

Dalam konteks development web, kita harus mempertimbangkan berbagai macam teknologi. Orang mungkin berinteraksi dengan situs web menggunakan pembaca layar atau tampilan braille, dengan pembesar layar, melalui kontrol suara, menggunakan perangkat switch, atau dengan bentuk teknologi pendukung lainnya yang mengadaptasikan antarmuka default laman untuk membuat antarmuka yang lebih spesifik yang bisa mereka gunakan.

Banyak dari teknologi pendukung ini yang mengandalkan *semantik yang dinyatakan lewat program* untuk membuat pengalaman pengguna yang bisa diakses, dan itulah yang sebagian besar akan dibahas pelajaran ini. Namun sebelum bisa menjelaskan semantik yang dijelaskan lewat program, kita perlu membicarakan sedikit tentang *kemampuan*.

6.2 Kemampuan

Bila kita menggunakan alat (bantu) atau perangkat atau buatan manusia, biasanya kita melihat bentuk dan desainnya untuk memberikan gambaran mengenai apa manfaatnya dan cara kerjanya. *Kemampuan* adalah objek yang menawarkan, atau memberi, penggunaanya kesempatan untuk melakukan suatu aksi. Semakin baik kemampuan tersebut didesain, semakin nyata atau intuitif penggunaannya.

Contoh klasik adalah cerek atau teko teh. Anda bisa dengan mudah mengetahui bahwa Anda harus mengambilnya melalui pegangannya, bukan di lehernya, sekalipun Anda belum pernah melihat teko itu sebelumnya.



Itu sebabnya kemampuan di sini mirip dengan kemampuan yang Anda lihat pada benda lainnya -- ceret penyiram, teko minuman, mug kopi, dan seterusnya. Anda barangkali *bisa* mengangkat teko pada lehernya, namun pengalaman Anda dengan kemampuan serupa akan memberi tahu pegangannya adalah opsi yang lebih baik.

Dalam Graphic User Interface, kemampuan menyatakan tindakan yang bisa kita ambil, namun hal itu bisa jadi meragukan karena tidak berinteraksi dengan objek fisik. Kemampuan GUI didesain khusus agar tidak meragukan: tombol, kotak centang, dan bilah gulir dimaksudkan untuk memberitahukan penggunaannya dengan pelatihan yang sesedikit mungkin.

Misalnya, Anda mungkin menafsirkan penggunaan beberapa elemen bentuk umum (kemampuan) seperti ini:

- Tombol radio — "Saya bisa memilih salah satu opsi ini."
- Kotak centang — "Saya bisa memilih 'ya' atau 'tidak' terhadap opsi ini."
- Bidang teks — "Saya bisa mengetikkan sesuatu ke dalam area ini."
- Menu tarik-turun — "Saya bisa membuka elemen ini untuk menampilkan opsi saya."

Anda bisa menarik kesimpulan tentang elemen ini *hanya karena Anda bisa melihatnya*. Secara alami, orang yang tidak bisa melihat petunjuk visual yang disediakan oleh sebuah elemen tidak bisa memahami maknanya atau secara intuitif menangkap nilai kemampuan. Jadi kita harus memastikan informasi diekspresikan cukup fleksibel untuk diakses oleh teknologi pendukung yang bisa membentuk suatu antarmuka alternatif agar cocok dengan kebutuhan penggunanya.

Paparan non-visual atas penggunaan kemampuan ini disebut *semantik*.

6.3 Pembaca layar

Satu tipe teknologi pendukung yang populer adalah *pembaca layar*, yaitu sebuah program yang memungkinkan orang yang memiliki masalah penglihatan untuk menggunakan komputer dengan membacakan teks layar dengan suara buatan. Pengguna bisa mengontrol apa yang dibaca dengan menggerakkan kursor ke area yang relevan dengan keyboard.

Kami meminta [Victor Tsaran](#) untuk menjelaskan bagaimana, sebagai orang buta, ia bisa mengakses web dengan menggunakan pembaca layar bawaan di OS X, yang disebut VoiceOver. Lihat [video ini](#) tentang Victor yang menggunakan VoiceOver.

Kini, giliran Anda mencoba menggunakan pembaca layar. Inilah laman dengan *ChromeVox Lite*, pembaca layar minimal namun fungsional yang ditulis dalam JavaScript. Layar sengaja dikaburkan untuk menyimulasikan pengalaman penglihatan-minim dan memaksa pengguna untuk melakukan tugas dengan pembaca layar. Tentu saja, Anda perlu menggunakan browser Chrome untuk latihan ini.

Laman demo ChromeVox lite

Anda bisa menggunakan panel kontrol di bagian bawah layar untuk mengontrol pembaca layar. Pembaca layar ini memiliki fungsionalitas sangat minim, namun Anda bisa menyusuri materi dengan menggunakan tombol **Previous** dan **Next**, dan bisa mengeklik sesuatu dengan menggunakan tombol **Click**.

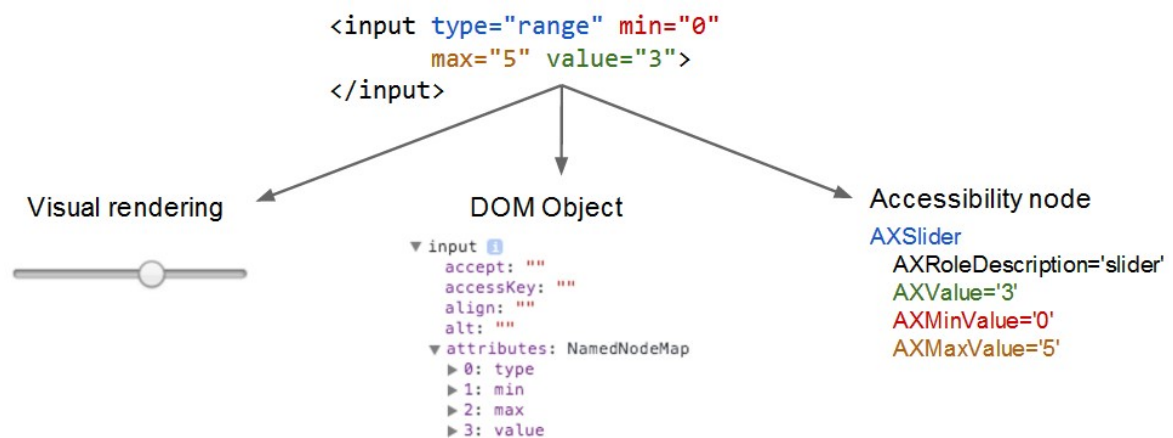
Cobalah menggunakan laman ini dengan ChromeVox lite yang telah diaktifkan untuk merasakan penggunaan pembaca layar. Ingatlah sebenarnya pembaca layar (atau teknologi pendukung lainnya) membuat suatu pengalaman pengguna alternatif lengkap bagi pengguna

berdasarkan pada semantik yang diekspresikan lewat program. Sebagai ganti antarmuka visual, pembaca layar menyediakan antarmuka yang terdengar.

Perhatikan bagaimana pembaca layar memberi tahu Anda informasi tentang setiap elemen antarmuka. Anda tentunya mengharapkan pembaca yang didesain dengan baik untuk memberi tahu Anda semua, atau setidaknya, informasi tentang elemen yang ditemukannya berikut ini.

- *Peran* atau tipe elemen, jika telah ditetapkan (seharusnya sudah).
- *Nama* elemen, jika memiliki (seharusnya sudah).
- *Nilai* elemen, jika memilikinya (mungkin atau mungkin tidak).
- *Keadaan* elemen, mis., apakah diaktifkan atau dinonaktifkan (jika berlaku).

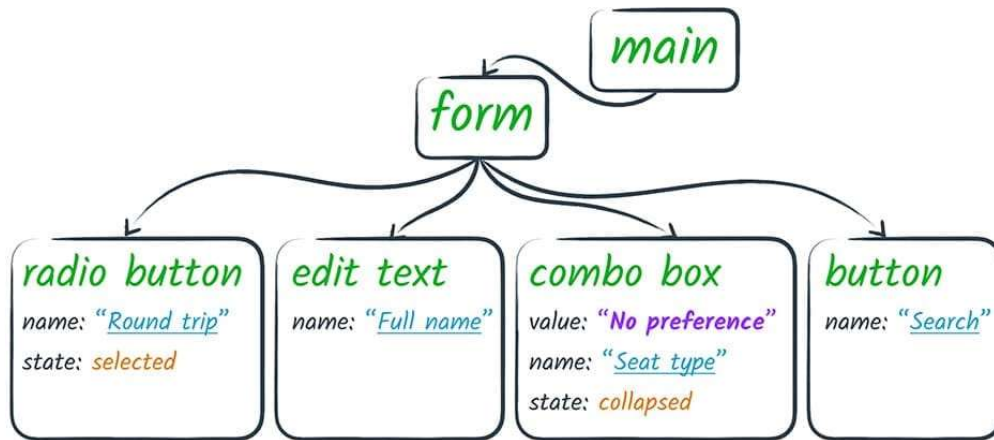
Pembaca layar mampu membentuk UI alternatif ini karena elemen asli berisi metadata aksesibilitas bawaan. Karena mesin rendering menggunakan kode asli untuk membentuk antarmuka visual, pembaca layar akan menggunakan metadata dalam simpul DOM untuk membentuk versi yang bisa diakses, seperti yang satu ini.



6.4 Pohon Aksesibilitas

Bayangkan Anda sedang membangun antarmuka pengguna *khusus untuk pengguna pembaca layar*. Di sini, Anda tidak perlu membuat UI visual sama sekali, melainkan cuma menyediakan informasi yang cukup untuk digunakan pembaca layar.

Yang akan Anda buat adalah semacam API yang menjelaskan struktur laman, mirip dengan DOM API, namun Anda bisa menghindari dengan sedikit informasi dan simpul lebih sedikit, karena banyak dari informasi itu yang hanya berguna bagi presentasi visual. Penampilannya mungkin seperti ini.



Pada dasarnya inilah yang sesungguhnya akan disajikan ke pembaca layar. Browser mengambil pohon DOM dan memodifikasinya menjadi suatu bentuk yang berguna untuk teknologi pendukung. Kami menyebut pohon yang telah dimodifikasi ini dengan *Pohon Aksesibilitas*.

Anda mungkin membayangkan pohon aksesibilitas ini seperti mirip dengan laman web tua dari tahun 90-an: sedikit gambar, banyak tautan, mungkin dengan satu bidang dan tombol.



Dengan memindai laman secara visual seperti ini akan memberi Anda pengalaman yang mirip dengan apa yang akan didapat oleh pengguna pembaca layar. Antarmuka memang ada, namun sederhana dan langsung, mirip sekali dengan antarmuka pohon aksesibilitas.

Kebanyakan teknologi pendukung berinteraksi dengan pohon aksesibilitas. Alurnya berjalan seperti ini.

