them directly.

Coding Segment:

```
try {

    $operationResult = performCriticalOperation();

}

catch (Exception $exception) {

    error_log($exception->getMessage());

}
```