Front-end Member

1. Master Styling and Layout Techniques with HTML and CSS

Objective: The student demonstrates proficiency in styling and laying out web pages using HTML elements and CSS, including advanced techniques like flexbox, pseudo-elements, media queries, and CSS Grid.

Criteria	Poor - 0	Fair - 1	Good - 2	Excellent - 3
HTML Structure	Incorrect or missing use of HTML elements, affecting structure and semantics.	Basic structure is in place, but with misuse of HTML elements or improper semantics.	Mostly correct use of HTML elements and structure, with minor issues.	Clean, semantic, and well-organized HTML structure that follows best practices.
CSS Styling	Poor or missing CSS, resulting in a non-functional or unstyled page.	Basic styles are applied, but inconsistencies or errors are present.	Mostly consistent and functional CSS, with minor styling issues.	CSS is applied effectively with advanced techniques, achieving a clean, modern design.
Responsive Design	No responsiveness; layout breaks on different screen sizes.	Some responsiveness, but layout has significant issues on smaller screens.	Mostly responsive layout, with minor issues on specific screen sizes.	Fully responsive layout using media queries and modern techniques, ensuring a seamless experience across devices.
Advanced CSS Techniques	No use of advanced techniques like flexbox, grid, or pseudo-elements.	Basic use of advanced techniques, but with incorrect or inefficient implementation.	Mostly correct use of advanced techniques, with minor inefficiencies.	Masterful use of flexbox, grid, and pseudo-elements to achieve sophisticated layouts and styles.

2. Leverage JavaScript for Dynamic Interactions and DOM Manipulation

Objective: The student is proficient in using JavaScript to manipulate the DOM, employing advanced techniques such as destructuring, the spread operator, async/await, and looping to create dynamic, interactive web applications.

Criteria	Poor - 0	Fair - 1	Good - 2	Excellent - 3
DOM Manipulation	No or incorrect use of JavaScript for DOM manipulation.	Basic DOM manipulation is achieved, but with errors or inefficient code.	DOM manipulation works correctly, with minor inefficiencies.	Efficient and advanced DOM manipulation, using best practices and modern techniques.
Dynamic Interactions	No or ineffective use of JavaScript for dynamic features.	Basic interactivity is added, but with notable issues or poor performance.	Dynamic interactions mostly work as intended, with minor issues.	Seamless, performant dynamic interactions using modern JavaScript features like async functions and event listeners.
JavaScript Features	No or incorrect use of destructuring, spread operator, async/await, or loops.	Basic use of advanced JavaScript features, but with errors or poor implementation.	Mostly correct use of advanced JavaScript features, with minor inefficiencies.	Proficient use of destructuring, spread operator, async/await, and loops to write clean, efficient, and scalable code.
Event Handling	No event handling or event handling is incorrect.	Basic event handling works, but with errors or inefficiencies.	Event handling works correctly, with minor issues or inefficiencies.	Event handling is efficient, scalable, and correctly implemented using modern

				JavaScript techniques.

3. Ensure Quality and Accessibility with Testing and SEO Best Practices

Objective: The student ensures the quality of web applications using End-to-End testing with Cypress, prioritizes accessibility, and implements SEO best practices to optimize the user experience and search engine visibility.

Criteria	**Poor - 0**	**Fair - 1**	**Good - 2**	**Excellent - 3**
Test Coverage	No or minimal test coverage. Critical functionality is untested.	Basic test coverage is present but important edge cases are missing.	Test coverage is mostly complete, with a few minor features or edge cases untested.	Comprehensive test coverage across all major features, including edge cases and error handling.
Test Organization and Structure	Tests are disorganized or unstructured, making them difficult to understand or maintain.	Basic organization of tests is present, but structure is inconsistent or confusing.	Tests are mostly well-structured, but with minor issues in consistency or clarity.	Tests are organized and structured clearly, following best practices, making them easy to understand and maintain.
Assertions	No or incorrect assertions used in tests.	Basic assertions are present, but they lack thoroughness or are used incorrectly.	Assertions are mostly correct, but some are incomplete or inefficient.	Strong and comprehensive assertions are used, testing for expected outcomes in various scenarios.
Test Performance	Tests are slow or unreliable, frequently leading to	Basic performance is achieved, but tests are still slow or	Tests are mostly efficient and reliable, with minor	Tests run efficiently, are reliable, and provide fast feedback

	timeouts or false positives.	occasionally unreliable.	performance issues.	without timeouts or errors.
Error Handling in Tests	No tests include scenarios for error handling.	Basic error handling tests are present, but they are incomplete or inaccurate.	Error handling is mostly tested, but with minor gaps.	Comprehensive testing for error handling, including edge cases, with clear and effective tests.
Mocking and Stubbing	No use of mocks or stubs, relying on real services, which makes tests brittle or slow.	Basic mocks or stubs are used, but they are incomplete or incorrectly implemented.	Mocks and stubs are mostly correct, but with minor issues in accuracy or completeness.	Effective use of mocks and stubs to isolate tests, ensuring tests are fast, reliable, and independent of external services.
Cross-Browser Testing	No consideration for cross-browser testing.	Basic cross-browser testing is conducted, but with major gaps in coverage.	Cross-browser testing is mostly completed, but with minor issues in functionality on specific browsers.	Comprehensive cross-browser testing ensures the application functions properly across all major browsers.
Test Scenarios and User Flows	No or incorrect test scenarios that fail to reflect actual user interactions.	Basic test scenarios are present, but they are incomplete or do not fully reflect real user flows.	Test scenarios mostly reflect real user flows, with minor issues in coverage.	Test scenarios are well-crafted, accurately simulating real-world user flows and edge cases.
Accessibility	Accessibility features are missing or incorrectly implemented.	Basic accessibility features are present, but with significant issues.	Mostly correct implementation of accessibility features, with minor issues.	Fully accessible web application, following best practices such as ARIA roles,

				semantic HTML, and keyboard navigation.
SEO Best Practices	No SEO considerations are implemented.	Basic SEO practices are in place, but with notable issues.	Mostly correct SEO practices, but with room for improvement.	Fully optimized for search engines, with well-structured metadata, alt-text, and semantic HTML.

4. Build Modern Web Applications with React

Objective: The student builds modern web applications using React, mastering component rendering, state management with hooks, conditional rendering, and implementing custom hooks. Emphasis on Test Driven Development (TDD) is also applied.

Criteria	Poor - 0	Fair - 1	Good - 2	Excellent - 3
Component Rendering	No or incorrect rendering of components in React.	Basic component rendering works, but with issues or inefficiencies.	Components render correctly, with minor issues in structure or performance.	Components are rendered efficiently, with optimal structure, reusability, and performance.
State Management (Hooks)	No or incorrect use of state in React.	Basic state management works, but with notable issues.	State is managed mostly correctly using hooks, with minor issues or inefficiencies.	Proficient use of hooks (useState, useEffect, etc.) for state management, ensuring clean and efficient code.
Conditional Rendering	No or incorrect use of conditional rendering.	Basic conditional rendering works, but with	Conditional rendering is mostly correct, with minor	Efficient and correct use of conditional rendering, ensuring

		errors or inefficiencies.	inefficiencies or logic errors.	smooth and logical component behaviour.
Test Driven Development (TDD)	No or ineffective use of TDD principles in React development.	Basic TDD practices are followed, but with limited tests or incorrect implementation.	TDD is mostly followed, with minor gaps in coverage or test quality.	TDD is consistently applied, with comprehensive and effective tests ensuring robust React applications.