1. Sebuah aplikasi (browser atau aplikasi lainnya) mengekspos versi semantik UI-nya kepada teknologi pendukung melalui API.
2. Teknologi pendukung dapat menggunakan informasi yang dibacanya melalui API untuk membuat presentasi antarmuka pengguna alternatif bagi pengguna. Misalnya, pembaca

layar akan membuat sebuah antarmuka yang digunakan pengguna untuk mendengarkan representasi aplikasi yang dibacakan.

3. Teknologi pendukung bisa juga memungkinkan pengguna berinteraksi dengan aplikasi dalam cara berbeda. Misalnya, kebanyakan pembaca layar menyediakan kait yang memungkinkan pengguna dengan mudah mensimulasikan klik mouse atau ketukan jari.
4. Teknologi pendukung yang menyampaikan maksud pengguna (misalnya "klik") kembali ke aplikasi melalui API aksesibilitas. Selanjutnya aplikasi bertanggung jawab untuk menafsirkan aksi sebagaimana mestinya dalam konteks UI asal.

Untuk browser web, ada langkah ekstra di setiap arah, karena browser sebenarnya adalah platform untuk menjalankan aplikasi web. Jadi browser perlu menerjemahkan aplikasi web menjadi pohon aksesibilitas, dan harus memastikan bahwa kejadian yang sesuai akan dipicu di JavaScript berdasarkan tindakan pengguna yang berasal dari teknologi pendukung.

Namun itu semua adalah tanggung jawab browser. Pekerjaan kita sebagai web developer sekadar mengetahui bahwa ini terjadi, dan mengembangkan laman web yang memanfaatkan proses ini untuk membuat suatu pengalaman yang bisa diakses oleh pengguna kita.

Kita melakukannya dengan memastikan bahwa kita mengekspresikan semantik laman dengan benar: dengan memastikan elemen penting di laman memiliki peran, keadaan, dan properti yang bisa diakses dengan benar, dan bahwa kita menetapkan nama dan keterangan yang bisa diakses. Selanjutnya browser bisa memungkinkan teknologi pendukung mengakses informasi itu untuk membuat pengalaman yang disesuaikan.

6.5 Semantik di HTML asli

Browser bisa mengubah pohon DOM menjadi sebuah pohon aksesibilitas karena kebanyakan DOM memiliki makna semantik *implicit*. Yakni, DOM menggunakan elemen HTML asli yang dikenali oleh browser dan berfungsi dengan cara yang bisa diprediksi pada berbagai platform. Aksesibilitas untuk elemen HTML asli seperti tautan atau tombol dengan demikian ditangani secara otomatis. Kita bisa memanfaatkan aksesibilitas bawaan itu dengan menulis HTML yang mengekspresikan semantik elemen laman kita.

Akan tetapi, kadang-kadang kita menggunakan elemen yang tampak seperti elemen asli padahal bukan. Misalnya, "tombol" bukanlah tombol sama sekali.

Ini dapat dibuat di HTML dengan banyak cara; salah satu caranya ditampilkan di bawah ini.

```
<div class="button-ish">Give me tacos</div>
```

Bila kita tidak menggunakan elemen tombol sesungguhnya, pembaca layar tidak memiliki cara untuk mengetahui telah sampai di mana. Selain itu, kita nanti harus melakukan pekerjaan ekstra berupa penambahan `tabindex` untuk membuatnya bisa digunakan oleh pengguna

keyboard-saja karena, terhubung sekarang telah dibuat kodenya, maka hanya bisa digunakan dengan mouse.

Kita bisa memperbaikinya secara mudah dengan menggunakan elemen `button` biasa sebagai ganti `div`. Penggunaan elemen asli juga berguna untuk menjagakan interaksi keyboard buat kita. Ingatlah bahwa Anda tidak harus kehilangan efek visual yang menyenangkan hanya lantaran menggunakan elemen asli; Anda bisa menata gaya elemen asli untuk membuatnya terlihat seperti yang Anda inginkan dan tetap mempertahankan semantik implisit dan perilakunya.

Sebelumnya kita telah memperhatikan bahwa pembaca layar akan membacakan peran, nama, keadaan, dan nilai elemen. Dengan menggunakan semantik yang tepat elemen, peran, keadaan, dan nilai telah tercakup, namun kita juga harus memastikan bahwa kita membuat nama elemen yang dapat ditemukan.

Secara umum, ada dua tipe nama:

- *Label yang terlihat*, yang digunakan oleh semua pengguna untuk mengaitkan makna dengan elemen, dan
- *Alternatif berupa teks*, yang hanya digunakan bila tidak memerlukan label visual.

Untuk elemen level-teks, kita tidak perlu melakukan apa-apa, karena menurut definisi, elemen akan berisi beberapa teks. Akan tetapi, untuk elemen masukan atau elemen kontrol, serta materi visual seperti gambar, kita perlu memastikan bahwa kita menetapkan sebuah nama. Sebenarnya, menyediakan alternatif berupa teks bagi materi non-teks adalah [item paling pertama pada daftar periksa WebAIM](#).

Salah satu cara melakukannya adalah mengikuti saran bahwa "Masukan formulir memiliki label teks terkait." Ada dua cara untuk mengaitkan label dengan elemen formulir, misalnya kotak centang. Salah satu dari metode ini menyebabkan teks label juga menjadi target klik untuk kotak centang, sehingga juga berguna bagi pengguna mouse atau layar sentuh. Untuk mengaitkan label dengan elemen, bisa dengan

Menempatkan elemen masukan di dalam elemen label

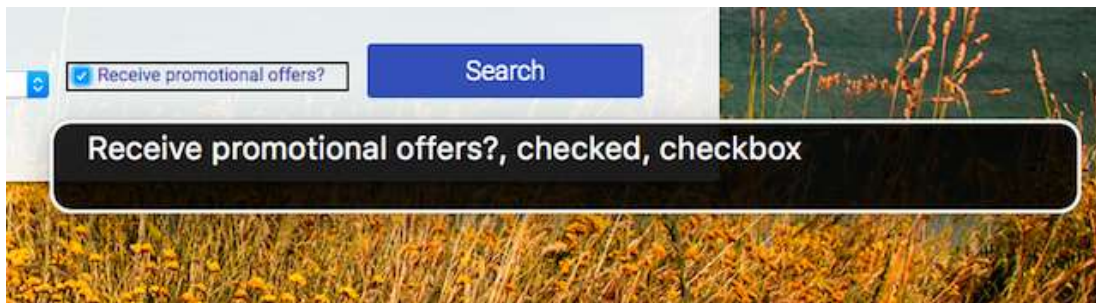
```
<label>  
  <input type="checkbox">Receive promotional offers?</input>  
</label>
```

atau

Menggunakan atribut `for` label dan merujuk `id` elemen

```
<input id="promo" type="checkbox"></input>  
<label for="promo">Receive promotional offers?</label>
```

Bila kotak centang telah diberi label dengan benar, pembaca layar bisa melaporkan bahwa elemen memiliki peran kotak centang, dalam keadaan dicentang, dan dinamai "Receive promotional offers?".



Berhasil: Sebenarnya Anda bisa menggunakan pembaca layar untuk menemukan label yang tidak dikaitkan dengan benar dengan berpindah-pindah tab di laman dan memverifikasi peran, keadaan, dan nama yang dibacakan.

6.6 Alternatif Berupa Teks untuk Gambar

Gambar adalah komponen penting pada sebagian besar laman web, dan tentu saja menjadi titik-lekat khusus bagi pengguna yang lemah penglihatannya. Kita harus mempertimbangkan peran yang dimainkan gambar di laman untuk merencanakan tipe alternatif berupa teks yang harus digunakan. Perhatikan gambar ini.

```
<article>
  <h2>Study shows 9 out of 10 cats quietly judging their owners as they sleep</h2>
  
</article>
```

Studi menunjukkan 9 dari 10 kucing dengan tenang menilai pemiliknya saat mereka tidur



Di sini, kita memiliki gambar seekor kucing, yang mengilustrasikan perilaku kecenderungan menilai yang sudah dikenali pada kucing. Pembaca layar akan membacakan gambar ini dengan menggunakan nama literalnya, `"/160204193356-01-cat-500.jpg"`. Itu memang akurat, namun tidak berguna sama sekali.

Anda bisa menggunakan atribut `alt` untuk menyediakan alternatif berupa teks yang berguna bagi gambar — misalnya, "A cat staring menacingly off into space."

```

```

Kemudian pembaca layar bisa membacakan keterangan singkat mengenai gambar tersebut (terlihat di bilah hitam VoiceOver) dan pengguna bisa memilih apakah akan berpindah ke artikel tersebut.



Sepasang komentar tentang `alt`:

- `alt` memungkinkan Anda menetapkan string sederhana untuk digunakan bila gambar tidak tersedia, misalnya bila gambar gagal dimuat, atau diakses melalui bot perayapan web, atau kebetulan ditemukan oleh pembaca layar.
- `alt` berbeda dengan `title`, atau tipe teks, yang *hanya* akan digunakan jika gambar tidak tersedia.

Menulis teks alternatif yang berguna ada seninya. Agar string bisa digunakan, alternatif berupa teks perlu menyampaikan konsep yang sama dengan gambarnya, dalam konteks yang sama.

Perhatikan gambar logo yang ditautkan di masthead laman seperti yang ditampilkan di atas. Kita bisa menjelaskan gambar tersebut dengan sangat akurat sebagai "logo The Funion".

```

```

Mungkin kita tergoda untuk memberikan alternatif berupa teks yang lebih sederhana berupa "beranda" atau "laman utama", namun itu tidak menguntungkan bagi pengguna yang penglihatannya lemah maupun tajam.

Namun bayangkan seorang pengguna pembaca layar yang ingin mencari logo masthead di laman; memberinya nilai alt berupa "beranda" sesungguhnya malah akan menambah bingung. Pengguna yang berpenglihatan tajam pun akan menghadapi kendala yang sama — yang mengetahui logo situs dengan mengekliknya — seperti halnya pengguna pembaca layar.

Di lain pihak, menjelaskan gambar tidak selalu berguna. Misalnya, perhatikan gambar kaca pembesar di dalam tombol telusur yang berisi teks "Telusur". Jika teks itu tidak ada, Anda pasti akan memberi gambar itu nilai alternatif berupa "telusur". Namun karena kita memiliki teks yang terlihat, pembaca layar akan mengambil dan membacakan kata "telusur"; sehingga, nilai alt yang identik pada gambar menjadi berlebihan.

Akan tetapi, kita tahu bahwa jika membiarkan teks alt, mungkin kita akan mendengar nama file gambar sebagai gantinya, yang tidak ada gunanya dan mungkin akan membingungkan. Dalam hal ini, Anda bisa menggunakan atribut alt kosong saja, dan pembaca layar akan melewati gambar sama sekali.

```

```

Singkatnya, semua gambar harus memiliki atribut alt, namun tidak semuanya harus memiliki teks. Gambar penting harus memiliki teks alternatif penjas yang menjelaskan secara singkat mengenai gambar itu, sedangkan gambar dekoratif harus memiliki atribut alt yang kosong — yakni, alt="".

