TOSA

Your skills. Your advantage.

Tosa Web Developer

Exam Blueprint

contact@isograd.com

Table of Contents

Introduction

to the Tosa® Exam Blueprint

For Tosa® Assessment and Certification

Tosa® (Test on Software Applications)

Tosa® assessments and certifications are designed to determine a candidate's proficiency level by evaluating their skills in office software and digital tools commonly used in a professional or educational environment.

These tests are specifically developed to validate the professional competencies of candidates looking to enhance their employability (employees, students, job seekers, and individuals undergoing career transitions).

Tosa® assessments and certifications are adaptive tests, developed using scientific methodologies. The scoring is based on Item Response Theory (IRT). The test algorithm adapts to each candidate's response in real time, adjusting the difficulty level of subsequent questions until it precisely determines the candidate's skill level by calculating the upper limit of their competencies. As a result, the tests provide a detailed and unique diagnosis of each candidate's abilities.

The rigor and reliability of Tosa® tests stem from the combination of a mathematical model for analyzing question difficulty and the relevance of the questions selected for each candidate (IRT).

Tosa® Exam Blueprint Objective

This exam blueprint outlines all the skills assessed within the domains and sub-domains of the Tosa® assessment and certification tests for Web Developer.

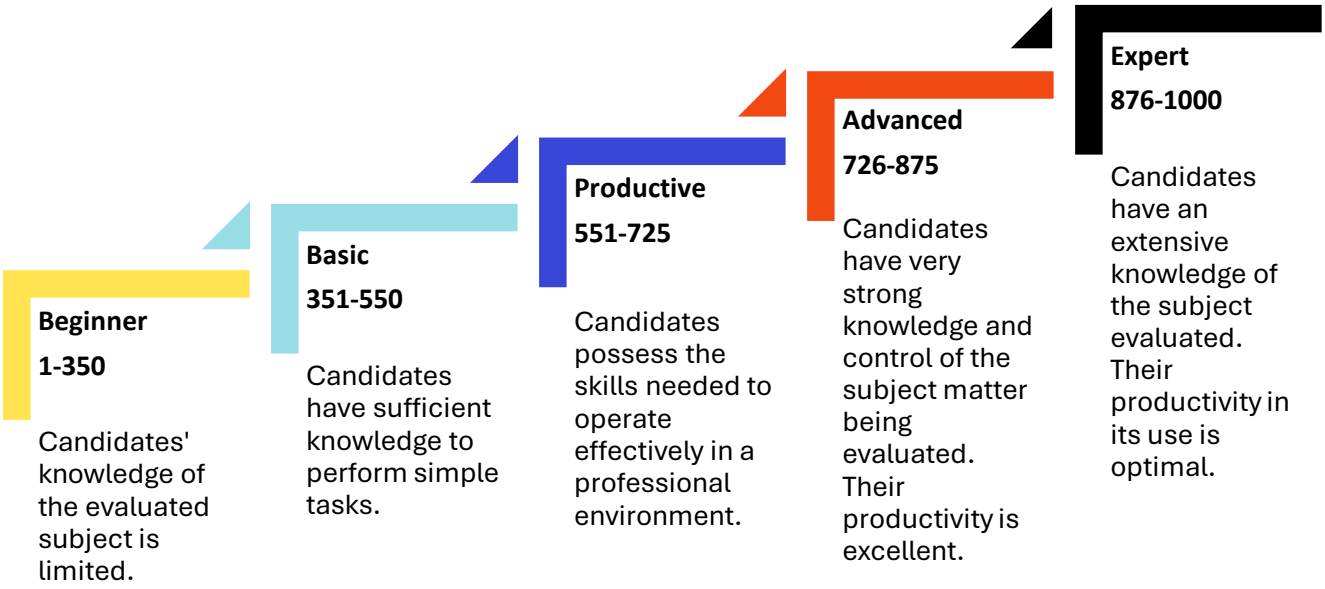
Tosa® assessment and certification solutions are designed to determine learners' proficiency levels using a single scoring scale—ranging from 0 to 1000 for certification—and divided into five levels, from "Beginner" to "Expert," for the assessment.

