Frontend-Entwickler-Interview

Mit HTML beginnen:

1. Semantische Tags: `<article>`, `<section>`, `<header>`, `<footer>`, `<nav>` verstehen und verwenden.
2. Formulare: Validierung implementieren, `<input>`, `<textarea>`, `<select>`, `<button>` verarbeiten.
3. Tabellen: Mit `<table>`, `<thead>`, `<tbody>`, `<tfoot>` zugängliche Tabellen erstellen.
4. Metadaten: `<meta>`-Tags für Zeichensatz, Anzeigebereich und SEO verwenden.
5. Links und Anker: `<a>`-Tags, `href`, `target` und `download`-Attribute verstehen.
6. Medienelemente: `<img>`, `<video>`, `<audio>` korrekt mit Attributen wie `src`, `alt`, `controls` verwenden.
7. Listen: Geordnete `<ol>` und ungeordnete `<ul>`-Listen erstellen, einschließlich verschachtelter Listen.
8. Überschriften: Richtige Überschriftenhierarchie `<h1>` bis `<h6>` verwenden.
9. Einbetten von Inhalten: `<iframe>`, `<embed>` und `<object>` zum Einbetten externer Inhalte verwenden.
10. HTML5-APIs: Kenntnisse über Geolocation, Web Storage und Fetch API.

Nun, CSS:

11. Box-Modell: Rand, Einrückung, Rahmen und deren Auswirkungen auf das Layout verstehen.
12. Flexbox: Ausrichtung, Umbruch und Reihenfolge mit Flexbox-Eigenschaften meistern.
13. Grid-Layout: Komplexe Layouts mit CSS Grid erstellen.
14. Responsive Design: Medienabfragen, Viewport-Meta-Tag und responsive Bilder verwenden.
15. CSS-Preprozessoren: Kenntnisse über Sass-, Less- oder Stylus-Syntax und -Funktionen.
16. CSS-in-JS: Frameworks wie styled-components oder emotion verstehen.
17. Animationen und Übergänge: Gleitende Übergänge und Keyframe-Animationen implementieren.
18. Styling von Formularen: Formularelemente anpassen und deren Erscheinungsbild verbessern.
19. CSS-Reset und Normalize: Wann und warum sie verwendet werden, wissen.
20. CSS Grid vs. Flexbox: Unterschiede verstehen und das richtige Werkzeug für den Job auswählen.

JavaScript:

21. ES6+-Funktionen: Pfeilfunktionen, Destrukturierung, Spread/Rest-Operatoren und Vorlageliterale verwenden.
22. DOM-Manipulation: Elemente auswählen, DOM ändern und Ereignisse verarbeiten.

23. Asynchrones JavaScript: Versprechen, async/await und Fetch-API verstehen.
24. Ereignisschleife: Erklären, wie die Ereignisschleife in JavaScript funktioniert.
25. Closures: Closures verstehen und effektiv verwenden.
26. Prototypische Vererbung: Erklären, wie prototypische Vererbung in JavaScript funktioniert.
27. Module: ES6-Module mit `import` und `export` verwenden.
28. Fehlerbehandlung: `try/catch`-Blöcke verwenden und unbehandelte Promise-Ablehnungen verstehen.
29. JavaScript-Leistung: Code für bessere Leistung optimieren.
30. Browser-Konsole: Entwicklertools des Browsers zum Debuggen verwenden.

Frameworks:

31. React.js: Komponenten, JSX, Zustand, Eigenschaften und Hooks verstehen.
32. Vue.js: Vue-Instanz, Direktiven, Komponenten und Reaktivität verstehen.
33. Angular: Komponenten, Dienste, Abhängigkeitsinjektion und Routing verstehen.
34. Zustandsverwaltung: Redux, Vuex oder Context API zur Zustandsverwaltung verwenden.
35. Routing: Clientseitiges Routing mit React Router, Vue Router usw. implementieren.
36. Komponentenbasierte Architektur: Wiederverwendbare Komponenten verstehen und implementieren.
37. Lebenszyklusmethoden: React-Lebenszyklusmethoden oder Vue-Hooks kennen.
38. UI-Bibliotheken: Bibliotheken wie Bootstrap, Tailwind oder Material-UI verwenden.
39. Test-Frameworks: Tests mit Jest, Jasmine oder Cypress schreiben.
40. Build-Tools: Webpack, Babel oder Parcel zum Erstellen von Projekten verwenden.

Tools und Versionskontrolle:

41. Git: Git für die Versionskontrolle verwenden, einschließlich Verzweigung, Zusammenführen und Zurücksetzen.
42. npm/yarn: Projektabhängigkeiten und -skripte verwalten.
43. package.json: Skripte, Abhängigkeiten und Entwicklungsabhängigkeiten verstehen.
44. Task-Runner: Gulp oder Grunt zum Automatisieren von Aufgaben verwenden.
45. Linting: ESLint oder Prettier zur Codequalität verwenden.
46. Browsersync: Für Live-Reloading während der Entwicklung verwenden.

47. Figma/Adobe XD: Design-Übergabe verstehen und mit Designern zusammenarbeiten.
48. API-Integration: Daten von RESTful- oder GraphQL-APIs abrufen.
49. Umgebungsvariablen: Umgebungspezifische Konfigurationen verwalten.
50. Continuous Integration: CI/CD-Pipelines mit GitHub Actions oder Jenkins einrichten.