6.7 Semantik dan Menyusuri Materi

Anda telah mempelajari tentang kemampuan, semantik, dan bagaimana teknologi pendukung menggunakan pohon aksesibilitas untuk membuat pengalaman pengguna alternatif bagi pengguna mereka. Anda bisa melihat bahwa menulis HTML semantik yang ekspresif akan memberi banyak aksesibilitas dengan upaya sangat kecil, karena banyak elemen standar memiliki semantik dan perilaku pendukung bawaan.

Dalam pelajaran ini, kita akan membahas beberapa semantik yang kurang dimengerti namun sangat penting bagi pengguna pembaca layar, terutama berkenaan dengan navigasi. Di laman sederhana yang berisi banyak kontrol namun tidak banyak materinya, akan mudah memindai laman untuk menemukan apa yang Anda butuhkan. Namun pada laman yang sarat materi, seperti entri Wikipedia atau agregator berita, tidak praktis membaca tuntas semua hal dari atas ke bawah; Anda perlu cara untuk menyusuri materinya secara efisien.

Developer sering kali salah memahami bahwa pembaca layar membosankan dan lambat digunakan, atau bahwa segala sesuatu di layar harus bisa difokus agar bisa ditemukan oleh pembaca layar. Sering kali bukan itu masalahnya.

Pengguna pembaca layar sering kali mengandalkan daftar heading untuk menemukan informasi. Kebanyakan pembaca layar memiliki cara mudah untuk mengisolasi dan memindai daftar heading laman, yakni sebuah fitur penting yang disebut *rotor*. Mari kita lihat cara menggunakan heading HTML secara efektif untuk mendukung fitur ini.

6.8 Menggunakan heading secara efektif

Pertama, mari kita ulangi kembali poin sebelumnya: *urutan DOM itu penting*, bukan hanya untuk urutan fokus melainkan untuk urutan pembaca layar. Saat Anda bereksperimen dengan pembaca layar seperti VoiceOver, NVDA, JAWS, dan ChromeVox, Anda akan menemukan daftar heading mengikuti urutan DOM, bukan urutan visual.

Hal ini berlaku untuk pembaca layar pada umumnya. Karena pembaca layar berinteraksi dengan pohon aksesibilitas, dan pohon aksesibilitas berdasarkan pada pohon DOM, urutan yang dipahami oleh pembaca dengan demikian berdasarkan pada urutan DOM. Ini berarti struktur heading yang tepat menjadi kian penting.

Di kebanyakan laman yang terstruktur dengan baik, level heading disarangkan untuk menunjukkan hubungan induk-anak di antara blok materi. [Daftar Periksa WebAIM](#) berulang kali merujuk teknik ini.

- [1.3.1](#) menyebutkan "Markup semantik digunakan untuk menunjukkan heading"
- [2.4.1](#) menyebutkan struktur heading sebagai teknik untuk melangkahi blok materi
- [2.4.6](#) mendiskusikan beberapa detail untuk penulisan heading yang berguna
- [2.4.10](#) menyebutkan "masing-masing bagian materi ditetapkan menggunakan heading, bila memang sesuai"

Tidak semua heading harus terlihat di layar. [Wikipedia](#), misalnya, menggunakan teknik yang sengaja menempatkan sebagian heading di luar layar untuk membuatnya *hanya* bisa diakses oleh pembaca layar dan teknologi pendukung lainnya.

```
<style>
.sr-only {
  position:absolute;
  left:-10000px;
  top:auto;
  width:1px;
  height:1px;
```

```
  overflow:hidden;
}
</style>

<h2 class="sr-only">This heading is
offscreen.</h2>
```

Note: Situs WebAIM mendiskusikan teknik ini panjang lebar dalam [artikel ini di materi di luar layar](#).

Bagi aplikasi yang kompleks, ini bisa menjadi cara yang bagus untuk mengakomodasi heading bila desain visual tidak memerlukan atau memiliki ruang bagi heading yang terlihat.

Perhatian: Perlu kiranya kita tidak bertindak lebih jauh dengan teknik ini. Ingatlah bahwa pengguna teknologi pendukung mungkin juga dapat melihat layar bagi dirinya sendiri, jadi melangkah terlalu jauh ke pembuatan materi untuk "pembaca layar saja" sebenarnya mungkin akan menurunkan kualitas pengalaman pengguna bagi sebagian orang. Hal ini juga bisa membuat Anda kerepotan dalam pemeliharannya nanti.

Opsi navigasi lainnya

Walaupun laman dengan heading yang baik akan membantu navigasi pengguna pembaca layar, ada beberapa elemen lainnya yang bisa mereka gunakan untuk menyusuri laman, termasuk *tautan*, *kontrol formulir*, dan *landmark*.

Pembaca bisa menggunakan fitur rotor milik pembaca layar (cara mudah untuk mengisolasi dan memindai daftar heading laman) untuk mengakses *daftar tautan* di laman tersebut. Kadang-kadang, seperti di wiki, ada banyak tautan, jadi pembaca dapat menelusuri suatu istilah dalam tautan tersebut. Ini akan membatasi hit pada tautan yang sebenarnya berisi istilah tersebut, bukannya setiap temuan istilah pada laman.

Fitur ini hanya berguna jika pembaca layar bisa menemukan tautan tersebut dan teks tautan itu bermakna. Misalnya, ini beberapa pola umum yang membuat tautan sulit ditemukan.

- Tag jangkar tanpa atribut href. Sering kali digunakan dalam aplikasi laman-tunggal, target tautan ini menyebabkan masalah bagi pembaca layar. Anda bisa membaca selengkapnya di [artikel mengenai aplikasi laman-tunggal ini](#).
- Tombol yang diimplementasikan bersama tautan. Hal ini menyebabkan pembaca layar menafsirkan materi sebagai tautan, dan fungsionalitas tombol akan hilang. Untuk kasus-kasus ini, ganti tag jangkar dengan tombol sungguhan dan beri gaya dengan semestinya.
- Gambar digunakan sebagai materi tautan. Kadang-kadang diperlukan, gambar bertautan bisa menjadi tidak berguna bagi pembaca layar. Untuk menjamin bahwa tautan diekspos dengan benar ke teknologi pendukung, pastikan gambar tersebut memiliki teks atribut alt.

Masalah lain adalah tautan yang buruk. Teks yang bisa diklik seperti "ketahui selengkapnya" atau "klik di sini" tidak menyediakan informasi semantik tentang akan ke mana tautan tersebut. Sebagai gantinya, gunakan teks deskriptif seperti "ketahui selengkapnya tentang desain responsif" atau "lihat tutorial kanvas ini" untuk membantu pembaca layar menyediakan konteks yang bermakna tentang tautan.

Rotor bisa juga mengambil *daftar kontrol formulir*. Dengan daftar ini, pembaca bisa menelusuri item tertentu dan langsung menuju ke sana.

Kesalahan umum yang dibuat pembaca layar adalah pengucapan. Misalnya, pembaca layar mungkin mengucapkan "Udacity" sebagai "oo-dacity", atau membacakan nomor ponsel sebagai integer besar, atau membaca teks berhuruf besar seakan berupa akronim. Menariknya, pengguna pembaca layar sudah sangat terbiasa dengan keganjilan ini dan telah memperhitungkannya.

Sebagian developer mencoba memperbaiki situasi ini dengan menyediakan teks khusus pembaca layar yang dieja secara fonetik. Inilah aturan sederhana untuk ejaan fonetik: **jangan lakukan**; ini hanya akan memperburuk masalah! Jika, misalnya, pengguna menggunakan tampilan braille, kata akan salah eja, sehingga menyebabkan semakin bingung. Pembaca layar memungkinkan kata dieja nyaring, sehingga membiarkan pembaca mengontrol pengalaman mereka dan memutuskan kapan memerlukannya.

Pembaca bisa menggunakan rotor untuk melihat *daftar landmark*. Ini membantu pembaca menemukan materi utama dan serangkaian landmark navigasi yang disediakan oleh elemen landmark HTML.

HTML5 memperkenalkan beberapa elemen baru yang membantu mendefinisikan struktur semantik laman, termasuk `header`, `footer`, `nav`, `article`, `section`, `main`, dan `aside`. Semua elemen ini secara spesifik menyediakan petunjuk struktural di laman tanpa memaksakan penataan gaya bawaan (yang bagaimana pun juga harus Anda lakukan dengan CSS).

Elemen struktural semantik menggantikan beberapa blok `div` repetitif, dan menyediakan cara yang lebih jelas dan lebih deskriptif untuk menyatakan struktur laman secara intuitif baik bagi penulis maupun pembaca.

BAB VII Pengantar ARIA

Sejauh ini, kita didorong untuk menggunakan elemen HTML asli karena memberi Anda fokus, dukungan keyboard, dan semantik bawaan, namun ada kalanya layout sederhana dan HTML asli tidak dapat menjalankan tugasnya. Misalnya, saat ini tidak ada elemen HTML terstandarisasi untuk bentuk UI umum, menu munculan. Atau tidak ada elemen HTML yang menyediakan karakteristik semantik, misalnya "pengguna perlu mengetahui tentang hal ini secepatnya".

Dalam pelajaran ini, nanti, kita akan mendalami cara mengekspresikan semantik yang tidak bisa dinyatakan dalam HTML.

Spesifikasi [Web Accessibility Initiative's Accessible Rich Internet Applications](#) (WAI-ARIA, atau cukup ARIA) bagus untuk menjembatani bidang-bidang masalah aksesibilitas yang tidak bisa ditangani dengan HTML asli. Hal ini bisa dilakukan dengan memungkinkan Anda menetapkan atribut yang memodifikasi cara elemen diterjemahkan ke dalam pohon aksesibilitas. Mari kita amati sebuah contoh.