The purpose of this blueprint is to specify the technical knowledge expected at each level and within each of the four main skill categories of Web Developer. It is intended to help identify the most appropriate teaching or training programs to match a learner's target score.

Unique Tosa® Scoring

The Tosa® assessments and certifications are based on a unique score, divided into five levels.

- Ranging from 1 to 1000 for the certification.
- Divided into five levels, from Beginner to Expert, for the assessment.



Certification failed	**Certification failed**	**Certification earned**	**Certification earned**	**Certification earned**
Certificate of completion issued	Certificate of completion issued	Diploma & Credly digital badge issued	Diploma & Credly digital badge issued	Diploma & Credly digital badge issued

About the Tosa® Web Developer certification

The Tosa Web Developer Certification relies on a database of more than 200 questions. It is composed of 35 questions from the question database and lasts for 1 hour and 30 minutes. The algorithm adapts to each of the candidate's answers to adjust the difficulty level of the questions until reaching the candidate's skill limit. This ensures a precise and accurate result.

Since the test is adaptive, the series of questions that a candidate gets is unique for each test. This algorithm allows for a more accurate evaluation of the candidate's level. It also limits cheating and the memorization of questions on different passages.

Our platform allows individuals to take the certification in a classroom setting, an approved testing center, or remotely via our integrated asynchronous online proctoring solutions.

Our remote proctoring solutions provide added flexibility for both the administrator and the candidate, allowing the certification exam to be taken anywhere, at any time. The candidate only needs an internet connection and a computer equipped with a working webcam and microphone.

Candidates receive a numeric score out of 1000 points. This score corresponds to one of five proficiency levels. Candidates who score between 1 and 550 points do not earn the certification. They receive a certificate of completion in lieu of a diploma. Candidates who score 551 points or above earn the certification and will receive a diploma by email within five business days. If a candidate scores 551 points or above, they will also be eligible to receive a digital Credly badge.

There are no prerequisites to be eligible to take the exam, but to ensure a candidate is well prepared on exam day, we suggest they:

- Take at least one Tosa Web Developer adaptive assessment to estimate their level and get familiar with the test format
- Use the free practice tests on our website for training
- Follow an e-learning or training course (average duration per level is between 10 and 15 hours per certification, so around 150 hours total)

Tosa certification diplomas are valid for three years from the date of issue. This three-year period has been set to ensure that our certifications are consistently accurate and relevant, taking into account software version updates as well as the natural evolution of a candidate's skills over time. Limiting the certification period also reflects the need for life-long learning and continual professional development.

Tosa certifications can be retaken when they expire. Earners willing to improve their score and proficiency level can retake the exam at any time.

Tosa Sequence for Progressive Skills Development

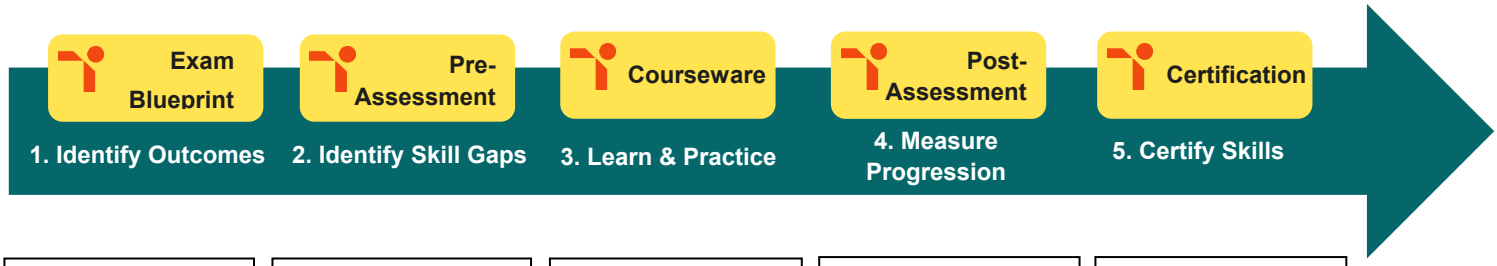
Students and professionals can tailor their certification journey with Tosa through vertical progression, demonstrating increasing proficiency in web development. Starting at a productive level, users can advance to advanced and expert levels as their skills grow. This clear path encourages continuous improvement and validates each stage of their development. Tosa's structure makes it easy to track progress and showcase evolving expertise.