Leistungsoptimierung:

51. Code-Splitting: Code-Splitting mit Webpack oder dynamischen Imports implementieren.
52. Lazy Loading: Bilder, Komponenten und Skripte lazy laden.
53. Minifizierung: CSS-, JavaScript- und HTML-Dateien minifizieren.
54. Caching-Strategien: HTTP-Caching-Header und Service Worker verwenden.
55. Bildoptimierung: Bilder für die Webnutzung komprimieren und optimieren.
56. Kritisches CSS: Kritisches CSS für schnellere Seitenladezeiten einbetten.
57. Web-Leistungsmetriken: Lighthouse, GTmetrix und PageSpeed Insights verstehen.
58. Schriftart-Laden: Schriftart-Laden mit WebFont Loader oder Selbsthosting optimieren.
59. Vermeidung von renderblockierenden Ressourcen: Sicherstellen, dass Skripte und Stile das Rendern nicht blockieren.
60. Leistungsbudgets: Leistungsbudgets festlegen und einhalten.

Barrierefreiheit:

61. ARIA-Rollen: ARIA-Rollen, -Zustände und -Eigenschaften für bessere Barrierefreiheit verwenden.
62. Semantisches HTML: Semantische Elemente auswählen, um die Barrierefreiheit zu verbessern.
63. Alt-Text für Bilder: Sinnvolle Alt-Texte für Bilder bereitstellen.
64. Tastaturnavigation: Sicherstellen, dass die Website nur mit der Tastatur navigierbar ist.
65. Farbkontrast: Tools verwenden, um den Farbkontrast zu überprüfen und zu verbessern.
66. Screen-Reader-Tests: Mit Screen Readern wie NVDA oder VoiceOver testen.
67. Fokusverwaltung: Sicherstellen, dass die Fokusverwaltung auf interaktiven Elementen korrekt ist.
68. Barrierefreiheitsrichtlinien: WCAG 2.1-Richtlinien befolgen.
69. Formular-Barrierefreiheit: Labels, Platzhalter und Validierung korrekt verwenden.
70. EPub- und AODA-Konformität: Grundlegende Konformitätsstandards verstehen.

Best Practices:

71. Code-Organisation: Saubere und modulare Code-Strukturen beibehalten.
72. Dokumentation: Klares Dokumentieren von Komponenten und APIs.
73. Cross-Browser-Tests: Auf mehreren Browsern und Geräten testen.
74. Progressive Enhancement: Websites erstellen, die für alle Benutzer funktionieren, unabhängig von der Browserunterstützung.
75. Sicherheit: XSS-Angriffe verhindern, Content Security Policy verwenden und APIs sichern.
76. SEO-Best Practices: Mit Meta-Tags, Überschriften und Alt-Text für Suchmaschinen optimieren.
77. Versionierung: Semantische Versionierung für Bibliotheken und Abhängigkeiten verwenden.
78. Kollaborationstools: GitHub, GitLab oder Bitbucket für die Teamarbeit verwenden.
79. Code-Reviews: An Code-Reviews teilnehmen und konstruktives Feedback geben.
80. Lernressourcen: Mit MDN, Blogs und Online-Kursen auf dem Laufenden bleiben.

Fortgeschrittene Themen:

81. WebSockets: Echtzeitkommunikation mit WebSockets implementieren.
82. PWA (Progressive Web Apps): Service Worker, Offline-Unterstützung und Push-Benachrichtigungen verstehen.
83. Canvas und SVG: Grafiken mit Canvas- und SVG-Elementen erstellen.
84. CSS Grid- und Flexbox-Layouts: Komplexe Layouts mit CSS Grid und Flexbox implementieren.
85. Benutzerdefinierte Elemente: Benutzerdefinierte HTML-Elemente mit Web Components erstellen.
86. Shadow DOM: Shadow DOM für die Kapselung verstehen und verwenden.
87. CSS-Variablen: Benutzerdefinierte Eigenschaften für Themen und dynamische Stile verwenden.
88. JavaScript-Designmuster: Designmuster wie Singleton, Observer und Factory implementieren.
89. Internationalisierung (i18n): Sprachunterstützung und Lokalisierung implementieren.
90. Leistungsprofiling: Tools wie Chrome DevTools zum Profiling von JavaScript- und DOM-Leistung verwenden.

Fachübergreifende Fähigkeiten:

91. Benutzererfahrung (UX): UX-Prinzipien verstehen und mit UX-Designern zusammenarbeiten.
92. Benutzeroberfläche (UI): Visuell ansprechende und benutzerfreundliche Oberflächen erstellen.

93. Projektmanagement: Agile Methoden, Scrum oder Kanban für das Projektmanagement verwenden.
94. Kommunikationsfähigkeiten: Effektiv mit Teammitgliedern und Stakeholdern kommunizieren.
95. Problemlösungsfähigkeiten: Probleme methodisch angehen und optimale Lösungen finden.
96. Anpassungsfähigkeit: Schnell neue Technologien und Tools lernen und sich anpassen.
97. Teamarbeit: Gut im Team arbeiten, Wissen teilen und andere anleiten.
98. Zeitmanagement: Aufgaben priorisieren und Zeit effektiv verwalten.
99. Kreativität: Kreative Lösungen für Design- und Codierungsherausforderungen finden.
100. Leidenschaft fürs Lernen: Neugierig bleiben und kontinuierlich die Fähigkeiten verbessern.