Dalam cuplikan kode berikut, kita menggunakan item daftar sebagai kotak centang khusus. Kelas "checkbox" CSS memberi elemen karakteristik visual yang diperlukan.

```
<li tabindex="0" class="checkbox" checked>
  Receive promotional offers
</li>
```

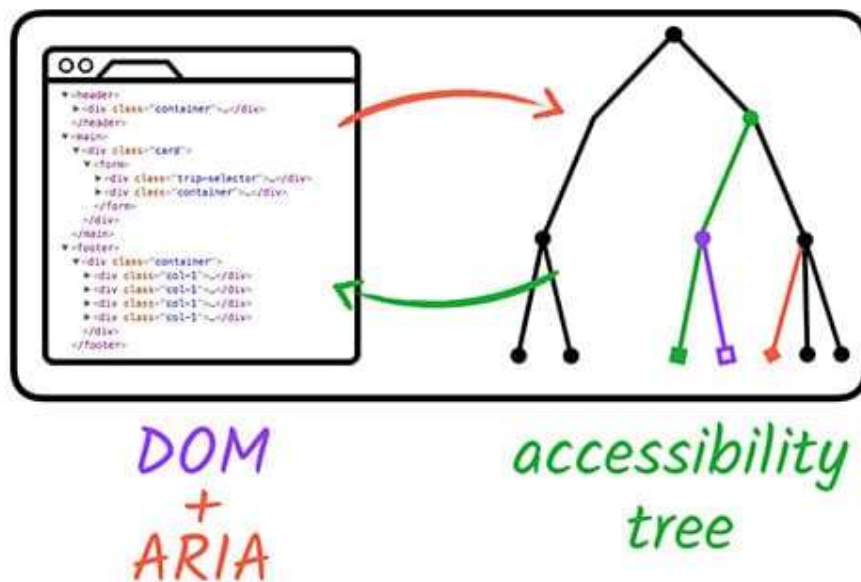
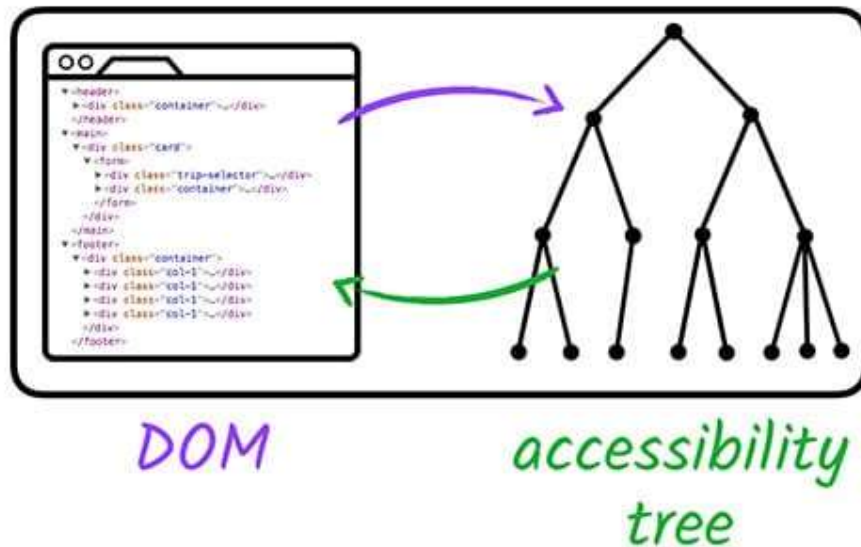
Walaupun cocok bagi pengguna yang berpenglihatan normal, pembaca layar tidak akan memberikan indikasi bahwa elemen dimaksudkan sebagai kotak centang, sehingga pengguna yang berpenglihatan lemah mungkin melewati elemen itu sama sekali.

Akan tetapi, dengan menggunakan atribut ARIA, kita bisa memberi elemen tersebut informasi yang terlewat sehingga pembaca layar bisa menafsirkannya dengan benar. Di sini, kita telah menambahkan atribut `role` dan `aria-checked` untuk mengidentifikasi secara eksplisit elemen tersebut sebagai kotak centang dan menetapkan bahwa elemen itu telah dicentang secara default. Kini item daftar akan ditambahkan ke pohon aksesibilitas dan pembaca layar akan melaporkannya dengan benar sebagai kotak centang.

```
<li tabindex="0" class="checkbox" role="checkbox" checked aria-checked="true">
  Receive promotional offers
</li>
```

Note: Kita akan membahas daftar atribut ARIA dan kapan menggunakannya [nanti](#).

ARIA bekerja dengan mengubah dan menambah pohon aksesibilitas DOM standar.



Walaupun ARIA memungkinkan kita secara halus (atau bahkan secara radikal) memodifikasi pohon aksesibilitas untuk elemen apa saja di laman, namun hanya elemen itu yang berubah. **ARIA tidak menambah perilaku inheren elemen**; ia tidak akan membuat elemen

dapat difokus atau memberinya event listener untuk keyboard. Itu tetap merupakan bagian dari tugas development kami.

Penting dipahami bahwa semantik default tidak perlu didefinisikan ulang. Apa pun penggunaannya, elemen `<input type="checkbox">` HTML standar tidak membutuhkan atribut ARIA tambahan `role="checkbox"` untuk diumumkan dengan benar.

Juga perlu diperhatikan bahwa elemen HTML tertentu memiliki batasan atas peran dan atribut ARIA apa saja yang bisa digunakan padanya. Misalnya, mungkin tidak ada peran/atribut tambahan yang diterapkan pada elemen `<input type="text">`.

Lihat [ARIA di spesifikasi HTML](#) untuk informasi selengkapnya.

Mari kita lihat kemampuan lain yang ditawarkan ARIA.

Apa yang bisa dilakukan ARIA?

Seperti yang Anda lihat pada contoh kotak centang, ARIA bisa memodifikasi semantik elemen yang ada atau menambahkan pada elemen bila tidak ada semantik asli. Pola semantik yang tidak ada sama sekali di HTML juga bisa dinyatakan, seperti menu atau panel tab. Sering kali, ARIA memungkinkan kita membuat elemen bertipe widget yang tidak akan memungkinkan bila dengan HTML biasa.

Misalnya, ARIA bisa menambahkan label ekstra dan teks keterangan yang hanya diekspos kepada API teknologi pendukung.

```
<button aria-label="screen reader only label"></button>
```

ARIA bisa menyatakan hubungan semantik antar elemen yang memperluas hubungan induk/anak standar, misalnya bilah gulir khusus yang mengontrol region tertentu.

```
<div role="scrollbar" aria-controls="main"></div>
<div id="main">
...
</div>
```

ARIA bisa "menghidupkan" bagian laman, agar segera memberi tahu teknologi pendukung bila bagian itu berubah.

```
<div aria-live="true">
  <span>GOOG: $400</span>
</div>
```

Salah satu dari aspek inti sistem ARIA adalah kumpulan *peran*-nya. Peran dalam istilah aksesibilitas setara dengan indikator singkatan untuk pola UI tertentu. ARIA menyediakan kosakata pola yang bisa kita gunakan lewat atribut `role` pada suatu elemen HTML.

Bila kita menerapkan `role="checkbox"` dalam contoh sebelumnya, kita memberi tahu teknologi pendukung bahwa elemen harus mengikuti pola "kotak centang". Yakni, kami menjamin bahwa keadaannya akan dicentang (baik telah dicentang atau belum dicentang), dan bahwa keadaan itu dapat diubah-ubah menggunakan mouse atau tombol spasi, persis seperti elemen kotak centang HTML standar.

Kenyataannya, karena fitur interaksi keyboard begitu kentara dalam penggunaan pembaca layar, maka penting sekali memastikan bahwa, saat membuat widget khusus, atribut `role` selalu diterapkan di tempat yang sama seperti atribut `tabindex`; ini memastikan bahwa kejadian keyboard pindah ke tempat yang tepat dan bahwa bila fokus jatuh pada suatu elemen, perannya akan dinyatakan dengan akurat.

Spesifikasi ARIA menjelaskan taksonomi nilai-nilai yang memungkinkan untuk atribut `role` dan atribut ARIA terkait yang boleh digunakan bersama-sama peran itu. Inilah sumber informasi definitif terbaik tentang cara kerja sama peran dan atribut ARIA dan bagaimana keduanya bisa digunakan dalam cara yang didukung oleh browser dan teknologi pendukung.

alert
alertdialog
button
checkbox
dialog
gridcell
link
log
marquee
menuitem
menuitemcheckbox
menuitemradio
option
progressbar
radio
scrollbar
slider
spinbutton
status
tab
tabpanel
textbox
timer
tooltip
treeitem
combobox
grid
listbox
menu
menubar
radiogroup
tablist
tree
treegrid
article
columnheader
definition
directory
document
group
heading
img
list
listitem
math
note
presentation
region
row
rowgroup
rowheader
separator
toolbar
application
banner
complementary
contentinfo
form
main
navigation
search

Akan tetapi, spesifikasinya sangat padat; tempat yang lebih mudah dicapai untuk memulai adalah [dokumen ARIA Authoring Practices](#), yang mendalami berbagai praktik terbaik untuk menggunakan peran dan properti ARIA yang tersedia.

ARIA juga menawarkan peran landmark yang memperluas opsi yang tersedia di HTML5. Lihat spesifikasi [Landmark Roles Design Patterns](#) untuk informasi selengkapnya.

7.1 Hubungan dan Label ARIA

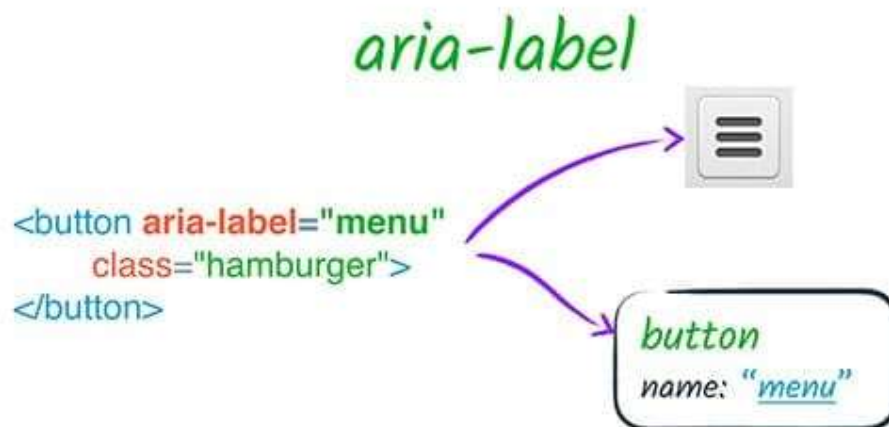
Label

ARIA menyediakan sejumlah mekanisme untuk menambahkan label dan keterangan ke elemen. Sebenarnya, ARIA adalah satu-satunya cara untuk menambahkan bantuan atau teks keterangan yang bisa diakses. Mari kita amati berbagai properti yang digunakan ARIA untuk membuat label yang bisa diakses.

aria-label

`aria-label` memungkinkan kita untuk menetapkan string yang akan digunakan sebagai label yang bisa diakses. Ini akan menggantikan mekanisme pelabelan asli lainnya, seperti elemen label — misalnya, jika `button` memiliki materi teks dan sebuah `aria-label`, maka hanya nilai `aria-label` yang akan digunakan.