Isograd Learning Lab

The Isograd Learning Lab is a multifaceted and adaptable courseware solution designed to help learners prepare for Tosa certification exams. It offers personalized, self-paced, and fully interactive learning experiences, equipping candidates with the essential digital skills that employers seek. These skills span a wide and varied range, applicable to learners just starting their career journey to those wishing to advance on their path.

The browser-based platform supports all learning styles with a wide array of features, including in-application exercises. These in-application capabilities allow learners to experience real-world examples of specific tasks within a given software environment. The lab includes inclusive learning resources, along with extension activities and project-based learning challenges that foster creativity and critical thinking.

All course content on the Isograd Learning Lab is aligned to a Tosa exam blueprint, which is the foundation of any Tosa certification exam. By aligning course content to an exam blueprint, learners will be prepared to take the exam once they complete the courseware.



1. Identify Outcomes	2. Identify Skill Gaps	3. Learn & Practice	4. Measure Progression	5. Certify Skills
Using a Tosa® exam blueprint, instructors integrate Tosa® certifications into their existing content or curriculum and set define outcomes.	Students begin with a formative pre-course assessment to identify their current skills level. Instructors gather data and adjust lesson plans based on students' results.	Instructors match courseware in the Isograd Learning Lab to student's proficiency levels and intended course objectives.	Once the courseware is completed, students take a summative assessment to verify their skills match the intended proficiency level. If skill gaps still exist, instructors can assign additional course content.	When instructors are satisfied with students' progress, certification exams are given.

Tosa® Web Developer Level Descriptions

At each level, candidates can:

Beginner

- Demonstrate a limited understanding of HTML/CSS/JavaScript, without the capacity to engage with the languages in a professional environment.

Basic

- Demonstrate basic knowledge of HTML, CSS, and JavaScript.
- Modify simple code (text, color, image) and understand the structure of a web page.
- Demonstrate knowledge of the basics of HTML tags, CSS styling, and very simple scripts, but may not yet be able to build a complete website independently.

Productive

- Create a simple, functional website using fundamental web development concepts and techniques.
- Demonstrate a solid command of HTML, CSS, and JavaScript basics.
- Independently develop moderately complex websites and identify appropriate solutions based on user needs.

Advanced

- Create complex websites using advanced web development concepts and techniques.
- Demonstrate mastery in integrating sophisticated features and managing user interactions smoothly and responsively.
- Solve complex technical problems, handle real-time data, and work with external APIs to enhance web functionality.

Expert

- Demonstrate nuanced expertise in web development, with an in-depth understanding of web development concepts and techniques and their application in real-world contexts.
- Design and develop innovative, high-quality web solutions.

Business Applications

Achievement of Beginner score defines little or limited knowledge of HTML/CSS/Java Script, including the languages' basic functions and features, highlighting the candidate's inability to perform web development in a professional environment.	*Entry-level web content editor, junior website assistant, administrative support staff, or any position requiring the use of HTML, CSS, and basic JavaScript to update website content, apply consistent styling, or perform simple user interface tasks such as form validation or basic interactivity.*	*Web production assistant, digital project coordinator, junior front-end developer, IT support technician, or any role requiring regular use of HTML, CSS, and JavaScript to build responsive webpages, update site components, and develop reusable code snippets that streamline web-related tasks in a business environment.*	*Front-end developer, digital project manager, UX/UI specialist, web consultant, or any position requiring advanced use of HTML, CSS, and JavaScript to build interactive dashboards, implement data-driven interfaces, and create scalable, modular code to support strategic web development initiatives.*	*Senior front-end engineer, senior web strategist, chief technology officer, digital experience consultant, or any role requiring expert-level proficiency in HTML, CSS, and JavaScript for architecting scalable web solutions, developing advanced user interfaces, and supporting executive decision-making through sophisticated interaction design, performance optimization, and automation.*

Domain 1: Fundamentals and Applications

Sub-domain 1: Getting started with Web Development

Includes identifying the roles of HTML, CSS, and JavaScript, and understanding key web concepts such as URLs, HTTP/HTTPS, APIs, and basic web infrastructure.

