

Study Guide

Mobile Web Specialist Certification

Use the *Study Guide* to prepare for the Mobile Web Specialist Certification exam. The *Guide* lists the competency areas and individual competencies against which you will be tested. There are also links to suggested web-based study resources. Note that these resources form only a small portion of what is available on the web, and we encourage you to do additional research.

Contents

- [Basic Website Layout and Styling](#)
- [Front End Networking](#)
- [Accessibility](#)
- [Progressive Web Apps](#)
- [Progressive Enhancement](#)
- [Performance Optimization and Caching](#)
- [Security](#)
- [Testing and Debugging](#)
- [JavaScript Design Principles](#)
- [ES2015 Concepts and Syntax](#)
- [Mobile Web Forms](#)
- [Front End JavaScript](#)

Basic Website Layout and Styling

Users expect responsive and visually engaging websites regardless of the device. A web application's layout and styling must respond to the current display, while continuing to provide intuitive functionality. You'll be asked to show you can use HTML and CSS to build a web application's responsive layout and style that includes:

- Appropriate document type declaration and viewport tags
- A responsive grid-based layout using CSS
- Media queries that provide fluid breakpoints across different screen sizes
- Multimedia tags to display video or play audio
- Responsive images that adjust for the dimensions and resolution of any mobile device

Resources:

- [Responsive Web Design](#)
- [A Complete Guide to Flexbox](#)
- [Using media queries](#)
- [Video and audio content](#)

Front End Networking

Because user engagement depends on reliable and effective network requests, you'll be asked to show you can use JavaScript to set up reliable front end networking protocols by:

- Requesting data using `fetch()`
- Checking response status, then parsing the data into usable format
- Rendering response data to a page
- Configuring POST requests to a database with `method` and `body` parameters
- Using correctly configured cross-origin resource sharing protocol (CORS) `fetch` requests, depending on the server's response headers
- Handling `fetch()` request errors with promise chaining
- Diagnosing network issues using debugging and development tools

Resources:

- [Using fetch](#)
- 


- [Introduction to fetch\(\)](#)
- [David Walsh's blog on fetch](#)
- [Jake Archibald's blog on fetch](#)
- [JavaScript Promises: an Introduction](#)
- [HTTP access control \(CORS\)](#)

Accessibility

Web pages and applications should be accessible to all users, including those with visual, motor, hearing, and cognitive impairments. Using HTML, CSS, JavaScript, you'll be asked to show you can integrate accessibility best practices into your web pages and applications by:

- Using a logical tab order for tabbed navigation
- Using skip navigation links to bypass navbars and asides
- Avoiding hidden content on the page that impedes tab navigation
- Using heading tags that provide a logical page structure
- Using text alternatives to visual content, such as `alt`, `<label>`, `aria-label`, and `aria-labelledby`
- Applying color contrast to all elements and following accessibility best practices
- Sending timely alerts for urgent messages using `aria-live`
- Using custom-built components that integrate keyboard shortcuts and convey their role, name, state, and current value to all users
- Using semantic markup to keep content and presentation separate when appropriate

Resources:

- [Web Fundamentals – Accessibility](#)
 - [Mobile Accessibility](#)
 - [Using tabindex](#)
 - [Focus](#)
 - [Skip Navigation Links](#)
 - [ARIA](#)
- 

Progressive Web Apps

Users expect native applications to be available offline and provide a feature-rich experience that is launchable from their home page. You'll be asked to show that you can use the Service Worker Toolbox, HTML, and JavaScript to build out progressive web application features similar to native applications by:


- Creating a web app that is available offline, and that caches elements by routing requests through a service worker using the Service Worker Toolbox
- Storing the default display orientation, theme color, display icon (add to home screen), and splash screen in the web application manifest (or using meta tags)
- Separating critical application functionality and UI into an application shell that can be loaded independently from the content

Resources:

- [Progressive Web Apps](#)
- [Web Fundamentals - The App Shell Model](#)
- [Your First Progressive Web App](#)
- [Using Service Workers](#)
- [Service Worker Libraries](#)

Progressive Enhancement

Users expect applications to function consistently on their browser regardless of their device's capability. Progressively enhanced applications use layered web technologies to provide the best possible experience across different browsers. You'll be asked to show that you can use semantic HTML, CSS, and JavaScript to progressively enhance the application based on browser capabilities by:

- Loading JavaScript modules progressively to enhance the user experience to the full extent allowed by their browser
 - Taking a "content-first" approach that progressively adds layers to code starting with:
 - Semantic HTML that provides a consistent experience and works in as many browsers as possible
 - CSS that only concerns itself with proper layout structures, and considers CSS as a progressive enhancement
- 

- JavaScript

Resources:

- [Progressive Enhancement](#)
- [Progressive Enhancement for JavaScript App Developers](#)
- [Progressive Enhancement: Start Using CSS Without Breaking Older Browsers](#)

Performance Optimization and Caching

Mobile users demand websites that load nearly instantly, despite poor or absent connectivity. Because many users also face expensive data caps, you must minimize their application's data footprint to reduce page load time as much as possible. You'll be asked to show you can perform performance audits on applications to reduce page load times and maintain responsive user experiences by:

- Preventing main thread blocking with a dedicated web worker
- Providing an optimized critical rendering path using:
 - Compressed or minified JavaScript, HTML and CSS files to reduce render blocking
 - Inline CSS for essential styles on a specific page, with asynchronous loading for additional styles as necessary
 - Inline JavaScript files for initial rendering only where necessary (or otherwise eliminated, deferred, or marked as `async`)
 - Ordered loading of remaining critical resources and early download of all critical assets to shorten the critical path length
 - Reduced DOM depth to minimize browser layout/reflow
- Prefetching files that load when resources are available, reducing the time to meaningful interaction
- Providing client storage that is appropriate to a web application's data persistence needs, including:
 - Session state management
 - Asset caching based on their impact on load time and offline functionality

- Using IndexedDB to store dynamic content in offline mode

Resources:


- [Web Fundamentals - Performance](#)
- [The Offline Cookbook](#)
- [Cache - MDN](#)
- [Storage](#)
- [Local Storage And How To Use It On Websites](#)
- [IndexedDB API](#)

Security

Security is essential in safeguarding server and user data. As a mobile web developer you must be aware of common security exploits and how to combat them. You'll be asked to show you can perform routine security audits and integrate security into your workflow to protect web applications from common security exploits by:

- Preventing cross-site scripting (XSS) using whitelists within content security policy headers, user input validation, HTTPS-only, and ensuring that tainted information can't be served to the DOM
- Restricting cross-site request forgery (CSRF) attacks by validating that request headers come from the same origin, and by using CSRF token validation
- Mitigating man-in-the-middle attacks that compromise user data and website integrity by requesting external resources through HTTPS
- Preventing session hijacking and user account theft by implementing secure logins and session management including:
 - Authenticating users by requiring a user name or ID and at least one item of verifiable information
 - Using secure cookies to maintain authenticated user sessions and prevent insecure transmission

Resources:

- [Cross-site Scripting \(XSS\)](#)
 - [Cross-Site Request Forgery \(CSRF\) Prevention Cheat Sheet](#)
 - [Man-in-the-middle attack](#)
 - [Session Hijacking](#)
 - [Content Security Policy](#)
- 

Testing and Debugging

Developers typically work in highly iterative deployment environments, relying on extensive testing and debugging to maintain functionality and code integrity. You'll be asked to show that you can verify expected behaviors and diagnose common web application bugs by:

- Writing unit tests that first verify a function's intended behavior, and then iteratively modifying its code until it passes those tests
- Setting breakpoints within a complicated function to determine exactly where it deviates from expected behavior
- Using console logs to output relevant debugging information
- Reproducing and fixing bugs based on user reported issues