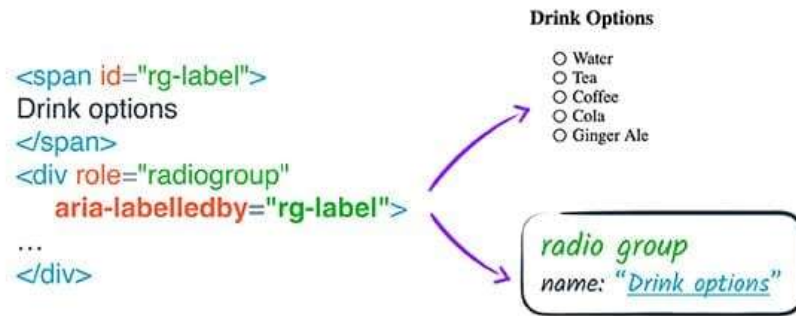
Anda dapat menggunakan atribut `aria-label` bila memiliki semacam indikasi visual kegunaan elemen, misalnya tombol yang menggunakan grafik sebagai ganti teks, namun tetap perlu mengklarifikasi kegunaan itu untuk siapa saja yang tidak bisa mengakses indikasi visual, misalnya tombol yang hanya menggunakan gambar untuk menunjukkan kegunaannya.



aria-labelledby

`aria-labelledby` memungkinkan kita menetapkan ID elemen lain dalam DOM sebagai label elemen.

aria-labelledby



Ini mirip sekali dengan penggunaan elemen `label`, dengan beberapa perbedaan penting.

1. `aria-labelledby` dapat digunakan pada sembarang elemen, tidak cuma elemen yang bisa diberi label.
2. Walaupun elemen `label` merujuk pada sesuatu yang dilabelinya, hubungannya terbalik untuk `aria-labelledby` — sesuatu yang diberi label merujuk pada sesuatu yang melabelinya.
3. Hanya satu elemen `label` yang dapat dikaitkan dengan elemen yang bisa diberi label, namun `aria-labelledby` bisa menggunakan daftar IDREF untuk membuat label dari beberapa elemen sekaligus. Label akan digabung sesuai urutan IDREF yang diberikan.
4. Anda bisa menggunakan `aria-labelledby` untuk merujuk elemen yang disembunyikan dan bila tidak demikian tidak akan ada dalam pohon aksesibilitas. Misalnya, Anda bisa menambahkan `span` tersembunyi di sebelah elemen yang ingin Anda beri label, dan merujuknya dengan `aria-labelledby`.
5. Akan tetapi, karena ARIA hanya memengaruhi pohon aksesibilitas, `aria-labelledby` tidak memberi Anda perilaku peneklikan label yang familier yang Anda dapat dari penggunaan elemen `label`.

Yang penting, `aria-labelledby` menggantikan **semua** sumber nama lainnya untuk elemen. Jadi, misalnya, jika sebuah elemen memiliki `aria-labelledby` dan `aria-label`, atau sebuah `aria-labelledby` dan label HTML asli, label `aria-labelledby` akan selalu didahulukan.

Hubungan

`aria-labelledby` adalah contoh sebuah *atribut hubungan*. Atribut hubungan membuat hubungan semantik antar elemen pada laman apa pun hubungan DOM-nya. Untuk `aria-labelledby`, hubungan itu adalah "elemen ini diberi label oleh elemen itu".

Spesifikasi ARIA mencantumkan **delapan atribut hubungan**. Enam di antaranya, `aria-activedescendant`, `aria-controls`, `aria-describedby`, `aria-labelledby`, dan `aria-owns`, mengambil referensi ke satu atau beberapa elemen untuk membuat tautan baru antar elemen pada laman.

Perbedaan di setiap kasus adalah apa arti tautan itu dan bagaimana menyajikannya kepada pengguna.

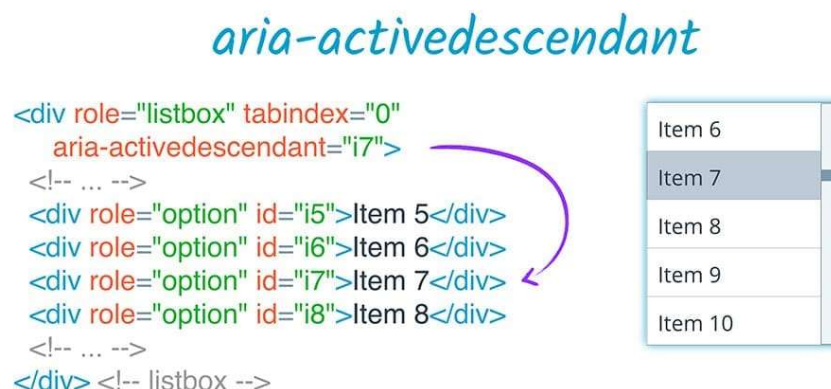
aria-owns

`aria-owns` adalah salah satu hubungan ARIA yang paling banyak digunakan. Atribut ini memungkinkan kita memberi tahu teknologi pendukung bahwa elemen yang terpisah di DOM harus diperlakukan sebagai anak dari elemen saat ini, atau untuk menyusun ulang elemen anak yang ada ke dalam urutan yang berbeda. Misalnya, jika sebuah sub-menu muncul secara visual diposisikan di dekat menu induknya, namun tidak bisa berupa anak DOM dari induknya karena akan memengaruhi presentasi visual, Anda bisa menggunakan `aria-owns` untuk menyajikan sub-menu tersebut sebagai anak dari menu induk ke pembaca layar.



aria-activedescendant

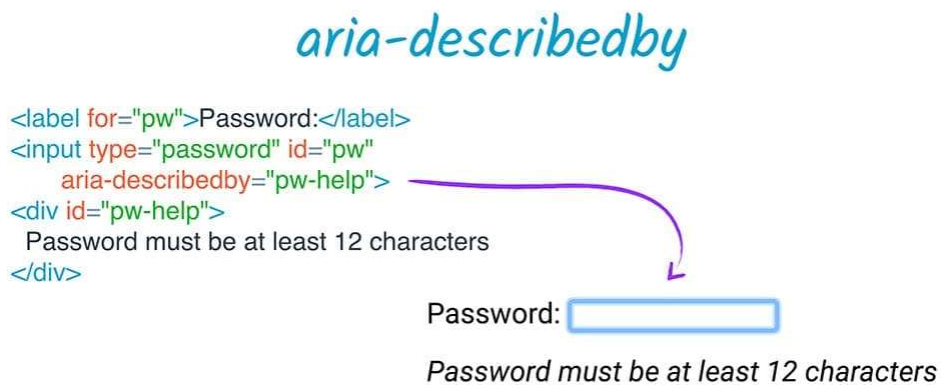
`aria-activedescendant` memainkan peranan terkait. Mirip dengan elemen aktif pada laman yang merupakan elemen berfokus, menyetel turunan aktif elemen memungkinkan kita untuk memberi tahu teknologi pendukung bahwa suatu elemen harus disajikan kepada pengguna sebagai elemen berfokus bila induknya memiliki fokus. Misalnya, dalam listbox, Anda mungkin ingin membiarkan fokus laman pada kontainer listbox, namun tetap memperbarui atribut `aria-activedescendant`-nya ke item daftar yang dipilih saat ini. Hal ini membuat item yang dipilih saat ini tampak oleh teknologi pendukung seakan item yang difokus.



aria-describedby

`aria-describedby` menyediakan keterangan yang bisa diakses dengan cara yang sama dengan label yang disediakan oleh `aria-labelledby`. Seperti halnya `aria-labelledby`, `aria-describedby` dapat mereferensikan elemen yang dalam keadaan lain tidak akan terlihat, baik yang disembunyikan dari DOM, atau yang disembunyikan dari pengguna teknologi pendukung. Ini merupakan teknik berguna bila ada beberapa teks penjelasan tambahan yang mungkin dibutuhkan pengguna, baik yang hanya berlaku pada pengguna teknologi pendukung atau pun semua pengguna.

Contoh umum adalah bidang masukan sandi dengan sejumlah teks deskriptif yang menjelaskan persyaratan sandi minimum. Tidak seperti label, keterangan ini mungkin atau mungkin tidak akan pernah disajikan kepada pengguna; mereka mungkin memiliki pilihan apakah akan mengaksesnya, atau keterangan tersebut mungkin ditampilkan setelah semua informasi lainnya, atau mungkin lebih dahulu ditempati oleh sesuatu yang lain. Misalnya, jika pengguna memasukkan informasi, masukan mereka akan dipantulkan kembali dan mungkin akan menginterupsi keterangan elemen. Sehingga, keterangan adalah cara yang bagus untuk mengomunikasikan informasi pelengkap, namun tidak esensial; ia tidak akan menghalangi informasi yang lebih penting seperti peran elemen.



aria-posinset & aria-setsize

Atribut hubungan selebihnya sedikit berbeda, dan bekerja bersama-sama. `aria-posinset` ("position in set") dan `aria-setsize` ("size of set") adalah tentang mendefinisikan hubungan antar elemen seinduk dalam suatu rangkaian, misalnya sebuah daftar.

Bila ukuran suatu rangkaian tidak bisa ditentukan melalui elemen yang ada dalam DOM — misalnya bila menggunakan lazy-rendering agar tidak semua daftar besar ada dalam DOM sekaligus — `aria-setsize` bisa menetapkan ukuran rangkaian sesungguhnya, dan `aria-posinset` bisa menetapkan posisi elemen dalam rangkaian tersebut. Misalnya dalam rangkaian yang bisa berisi 1000 elemen, anggaplah sebuah elemen tertentu memiliki `aria-posinset` sebesar 857 walaupun muncul lebih dahulu dalam DOM, kemudian menggunakan teknik HTML dinamis untuk memastikan pengguna bisa menyusuri daftar lengkap sesuai kebutuhan.

aria-posinset and aria-setsize

```
<div role="listbox">
  <div role="option"
    aria-posinset="857"
    aria-setsize="1000">Item 857</div>
  <div role="option"
    aria-posinset="858"
    aria-setsize="1000">Item 858</div>
  <!-- items 859-862 here -->
</div> <!-- listbox -->
```

Item 857
Item 858
Item 859
Item 860
Item 861

7.2 Menyembunyikan dan Memperbarui Materi

aria-hidden

Teknik penting lainnya dalam penyempurnaan pengalaman untuk pengguna teknologi pendukung antara lain memastikan bahwa hanya bagian laman yang relevan yang diekspos ke teknologi pendukung. Ada sejumlah cara untuk memastikan bagian DOM tidak diekspos ke API aksesibilitas.

Pertama, apa saja yang secara eksplisit disembunyikan dari DOM juga tidak akan disertakan di pohon aksesibilitas. Sehingga apa saja yang memiliki gaya CSS dengan atribut `visibility:hidden` atau `display: none` atau menggunakan `hidden` HTML5 juga akan disembunyikan dari pengguna teknologi pendukung.

Akan tetapi, elemen yang secara visual tidak dirender namun tidak secara eksplisit disembunyikan akan tetap disertakan dalam pohon aksesibilitas. Salah satu teknik umum adalah menyertakan "teks khusus pembaca layar" dalam elemen yang diposisikan di luar layar secara mutlak.

```
.sr-only {
  position: absolute;
  left: -10000px;
  width: 1px;
  height: 1px;
  overflow: hidden;
}
```

Selain itu, seperti yang telah kita lihat, bisa saja menyediakan teks khusus pembaca layar lewat atribut `aria-label`, `aria-labelledby`, atau `aria-describedby` yang mereferensikan elemen yang dalam keadaan lain disembunyikan.