Beginner

- Identifying the roles and basic functions of core web development languages (HTML, CSS, JavaScript).
- Understanding how HTML structures web pages and how CSS controls visual styling and layout.
- Recognizing key web acronyms and concepts such as HTML, CSS, URL, and HTTPS.
- Understanding the purpose of APIs and how websites use them to exchange data.

Basic

- Identifying how HTML, CSS, and JavaScript are used to create structure, styling, and interactivity in web pages.
- Understanding key web technologies and protocols, including HTTP, HTTPS, URLs, and basic web infrastructure.
- Recognizing different types of web development (front-end, back-end, full-stack) and how web technologies are classified.
- Understanding how APIs enable websites and web applications to retrieve and exchange data with external services.

Productive

- Identifying the Uses of the Web Development Languages
- Evaluating API Usage Scenarios in Web Interactions
- Understanding Web Development

Advanced

- N/A.

Expert

- N/A

Domain 2: HTML

Sub-domain 1: HTML tags, attributes, and structure

Covers: Knowledge and application of HTML tags and attributes, including their general purpose, optimization, and relationship to structure. SEO strategies and their relation to HTML within web development.

Beginner

- Identify basic HTML tags without necessarily using them correctly (e.g., <html>, <head>, <body>, <p>, <h1> to <h6>).
- Understand the general purpose of tags and the concept of a web page structure, but may not yet be able to produce structured or standards-compliant code.
- Visually detect the hierarchy of a simple web page (headings, paragraphs, lists) but may not yet have the technical mastery to reproduce it.

Basic

- Use basic HTML tags to structure simple content (headings, paragraphs, lists, links, images).
- Apply essential HTML attributes (id, class, src, alt, href) to enrich content and begin creating a structure readable by browsers.
- Begin to follow a logical content hierarchy and consider accessibility, though without consistently validating against standards.

Productive

- Structure a simple web page using basic HTML tags correctly (<html>, <head>, <body>, <p>, <h1> to <h6>, <a>, <img>, <ul>, <ol>, <li>) to produce documents that are readable by browsers and follow an appropriate visual hierarchy.
- Add essential HTML attributes (id, class, src, alt, href) to tags to enrich content and ensure proper interaction with screen readers and search engines.
- Validate a simple HTML document using automated tools to ensure syntactic compliance with web standards.

Advanced

- Organize a logical and coherent structure using semantic HTML tags (<header>, <nav>, <article>, <section>, <footer>, <aside>, <main>) to improve the accessibility and search engine optimization (SEO) of web pages.
- Design and integrate an HTML form using appropriate tags (<form>, <input>, <label>, <textarea>, <select>, <option>) and their attributes (name, type, required, placeholder) to enable user data entry and submission within web pages.
- Implement accessibility standards (WCAG 2.1 Level AA) by using ARIA attributes (e.g., role, aria-label) to ensure effective interaction with web content for users with disabilities.

Expert

- Optimize the semantic structure of a complex HTML document by incorporating technical SEO strategies to maximize visibility and deliver a smooth user experience.
- Integrate microdata directly into HTML tags to enhance search engine results and improve the relevance of web pages.
- Structure an HTML document following Progressive Web App (PWA) principles, including necessary tags (e.g., <link> for manifests, <meta> for responsive views) to ensure offline capability and mobile compatibility.

Sub-domain 2: Using HTML5 elements and features

Covers: Using HTML5 elements and features such as forms, tables, images, videos, and animations to create richer, more interactive, and dynamic web pages.

Beginner

- Explore the main types of HTML5 content (images, tables, forms, videos) through observation of simple examples.
- Understand the role of these elements in a web page to enrich content (illustrate, organize, collect information, display media).
- Can recognize, in code or on a web page, the tags related to images, tables, videos, and forms.

