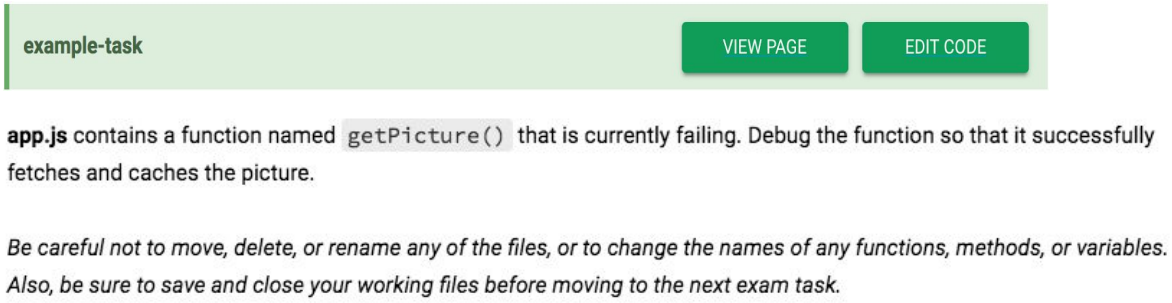
Study Guide
Mobile Web Specialist Certification

Use the *Study Guide* to prepare for the Google 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

Sample Question

Questions in the Mobile Web Specialist Certification exam are presented in the form of coding challenges based on the competency areas covered in this *Guide*. The following example shows you what you might expect to see in the certification exam:



To respond to this challenge, you would click **VIEW PAGE** to open the corresponding web page in Chrome. After determining what needs to be fixed, you would then click **EDIT CODE** to open the code editor (Atom) and debug **app.js** so that `getPicture()` works when you call it from the web page.

You will be asked to resolve a number of similar challenges within a four-hour exam, after which you will take a ten-minute exit interview. You must pass both the exam and exit interview to receive your certification.

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, CSS, and JavaScript to build a web application's responsive layout and style that includes:

- DOM elements that are accessed and manipulated using only JavaScript without the overhead of libraries or frameworks (such as jQuery)
- 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

- Touch and mouse events that contain large hit targets on the front end and work regardless of platform

Resources:

- Responsive Web Design
- A Complete Guide to Flexbox
- Using media queries
- Video and audio content
- Responsive Images by Google
- Supporting both TouchEvent and MouseEvent
- Touch events

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 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 service workers, and 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

- 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
- Progressive Web Apps Training
- Web Fundamentals - The App Shell Model
- Your First Progressive Web App
- Using Service Workers

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 carry out 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
    - Your browser's developer tools to diagnose performance issues on mobile devices

- 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:

- Offline Web Applications by Google
- Web Fundamentals - Performance
- The Offline Cookbook
- Cache - MDN
- Storage
- Local Storage And How To Use It On Websites
- IndexedDB API
- Get Started with Analyzing Network Performance in Chrome DevTools

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

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`
- 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:

- JavaScript Promises: an Introduction
- Promise
- Template literals
- Arrow Functions
- Default parameters
- For...of
- Map
- Set

Mobile Web 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