Lihat artikel WebAIM ini di [Teknik untuk menyembunyikan teks](#) untuk informasi selengkapnya mengenai pembuatan teks "khusus pembaca layar".

Terakhir, ARIA menyediakan mekanisme untuk mengecualikan materi dari teknologi pendukung tidak secara visual disembunyikan, dengan menggunakan atribut `aria-hidden`. Menerapkan atribut ini ke elemen secara efektif akan membuangnya *dan semua turunannya* dari pohon aksesibilitas. Pengecualian satu-satunya adalah elemen yang dirujuk melalui atribut `aria-labelledby` atau `aria-describedby`.

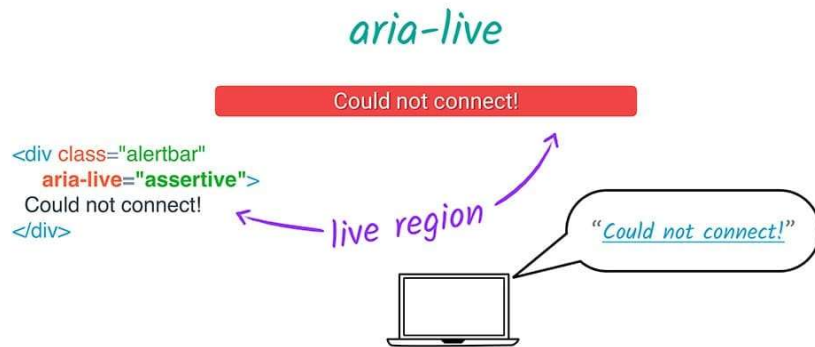
```
<div class="deck">
  <div class="slide" aria-hidden="true">
    Sales Targets
  </div>
  <div class="slide">
    Quarterly Sales
  </div>
  <div class="slide" aria-hidden="true">
    Action Items
  </div>
</div>
```

Misalnya, Anda dapat menggunakan `aria-hidden` jika membuat beberapa UI modal yang memblokir akses ke laman utama. Dalam hal ini, pengguna yang berpenglihatan normal mungkin akan melihat semacam overlay semi-transparan yang menunjukkan bahwa sebagian besar laman saat ini tidak bisa digunakan, namun pengguna pembaca layar mungkin tetap bisa menjelajahi bagian lain di laman tersebut. Dalam hal ini, seperti halnya membuat jebakan keyboard [telah dijelaskan sebelumnya](#), Anda perlu memastikan bahwa bagian-bagian laman yang saat ini berada di luar cakupan berupa `aria-hidden` juga.

Karena kini Anda telah memahami dasar-dasar ARIA, cara memainkannya dengan semantik HTML asli, dan cara menggunakannya untuk melakukan pembedahan yang cukup besar pada pohon aksesibilitas serta pengubahan semantik elemen tunggal, mari kita amati cara menggunakannya untuk menyampaikan informasi yang sensitif terhadap waktu.

aria-live

`aria-live` memungkinkan developer menandai bagian laman sebagai "live" dalam artian bahwa pembaruan harus segera dikomunikasikan dengan pengguna, tanpa memperhatikan posisi laman, daripada cuma terjadi saat menjelajahi bagian laman tersebut. Bila elemen memiliki atribut `aria-live`, bagian laman yang berisi elemen tersebut dan turunannya disebut *live region*.



Misalnya, live region dapat berupa pesan status yang muncul sebagai hasil aksi pengguna. Jika pesan tersebut cukup penting untuk menarik perhatian pengguna yang berpenglihatan normal, berarti cukup penting untuk mengarahkan perhatian pengguna teknologi pendukung ke pesan tersebut dengan menyematkan atribut `aria-live`-nya. Bandingkan `div` biasa ini

```
<div class="status">Your message has been sent.</div>
```

dengan pasangan "live"-nya.

```
<div class="status" aria-live="polite">Your message has been sent.</div>
```

`aria-live` memiliki tiga nilai yang diperbolehkan: `polite`, `assertive`, dan `off`.

- `aria-live="polite"` memberi tahu teknologi pendukung untuk memperingatkan pengguna pada perubahan ini bila telah menyelesaikan pekerjaan apa pun yang saat ini dilakukannya. Cocok sekali menggunakannya jika ada sesuatu yang penting namun tidak mendesak, dan inilah alasan untuk mayoritas penggunaan `aria-live`.
- `aria-live="assertive"` memberi tahu teknologi pendukung untuk menginterupsi apa pun yang sedang dilakukannya dan segera memperingatkan pengguna mengenai perubahan ini. Ini hanya untuk pemberitahuan penting dan mendesak, misalnya pesan status seperti "Ada kesalahan server dan perubahan Anda tidak disimpan; segarkan laman", atau pemberitahuan untuk bidang masukan sebagai akibat langsung dari aksi pengguna, misalnya tombol-tombol di stepper-widget.
- `aria-live="off"` memberi tahu teknologi pendukung untuk menangguhkan sementara interupsi `aria-live`.

Ada beberapa trik untuk memastikan live-region Anda bekerja dengan benar.

Pertama, region `aria-live` Anda harus telah disematkan dalam pemuatan laman pertama. Ini bukan aturan mutlak, melainkan jika Anda mengalami kesulitan dengan region `aria-live`, maka hal ini bisa menjadi masalah.

Kedua, pembaca layar yang berbeda akan bereaksi berbeda terhadap tipe perubahan yang berbeda. Misalnya, bisa saja memicu peringatan pada beberapa pembaca layar dengan mengubah gaya `hidden` elemen turunan dari `true` ke `false`.

Atribut lain yang bisa digunakan bersama `aria-live` akan membantu Anda menyempurnakan apa yang dikomunikasikan kepada pengguna bila `live-region` berubah.

`aria-atomic` menunjukkan apakah keseluruhan region harus dianggap sebagai satu kesatuan saat mengomunikasikan pembaruan. Misalnya, jika widget tanggal yang terdiri dari hari, bulan, dan tahun memiliki `aria-live=true` dan `aria-atomic=true`, dan pengguna menggunakan kontrol stepper untuk mengubah nilai bulan saja, maka isi lengkap dari widget tanggal akan dibaca lagi. Nilai `aria-atomic` mungkin menjadi `true` atau `false` (default-nya).

`aria-relevant` menunjukkan tipe perubahan yang harus disajikan kepada pengguna. Ada beberapa opsi yang dapat digunakan secara terpisah atau sebagai daftar token.

- *additions*, berarti elemen yang telah ditambahkan ke `live-region` adalah signifikan. Misalnya, penambahan bintang atau span ke log pesan status yang ada menunjukkan bahwa bintang itu akan diumumkan kepada pengguna (dengan anggapan `aria-atomic` adalah `false`).
- *text*, berarti isi teks yang akan ditambahkan ke simpul turunan adalah relevan. Misalnya, memodifikasi properti `textContent` bidang teks khusus akan membacakan teks yang dimodifikasi kepada pengguna.
- *removals*, berarti pembuangan suatu teks atau simpul turunan akan disampaikan kepada pengguna.
- *all*, berarti semua perubahan adalah relevan. Akan tetapi, nilai default `aria-relevant` adalah `additions text`, yang berarti jika Anda tidak menetapkan `aria-relevant`, ia akan memberi tahu pengguna mengenai penambahan ke elemen, yang kemungkinan besar merupakan hal yang memang Anda inginkan.

Terakhir, `aria-busy` memungkinkan Anda memberi tahu teknologi pendukung agar untuk sementara mengabaikan perubahan pada elemen, misalnya bila ada sesuatu yang sedang dimuat. Setelah semua berada pada tempatnya, `aria-busy` harus disetel ke `false` untuk menormalkan operasi pembaca.

7.3 Gaya yang Dapat Diakses

Kita telah mendalami dua pilar aksesibilitas, fokus, dan semantik yang sangat penting. Sekarang mari kita bahas yang ketiga, penataan gaya. Ini adalah topik luas yang dapat kita masukkan ke dalam tiga bagian.

- Memastikan bahwa elemen-elemen ditata gayanya untuk mendukung upaya aksesibilitas dengan menambahkan gaya untuk fokus dan beragam keadaan ARIA.

- Penataan gaya UI kami untuk fleksibilitas sehingga UI dapat diperbesar atau diatur skalanya guna mengakomodasi pengguna yang mungkin memiliki masalah dengan teks berukuran kecil.
- Memilih warna dan kontras yang tepat guna menghindari penyampaian informasi dengan warna saja.

Penataan gaya fokus

Umumnya, setiap kali kita memfokuskan elemen, kita mengandalkan lingkaran fokus browser bawaan (properti `outline` CSS) untuk menata gaya elemen. Lingkaran fokus ini berguna karena, tanpanya, mustahil pengguna keyboard dapat memberi tahu elemen mana yang memiliki fokus. [Daftar periksa WebAIM](#) menunjukkan pentingnya hal ini, yang mengharuskan bahwa "Tampak nyata secara visual elemen laman mana yang memiliki fokus keyboard saat ini (yakni, saat melakukan navigasi pada laman tersebut, Anda bisa melihat tempat Anda berada)."



Akan tetapi, kadang-kadang lingkaran fokus bisa tampak terdistorsi atau mungkin hanya tidak pas dengan desain laman Anda. Beberapa developer membuang gaya ini sama sekali dengan menyetel `outline` elemen ke 0 atau `none`. Namun tanpa indikator fokus, bagaimana pengguna keyboard dapat mengetahui dengan item mana ia berinteraksi?

Caution: Jangan menyetel `outline` ke 0 atau tidak ada tanpa memberikan alternatif fokus!

Anda mungkin familier dengan penambahan kondisi mengambang ke kontrol dengan menggunakan *kelas semu* CSS `:hover`. Misalnya, Anda mungkin menggunakan `:hover` pada elemen tautan untuk mengubah warna atau latar belakangnya saat mouse berada di atasnya. Serupa dengan `:hover`, Anda bisa menggunakan kelas semu `:focus` untuk menargetkan elemen bila memiliki fokus.

```
/* At a minimum you can add a focus style that matches your hover style */
: hover, : focus {
  background: #c0ffee;
}
```

Solusi alternatif untuk masalah menghapus lingkaran fokus adalah memberi elemen Anda gaya mengambang dan fokus yang sama, yang mengatasi masalah "di mana fokusnya?" untuk pengguna keyboard. Seperti biasa, meningkatkan pengalaman aksesibilitas berarti meningkatkan pengalaman seseorang.

Modalitas Input



Sign Up!

Untuk elemen bawaan seperti `button`, browser dapat mendeteksi apakah interaksi pengguna terjadi melalui mouse atau tekanan keyboard, dan biasanya hanya menampilkan lingkaran fokus untuk interaksi keyboard. Misalnya, bila Anda mengeklik `button` bawaan dengan mouse tidak ada lingkaran fokus, namun saat Anda menandainya dengan keyboard, lingkaran fokus akan muncul.