Basic

- Build a very simple table using the <table>, <tr>, and <td> tags to visually structure basic data, without advanced handling of titles or headers.
- Create a basic form with text fields (<input type="text">) and a submit button (<button>), without error handling or advanced features.

Productive

- Insert and configure an image or video in a web page using the HTML5 <img> and <video> tags and their attributes to present basic visual and multimedia content.
- Build and format a simple table using the <table>, <thead>, <tbody>, <tr>, <td>, and <th> tags to organize and display tabular data.
- Create a basic form using HTML5 tags <form>, <input>, <textarea>, <button> and their attributes to collect user input.

Advanced

- Integrate a video or interactive animation using the <video> and <canvas> tags along with their associated APIs to enhance the user experience with dynamic content.
- Structure complex tables using additional tags (<caption>, <colgroup>, <col>) and ensure accessibility through ARIA attributes and tools like scope to make data understandable for all users.
- Design an advanced form with specific input types (date, color, range, email, number) to make user interactions more secure and user-friendly.

Expert

- Optimize multimedia content integration using advanced HTML5 tags (<picture> for responsive images, <source> for adaptive videos), while considering performance and accessibility to ensure compatibility across different devices and browsers.
- Create and manipulate complex interactive animations using the HTML5 <canvas> element combined with JavaScript APIs to deliver high-performance experiences.
- Configure dynamic, responsive forms that incorporate autocomplete features, complex validations, and customized error messages to enhance user experience and ensure reliable data collection.

Sub-domain 3: Adding interactive and dynamic features to a web page

Covers: Using DOM events, JavaScript scripts, and web APIs to create more immersive and engaging user experiences.

Beginner

- Discover the role of JavaScript in a web page for enabling simple interactions.
- Understand the concept of events (click, hover) and their purpose in controlling page behavior, though may not yet be able to implement them.
- Identify the presence of a very simple JavaScript script in code.
- Demonstrate initial understanding of the logic of the DOM (the hierarchical structure of elements on a page).

Basic

- Use a small, provided JavaScript script to make simple changes to page content.
- Trigger a basic action in response to a user click using HTML attributes.
- Begin to understand how JavaScript interacts with HTML elements through the DOM, but only at a very basic level.

Productive

- Manipulate HTML elements using basic DOM methods in JavaScript to dynamically modify the content of a web page.
- Add interactions to a web page using DOM events (onclick, onmouseover, onchange) to respond to user actions.

Advanced

- Manipulate CSS classes and styles through JavaScript (e.g., classList, style) to dynamically change the appearance of elements in response to user actions.
- Handle complex DOM events using event handlers (e.g., addEventListener, event delegation) to centralize and optimize user interactions across multiple elements.
- Use standard web APIs to load and store data dynamically without reloading the page.

Expert

- Manipulate CSS classes and styles using JavaScript (e.g., classList, style) to dynamically change the appearance of elements in response to user actions.
- Manage complex DOM events using event handlers (e.g., addEventListener, event delegation) to centralize and optimize user interactions across multiple elements.
- Leverage standard web APIs to dynamically load and store data without reloading the page.

Domain 3: CSS

Sub-domain 1: Designing page layout

Covers: Using CSS to design layouts that center the user experience, improving readability and visual harmony with purposeful design choices in web development.

Beginner

- Discover the purpose of CSS in separating content from presentation on a web page.
- Understand the role of CSS rules in changing the appearance of a page (colors, size, margins).
- Apply very simple styles to a single HTML element.
- Recognize common CSS properties like color, background-color, and font-size, but may not yet be able to organize a page layout.

Basic

- Use basic CSS properties to position and space elements in a simple layout.
- Organize a basic page by placing elements vertically or side by side using display: block or inline-block.
- Apply basic styles to improve readability.
- Begin to demonstrate understanding of the value of the CSS box model, without yet mastering advanced layout systems like Flexbox or Grid.