Resources:


- [Get Started with Debugging JavaScript in Chrome DevTools](#)
- [Diagnose and Log to Console](#)
- [Debugging Service Workers](#)

JavaScript Design Principles

Mobile web developers are expected to create code that is readable, modular, and maintainable in large teams. You'll be asked to show you can write code that conforms to object oriented design principles and JavaScript best practices using:

- Prototypal inheritance to write reusable code that can be passed onto multiple child objects/functions
- Immediately invoked function expressions that encapsulate functions
- Closures that access a variable outside of a function's scope
- "this" values bound to a function (e.g., in an event handler)

Resources:

- [Key Principles of Maintainable JavaScript](#)
 - [Learning JavaScript Design Patterns](#)
 - [Prototypal Inheritance in JavaScript](#)
- 

ES2015 Concepts and Syntax

Web developers must stay current with the latest JavaScript features that promote simpler and more readable code. With polyfills enabling code written in ES2015 JavaScript to be used in unsupported browsers, there is a strong incentive for developers to begin using the new features and syntax. You'll be asked to show that you understand and can write ES2015 JavaScript code using:

- JavaScript promises with ES2015 syntax that create asynchronous functions and incorporate graceful error handling
- Variables that can be used with block scope, function scope, and made immutable depending on context using `let`, `var`, and `const`
- String literals that include string interpolation and multi-line strings
- Arrow functions that create anonymous functions and use an unbounded `this`
- Self-contained modules that run a highly specific, reusable function
- Default function parameters that initialize default values for a function when no argument or `undefined` is provided
- `for...of` loops that can iterate over any iterable object while running a custom function on each
- Maps that allow for arbitrary key and value pairs that are iterable and include non-string keys
- Sets that contain only unique, iterable elements where an array would degrade performance

Resources:

- [ES2015](#)
- [JavaScript Promises: an Introduction](#)
- [Promise](#)
- [Template literals](#)
- [Arrow Functions](#)
- [JavaScript Modules: A Beginner's Guide](#)
- [Default parameters](#)
- [For...of](https://medium.freecodecamp.com/javascript-modules-a-beginner-s-guide-783f7d7a5fcc#ymbhiyn0f)<https://medium.freecodecamp.com/javascript-modules-a-beginner-s-guide-783f7d7a5fcc#ymbhiyn0f>
- [Map](#)

- [Set](#)

HTML Forms

Filling out online forms, especially on mobile devices, can be difficult. To improve the user experience you'll be asked to show that you can use basic HTML5, JavaScript, and the HTML5 Constraint Validation API, to design efficient and secure HTML web forms with:


- Appropriate `label` tags associated with inputs
- Inputs with appropriate `type`, `name` and `autocomplete` attributes
- Inputs with large touch targets for mobile forms
- Suggestions for user input using the `datalist` element
- Front-end validation of inputs (e.g., `pattern`, `maxlength`, `required`) and DOM elements, including:
 - Checking validation errors in real-time with pseudo-classes on inputs
 - Form validation prior to submission (Constraint Validation API)

Resources:

- [HTML Forms](#)
- [Constraint Validation](#)
- [Client-Side Form Validation with HTML5](#)
- [Data form validation](#)
- [Create Amazing Forms](#)

Front End JavaScript

Users expect web applications to be fully featured with rich content regardless of the device. Web applications must respond to mobile devices with the speed and user experience of native applications. You'll be asked to show that you can use JavaScript, without the overhead of loading a library (such as jQuery), to create front-end functionality that works across all devices and platforms using:

- DOM elements that are accessed and manipulated dynamically
 - Touch and mouse events that contain large hit targets on the front end and work regardless of platform
- 



- The HTML5 History API to provide a logical browser history in a single-page application

Resources:

- [Supporting both TouchEvent and MouseEvent](#)
- [Touch events](#)
- [Using the HTML5 History API](#)
- [Manipulating the browser history](#)