Logikanya di sini adalah bahwa pengguna mouse cenderung kurang memerlukan lingkaran fokus karena mereka tahu elemen apa yang mereka klik. Sayangnya saat ini tidak ada satu pun solusi lintas-browser yang menghasilkan perilaku yang sama ini. Sebagai hasilnya, jika Anda memberikan gaya `:focus` pada elemen apa pun, gaya tersebut akan ditampilkan *baik* bila pengguna mengeklik elemen atau memfokusnya dengan keyboard. Cobalah mengeklik tombol palsu ini dan perhatikan gaya `:focus` selalu diterapkan.

```
<style>
fake-button {
  display: inline-block;
  padding: 10px;
  border: 1px solid black;
  cursor: pointer;
  user-select: none;
}

fake-button:focus {
  outline: none;
  background: pink;
}
</style>
<fake-button tabindex="0">Click Me!</fake-button>
```

Ini bisa sedikit merepotkan, dan sering kali developer akan mengambil jalan untuk menggunakan JavaScript dengan kontrol khusus guna membedakan antara fokus mouse dan keyboard.

Di Firefox, CSS kelas semu `:-moz-focusing` memungkinkan Anda untuk menulis gaya fokus yang hanya berlaku jika elemen difokus melalui keyboard, sebuah fitur yang cukup bermanfaat. Sementara kelas semu saat ini hanya didukung di Firefox, [ada upaya yang sedang dilakukan untuk menjadikannya standar](#).

Ada juga [artikel sangat menarik yang ditulis oleh Alice Boxhall dan Brian Kardell](#) bahwa mengeksplorasi topik modalitas dan berisi kode prototipe untuk membedakan antara masukan

mouse dan keyboard. Anda bisa menggunakan solusinya sekarang, kemudian menyertakan kelas semu lingkaran fokus nanti bila sudah didukung secara luas.

Kondisi penataan gaya dengan ARIA

Saat Anda membangun komponen, praktik yang umum adalah mencerminkan keadaannya, dan penampilannya, dengan menggunakan kelas CSS, yang dikontrol dengan JavaScript.

Misalnya, perhatikan tombol toggle yang masuk ke dalam kondisi visual "ditekan" saat diklik dan mempertahankan kondisi tersebut sampai tombol diklik kembali. Untuk menata gaya kondisi ini, JavaScript Anda mungkin menambahkan kelas `pressed` ke tombol tersebut. Dan, karena Anda menginginkan semantik yang baik pada semua kontrol, Anda juga perlu menyetel keadaan `aria-pressed` untuk tombol ke `true`.

Teknik yang berguna untuk digunakan di sini adalah benar-benar membuang kelas, dan cukup gunakan atribut ARIA untuk menata gaya elemen. Kini Anda bisa memperbarui pemilih CSS untuk keadaan tombol yang ditekan dari

```
.toggle.pressed { ... }
```

ke ini.

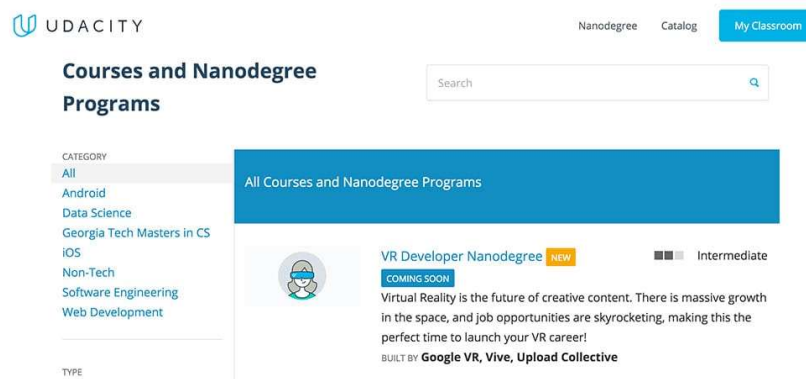
```
.toggle[aria-pressed="true"] { ... }
```

Hal ini menciptakan hubungan yang logis maupun semantik antara kondisi ARIA dan penampilan elemen, juga memangkas kode ekstra.

Desain responsif multi-perangkat

Kita mengetahui bahwa mendesain secara responsif guna menyediakan pengalaman multi-perangkat terbaik adalah ide bagus, namun desain responsif juga memiliki kelebihan dalam hal aksesibilitas.

Perhatikan situs seperti [Udacity.com](https://udacity.com):



Pengguna dengan kemampuan penglihatan rendah yang memiliki kesulitan membaca tulisan kecil mungkin memperbesar laman, mungkin sebesar 400%. Karena situs didesain secara responsif, UI akan mengatur ulang sendiri untuk "tampilan yang terlihat yang lebih kecil" (sebenarnya untuk laman yang lebih besar), yang sangat bagus untuk pengguna desktop yang memerlukan perbesaran layar dan untuk pengguna pembaca layar seluler juga. Ini saling menguntungkan. Ini adalah laman yang sama yang diperbesar hingga 400%:

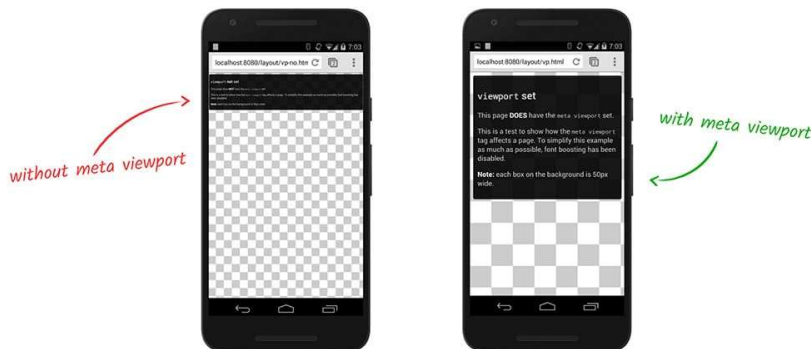


Courses and Nanodegree Programs

Kenyataannya, hanya dengan mendesain secara responsif, kita memenuhi [aturan 1.4.4 dari daftar periksa WebAIM](#), yang menyatakan bahwa suatu laman "...harus dapat dibaca dan fungsional saat ukuran teksnya digandakan."

Memeriksa semua desain responsif adalah di luar cakupan panduan ini, tetapi ini adalah beberapa hal yang perlu diingat dan penting yang akan bermanfaat bagi pengalaman responsif Anda dan memberi pengguna Anda akses yang lebih baik ke materi.

- Pertama, pastikan Anda selalu menggunakan tag meta viewport yang tepat. `<meta name="viewport" content="width=device-width, initial-scale=1.0">` Setelan `width=device-width` akan cocok dengan lebar layar dalam piksel yang tidak tergantung perangkat, dan setelan `initial-scale=1` menetapkan hubungan 1:1 antara piksel CSS dan piksel yang tidak tergantung perangkat. Melakukan hal ini akan memerintahkan browser untuk mengepaskan materi dengan ukuran layar, sehingga pengguna tidak hanya melihat sekumpulan teks.



Caution: Saat menggunakan tag meta viewport, pastikan Anda tidak menyetel `maximum-scale=1` atau menyetel `user-scalable=no`. Biarkan pengguna memperbesar jika mereka perlu!

- Teknik lain yang perlu diingat adalah mendesain dengan grid responsif. Saat Anda melihat dengan situs Udacity, mendesain dengan grid berarti materi Anda akan mengubah posisi/geometri saat laman mengubah ukuran. Layout ini sering kali dibuat menggunakan unit-unit relatif seperti persen, ems, atau rems sebagai ganti nilai piksel hard-code. Keuntungan melakukannya dengan cara ini adalah teks dan materi dapat memperbesar dan memaksa ke bawah laman. Jadi urutan DOM dan urutan pembacaan tetap sama, bahkan jika ada perubahan layout karena perbesaran.
- Selain itu, perhatikan agar menggunakan unit-unit relatif seperti `em` atau `rem` untuk hal-hal seperti ukuran teks, sebagai ganti nilai piksel. Beberapa browser mendukung pengubahan ukuran teks hanya dalam preferensi pengguna, dan jika Anda menggunakan nilai piksel untuk teks, setelan ini tidak akan memengaruhi salinan Anda. Namun jika Anda telah menggunakan unit-unit relatif seluruhnya, maka salinan situs akan diperbarui untuk mencerminkan preferensi pengguna.
- Terakhir, saat desain Anda ditampilkan di perangkat seluler, Anda harus memastikan bahwa elemen interaktif seperti tombol atau tautan cukup besar, dan memiliki cukup ruang di sekitarnya, sehingga mudah untuk ditekan tanpa secara tidak sengaja tumpang-tindih dengan elemen yang lain. Hal ini bermanfaat bagi semua pengguna, namun khususnya berguna bagi siapa saja yang memiliki penurunan motorik.

Ukuran target sentuh minimum yang disarankan adalah sekitar 48 piksel yang tidak tergantung perangkat pada situs dengan tampilan yang terlihat untuk seluler telah disetel dengan benar. Misalnya, sementara suatu ikon mungkin hanya memiliki lebar dan tinggi 24 piksel, Anda bisa menggunakan pengisi tambahan untuk menambahkan ukuran target ketuk hingga 48 piksel. Area piksel 48x48 sesuai dengan sekitar 9 mm yaitu sekitar ukuran area isi jari seseorang.



Target sentuh juga harus diberi ruang sekitar 8 piksel terpisah, baik horizontal maupun vertikal, sehingga jari pengguna yang menekan satu target ketuk tidak akan menyentuh target sentuh yang lain tanpa sengaja.

Warna dan kontras

Jika Anda memiliki penglihatan yang bagus, mudah untuk menganggap bahwa semua orang dapat melihat warna atau keterbacaan teks, dengan cara yang sama dengan Anda — namun tentu saja bukan itu masalahnya. Mari kita rangkum hal ini dengan melihat bagaimana kita secara efektif dapat menggunakan warna dan kontras untuk menciptakan desain menyenangkan yang dapat diakses bagi setiap orang.

Saat Anda dapat membayangkan, beberapa kombinasi warna yang mudah, bagi beberapa orang untuk dibaca ternyata sulit atau mustahil bagi orang lain. Ini biasanya bermuara pada *kontras warna*, hubungan antara *luminansi* warna latar belakang dan latar depan. Saat warna serupa, rasio kontras rendah; saat warna berbeda, rasio kontras pun tinggi.

[Panduan WebAIM](#) menyarankan rasio kontras AA (minimum) sebesar 4,5:1 untuk semua teks. Pengecualian akan dibuat untuk teks yang sangat besar (120-150% lebih besar dari teks isi default), yang rasionya dapat turun menjadi 3:1. Perhatikan perbedaan dalam rasio kontras yang ditampilkan di bawah ini.