Productive

- Lay out a simple web page using flexible boxes (CSS Flexbox) with properties such as display: flex, justify-content, and align-items to organize elements effectively.
- Apply colors, fonts, and background styles using CSS to create a visually harmonious interface.

Advanced

- Create a complex layout using CSS Grid with properties such as grid-template-rows, grid-template-columns, and gap to precisely structure content areas.
- Implement a responsive layout using media queries (@media) to ensure an optimal user experience across different devices (mobile, tablet, desktop).
- Customize typography and formatting styles (line-height, text-align, letter-spacing) to enhance content readability and adhere to design standards.

Expert

- Design advanced, dynamic layouts by combining CSS Grid and Flexbox to manage flexible, modular structures tailored to specific user needs.
- Incorporate a systematic design methodology into layouts to ensure visual and functional consistency across complex projects.
- Optimize web page performance using advanced CSS techniques (contain, will-change, targeted use of pseudo-elements like ::before and ::after) to reduce loading times.

Sub-domain 2: Styling web elements

Covers: Defining the style of a web element to create a consistent visual identity for a website or application.

Beginner

- Apply simple formatting styles (e.g., solid background-color) to distinguish an element.
- Discover the use of basic pseudo-classes such as :hover to slightly modify a style when the mouse hovers over an element.

Basic

- Combine multiple CSS text properties (e.g., font-family, line-height) to improve readability.
- Add simple borders (border) and adjust margins to help structure the layout.
- Use pseudo-classes such as :focus to enhance accessibility and provide user feedback.

Productive

- Apply basic CSS properties such as color, font-family, and font-size to style text in accordance with basic visual guidelines.
- Customize the appearance of an element using properties like background-color, border, and box-shadow to visually highlight it.
- Use pseudo-classes (:hover, :focus) to add simple interactive styles that enhance the user experience.

Advanced

- Create advanced visual effects using gradients (linear-gradient, radial-gradient) and complex shadows (box-shadow, text-shadow) to add depth and dynamism to the design.
- Design custom buttons and interactive elements by combining pseudo-classes and pseudo-elements (::before, ::after) to strengthen visual identity and user interaction.
- Structure a global CSS theme using variables (e.g., --primary-color) to ensure visual consistency across all pages of a website or application.

Expert

- Define and apply a complete design system using advanced techniques to ensure a consistent visual identity.
- Create complex CSS animations (@keyframes, transition, animation) to deliver interactions that align with the project's visual identity.
- Optimize styling and visual performance by applying advanced techniques such as structural pseudo-classes (:nth-child) and minimizing CSS file size to ensure a fast and accessible user experience.

Sub-domain 3: Styling for differing devices

Covers: Adapting the style of a web element for different devices to ensure users have an optimal experience regardless of the device they are using.

Beginner

- Discover the differences between fixed units (px) and relative units, and primarily use fixed units to define size and spacing.
- Observe display differences across various devices without adapting the styles.
- Use developer tools to preview pages at different resolutions without modifying the styles.

Basic

- Begin using simple relative units (%, em) to adjust basic sizes (text, margins).
- Set up basic media queries to modify simple properties (font size, margins) across major screen types (mobile, tablet, desktop).
- Perform display testing on standard resolutions to identify major responsiveness issues.

Productive

- Use relative units (em, rem, %) instead of fixed units to ensure better adaptability of sizes and spacing across different screen sizes.
- Apply basic media queries to adjust core styles (fonts, margins, widths) for various screen types.
- Test how a web page displays on common resolutions (mobile, tablet, desktop) using developer tools to identify and correct potential layout issues.

Advanced

- Create fluid layouts by combining relative units (%, vh, vw) with modern CSS properties such as min(), max(), and clamp() to ensure optimal adaptability across a wide range of screens.
- Integrate custom breakpoints into media queries to apply styles tailored to specific devices.
- Optimize visual hierarchy by adjusting sizes, colors, and spacing based on resolution to maintain a consistent and harmonious user experience across all devices.

Expert

- Design a responsive user interface using advanced techniques such as CSS container queries to enable precise contextual adaptations, independent of overall screen resolution.
- Optimize the performance of adaptive styles by minimizing redundancy in media queries and using tools like PostCSS or CSS preprocessors (e.g., SASS) to efficiently manage complex projects.

Domain 4: JavaScript

Sub-domain 1: Defining DOM events

Covers: The progressive mastery of DOM events and JavaScript-driven interactivity. Recognizing basic user events to manipulate elements and styles in response, and building complex, responsive interface components using custom events and web APIs.

Beginner

- Understand the concept of user events (click, hover) and their role in making a web page interactive.
- Use simple JavaScript functions without event handlers to occasionally modify HTML content.
- Observe how an HTML element behaves in response to user actions without yet responding through code.

Basic

- Attach a simple event handler to an element to trigger a function in response to basic user actions (click, hover).
- Select one or more HTML elements using basic DOM selectors (getElementById, querySelector) for manipulation.
- Directly modify CSS properties such as display or visibility using JavaScript to show or hide content in response to an event.

Productive

- Use a simple JavaScript event handler to respond to simple user actions such as clicks, hovers, or form submissions in order to add interactive functionality to a web page.
- Access and manipulate HTML elements using DOM selectors to modify them in response to user events.
- Implement content show/hide behavior using CSS properties (display, visibility) triggered by DOM events.

Advanced

- Handle complex events (mouseover, keydown, scroll) to enhance the user experience.
- Dynamically manipulate CSS classes (classList.add, classList.toggle) via DOM events to modify styles and element behavior in real time.
- Create advanced interactions such as dropdown menus, modals, and carousels by combining DOM events with JavaScript loops and conditionals.

Expert

- Create and trigger a custom event using CustomEvent to meet specific needs within a web application.
- Integrate a JavaScript API to build advanced interactions based on user context, such as lazy loading elements or triggering animations on scroll.

Sub-domain 2: Building dynamic visuals and connecting to external data

Covers: Utilizing animations, transitions, and data integration. Recognizing visual effects and making basic style changes to using JavaScript and CSS to create smooth, interactive animations and retrieve external data via APIs.

Beginner

- Understand what a visual animation or transition is and observe them on existing web pages, without manipulating them.
- Manually modify simple CSS properties (e.g., color, background-color) using JavaScript, without applying transitions or animations.
- Know the basics of how HTTP requests work, but may not yet use APIs to retrieve data.

Basic

- Trigger CSS style changes via JavaScript, preparing elements to receive CSS transitions.
- Use JavaScript to modify CSS properties such as transform or opacity, without yet coordinating or creating smooth animations.
- Perform basic HTTP requests using the Fetch API (without error handling or advanced processing), and display raw results on the page.

Productive

- Use CSS transitions via JavaScript to add smooth effects during user interactions, enhancing the visual appeal of web pages.
- Implement simple animations by combining CSS properties with JavaScript (e.g., element.style.transform or element.style.opacity) to make elements interactive.
- Use the Fetch API to send basic HTTP GET requests and display the retrieved data on a web page in order to integrate external information into a site.

Advanced

- Create a complex animation by combining JavaScript and CSS (e.g., synchronizing @keyframes animations with DOM events) to produce advanced and dynamic visual effects.
- Manage transitions and animations asynchronously with JavaScript to coordinate multiple actions and interactions.
- Handle JSON responses to develop connected, data-driven functionality.

Expert

- Implement complex, high-performance animations using the Web Animations API (WAAPI) or specialized JavaScript libraries to create an immersive and personalized user experience.
- Optimize the performance of animations and transitions (e.g., by avoiding costly properties like width or height, and using the will-change property) to ensure fast and smooth web applications.

Domain 5: Integration

Sub-domain 1: Integrating components and features

Covers: Building complete web interfaces, from assembling basic components and navigation to using JavaScript frameworks and APIs for dynamic, optimized applications.

Beginner

- Understand the basic structure of a web page (forms, images, sections, tables) without yet assembling these components into a coherent whole.
- Insert simple HTML elements on a page without managing their organization or navigation.
- Use static buttons and forms without handling data submission or related interactions.

Basic

- Insert and position basic HTML elements (text, images, buttons) in isolation, without yet structuring the overall page layout.
- Set up simple links between a few pages or sections, without advanced navigation handling (e.g., basic hyperlinks).
- Use static forms without managing submission or data processing, primarily to collect simple information.

Productive

- Assemble HTML components such as forms, tables, images, and sections to integrate various elements of a web page in a coherent way.
- Link pages and sections of a web application using internal and external links, enabling functional navigation between pages and within a single page.
- Integrate simple interactive elements like forms and buttons, and handle data submission through basic requests or redirections.

Advanced

- Organize the integration of dynamic components using JavaScript libraries or frameworks to manage advanced interactive elements and incorporate real-time or conditional data.
- Handle form behavior, including validation and data submission to a server via AJAX, to create interactive and responsive interfaces that offer a smooth user experience.
- Structure the application with reusable, dynamic components to maintain a consistent and scalable development logic.

Expert

- Develop a modular and flexible architecture by integrating dynamic components and specific features through a JavaScript framework to build a web application.
- Seamlessly integrate an API using external services (e.g., payment API, geolocation API, mapping API) to enhance the user experience.
- Optimize integration performance by ensuring fast load times for an optimal user experience.

Sub-domain 2: Integrating APIs

Covers: Integrating APIs into web applications, from making basic data requests and displaying simple results to handling authentication, processing complex responses, and building advanced service architectures with multiple or specialized APIs.

Beginner

- Demonstrate a basic understanding of service architecture.
- Identify the role of a public API.

Basic

- Perform simple HTTP requests to a public API to retrieve data.
- Display the retrieved data in a basic user interface format, such as a list or table.
- Test the retrieval and display of simple information (e.g., products, articles) without advanced error handling or complex interactions with the API.

Productive

- Integrate a simple public API and display the data in a user interface as a list or table.
- Use an external API to show basic information, such as products or articles, and test manipulating the data within the page.

Advanced

- Implement an authentication mechanism via a secure API to access a protected service, then display the retrieved information within an application.
- Perform dynamic API calls by sending data to the API and handling complex responses such as JSON objects or errors.

Expert

- Create a service architecture that integrates multiple external APIs to power an application, ensuring the management of simultaneous calls and optimized performance.
- Integrate a complex, specialized API—such as a payment gateway or artificial intelligence service—to enhance the application with advanced features while maintaining security and overall performance.

Sub-domain 3: Integrating external components

Covers: Integrating external components from basic visuals to advanced libraries and services to enhance functionality, interactivity, and scalability in web applications.

Beginner

- Understand the role of external components and observe their use on existing websites or applications.
- Manually insert simple visual elements (images, videos) from external sources without specific technical integration.
- Use basic icon libraries by simply copying external files or links into a web page.

Basic

- Import and use icon libraries via CDN links or local files, integrating icons into web pages in a simple way.
- Add basic third-party widgets (e.g., social buttons, subscription forms) without advanced customization to enhance interactivity.

Productive

- Integrate simple external components such as icon libraries or third-party widgets to add basic features like maps or icons in an application, enhancing interactivity and visual appeal.
- Use external user interface (UI) components (e.g., navigation bars, buttons, or alerts) from libraries to make the application more functional and professional.
- Embed external visual elements like images or videos from outside sources to enrich the user experience without technical complexity.

Advanced

- Integrate UI libraries and frameworks (e.g., React Bootstrap, Tailwind CSS) to build dynamic, customizable components that support responsive interfaces for complex websites or applications.
- Use external JavaScript plugins (e.g., for image carousels, modals, or interactive charts) to add advanced features.
- Handle the integration of complex external components such as payment solutions or authentication services to securely manage transactions or user logins.

Expert

- Create a modular architecture that allows seamless integration of multiple external components (frameworks, APIs, widgets) to build a scalable and maintainable application.
- Integrate advanced interactive external components such as live chat systems, custom search engines, or real-time analytics tools to enrich application functionality and deliver personalized, engaging user experiences.

TOSA

Your skills. Your advantage.

contact@isograd.com

www.tosa.org