15.9:1	5.7:1	3.5:1	1.6:1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Deleniti iusto inventore magni error, qui, eligendi sint pariatur dolorum.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Deleniti iusto inventore magni error, qui, eligendi sint pariatur dolorum.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Deleniti iusto inventore magni error, qui, eligendi sint pariatur dolorum.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Deleniti iusto inventore magni error, qui, eligendi sint pariatur dolorum.
<div> #222222</div> <div> #FFFFFF </div>	<div> #666666</div> <div> #FFFFFF </div>	<div> #888888</div> <div> #FFFFFF </div>	<div> #CCCCCC</div> <div> #FFFFFF </div>

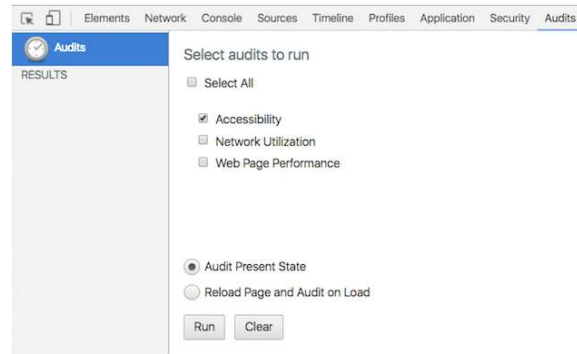
Rasio kontras 4,5:1 dipilih untuk level AA karena rasio ini mengganti hilangnya sensitivitas kontras yang biasanya dialami oleh pengguna yang kehilangan penglihatan setara dengan penglihatan sekitar 20/40. 20/40 umumnya dilaporkan sebagai ketajaman penglihatan biasa dari orang yang berusia sekitar 80 tahun. Untuk pengguna dengan gangguan penglihatan rendah atau defisiensi warna, kita dapat meningkatkan kontras hingga 7:1 untuk teks isi.

Anda bisa menggunakan [ekstensi Accessibility DevTools](#) untuk Chrome guna mengidentifikasi rasio kontras. Salah satu manfaat menggunakan Chrome Devtools adalah bahwa perangkat ini akan menyarankan alternatif AA dan AAA (disempurnakan) untuk warna Anda saat ini, dan Anda bisa mengeklik nilai untuk melakukan pratinjau di aplikasi.

Untuk menjalankan audit warna/kontras, ikuti langkah-langkah dasar ini.

1. Setelah memasang ekstensi, klik **Audits**
2. Hapus centang semuanya kecuali **Accessibility**
3. Klik **Audit Present State**

4. Perhatikan setiap peringatan kontras



WebAIM sendiri menyediakan [pemeriksa kontras warna](#) praktis yang bisa Anda gunakan untuk memeriksa kontras setiap pasangan warna.

Jangan menyampaikan informasi dengan warna saja

Ada sekitar 320 juta pengguna dengan defisiensi kemampuan melihat warna. Sekitar 1 dari 12 pria dan 1 dari 200 wanita memiliki beberapa bentuk "buta warna"; yang berarti sekitar 1/20 atau 5% pengguna Anda tidak akan merasakan pengalaman situs Anda sebagaimana yang Anda inginkan. Saat kita mengandalkan warna untuk menyampaikan informasi, kita menekan angka tersebut ke level yang tidak dapat diterima.

Note: Istilah "buta warna" sering kali digunakan untuk menjelaskan kondisi visual pada orang yang memiliki kesulitan dalam membedakan warna, namun sebenarnya hanya sedikit sekali orang yang benar-benar buta warna. Sebagian besar orang dengan defisiensi warna dapat melihat beberapa atau sebagian besar warna, namun memiliki kesulitan membedakan antara warna-warna tertentu seperti merah dengan hijau (yang paling umum), coklat dengan jingga, dan biru dengan ungu.

Misalnya, dalam sebuah formulir masukan, nomor telepon mungkin digarisbawahi dengan warna merah untuk menunjukkan bahwa nomor tersebut tidak valid. Namun bagi pengguna yang memiliki defisiensi warna atau pembaca layar, informasi tersebut tidak disampaikan dengan baik, bahkan benar-benar tidak baik. Maka, Anda harus selalu berupaya untuk menyediakan beberapa cara bagi pengguna untuk mengakses informasi penting.



Daftar periksa WebAIM menyatakan di bagian 1.4.1 bahwa "warna tidak boleh digunakan sebagai satu-satunya metode untuk menyampaikan materi atau membedakan elemen visual." Daftar periksa tersebut juga menyebutkan bahwa "warna saja tidak boleh digunakan untuk membedakan tautan dari teks sekelilingnya" kecuali jika warna tersebut memenuhi persyaratan kontras tertentu. Sebagai gantinya, daftar periksa menyarankan penambahan indikator tambahan seperti setrip bawah (menggunakan properti `text-decoration` CSS) untuk menunjukkan kapan tautan aktif.

Cara yang mudah untuk memperbaiki contoh sebelumnya adalah menambahkan pesan tambahan ke bidang tersebut, dengan mengumumkan bahwa ini tidak valid dan alasannya.



ACME Goods Co.

Billing / Shipping Information

Rob

Dodson

123 Google Road

555-5309

Missing area code!

Saat Anda membangun sebuah aplikasi, tetap perhatikan hal-hal ini dan berhati-hati dengan area yang Anda mungkin terlalu mengandalkan warna untuk menyampaikan informasi penting.

Jika Anda penasaran tentang bagaimana tampilan situs Anda bagi beberapa orang, atau jika Anda sangat mengandalkan penggunaan warna pada UI Anda, Anda bisa menggunakan [ekstensi NoCoffee Chrome](#) untuk menstimulasi beragam bentuk gangguan visual, termasuk berbagai jenis buta warna.

Mode kontras tinggi

Mode kontras tinggi memungkinkan pengguna untuk membalik warna latar belakang dan latar depan, yang sering kali membantu teks tampak lebih baik. Bagi mereka dengan gangguan penglihatan rendah, mode kontras tinggi dapat memudahkan mereka untuk menavigasi materi pada laman. Ada beberapa cara untuk memperoleh penyajian kontras tinggi di mesin Anda.

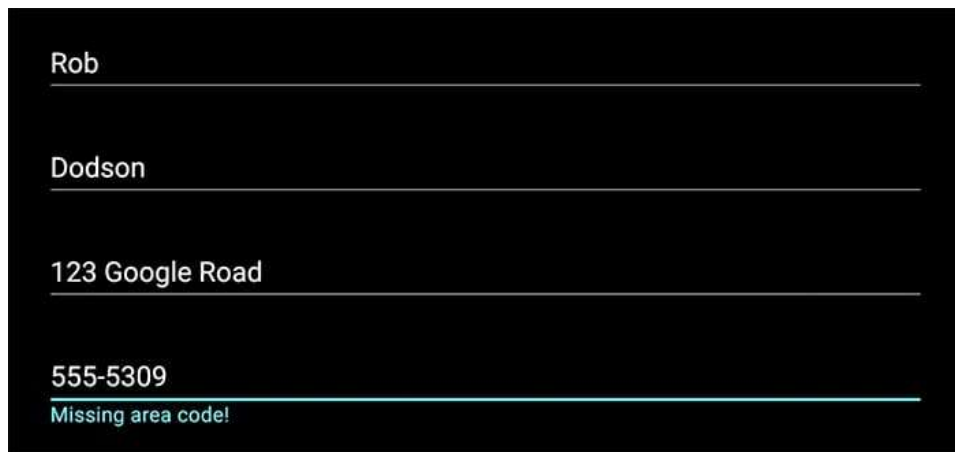
Sistem operasi seperti Mac OSX dan Windows menawarkan mode kontras tinggi yang dapat diaktifkan bagi siapa saja pada level sistem. Atau pengguna dapat memasang ekstensi, seperti [ekstensi Chrome High Contrast](#) untuk mengaktifkan kontras tinggi hanya di aplikasi khusus tersebut.

Latihan yang berguna dihidupkan pada setelan kontras tinggi dan memverifikasi bahwa semua UI di aplikasi Anda masih dapat dilihat dan dapat digunakan.

Misalnya, bilah navigasi dapat menggunakan warna latar belakang yang lembut untuk menunjukkan laman mana yang saat ini dipilih. Jika Anda menampilkannya dalam ekstensi kontras tinggi, yang perlahan seluruhnya menghilang, dan dengan cara ini pembaca mengerti laman mana yang aktif.



Demikian pula, jika Anda memperhatikan contoh dari pelajaran sebelumnya, garis bawah merah di bidang nomor ponsel yang tidak valid mungkin ditampilkan dalam warna biru-hijau yang sulit untuk dibedakan.



Jika Anda menemukan rasio kontras yang dicakup dalam pelajaran sebelumnya, Anda tidak akan mengalami masalah dalam hal mendukung mode kontras tinggi. Namun untuk menambah ketenangan, pertimbangkan untuk memasang ekstensi Chrome High Contrast dan periksa laman Anda untuk memeriksa apakah semuanya berfungsi, dan terlihat sebagaimana yang diharapkan.

7.4 Cara Melakukan Tinjauan Aksesibilitas

Menentukan apakah situs web atau aplikasi Anda dapat diakses seperti halnya merupakan tugas yang berat. Jika Anda mendekati aksesibilitas untuk pertama kalinya, luasnya topik dapat membuat Anda bertanya-tanya di mana harus memulai - setelah semua, bekerja untuk mengakomodasi beragam kemampuan berarti ada beragam masalah yang perlu dipertimbangkan. Dalam posting ini, saya akan memecah masalah ini menjadi logis, langkah demi langkah proses untuk meninjau situs yang ada untuk aksesibilitas.

Mulai dengan keyboard

Untuk pengguna yang tidak bisa atau memilih untuk tidak menggunakan mouse, navigasi keyboard adalah sarana utama mereka untuk mencapai semua yang ada di layar. Audiens ini termasuk pengguna dengan gangguan motorik, seperti Repetitive Stress Injury (RSI) atau kelumpuhan, serta pengguna pembaca layar. Untuk pengalaman keyboard yang bagus, bertujuan untuk memiliki urutan tab yang logis dan gaya fokus yang mudah dilihat.

Key points

- Mulailah dengan membedah situs Anda. Urutan di mana elemen difokuskan harus bertujuan untuk mengikuti urutan DOM. Jika Anda tidak yakin elemen mana yang harus menerima fokus, lihat Dasar-Dasar Fokus untuk penyegaran. Aturan umum yang umum adalah bahwa kontrol apa pun yang dapat berinteraksi dengan pengguna atau memberikan input agar bertujuan untuk menjadi fokus dan menampilkan indikator fokus (mis., Cincin fokus). Merupakan praktik umum untuk menonaktifkan gaya fokus tanpa memberikan alternatif dengan menggunakan garis besar: tidak ada dalam CSS, tetapi ini merupakan anti-pola. Jika pengguna keyboard tidak dapat melihat apa yang difokuskan, mereka tidak memiliki cara untuk berinteraksi dengan halaman. Jika Anda perlu membedakan antara fokus mouse dan keyboard untuk penataan, pertimbangkan untuk menambahkan perpustakaan seperti input apa.
- Kontrol interaktif khusus harus bertujuan untuk menjadi fokus. Jika Anda menggunakan JavaScript untuk mengubah `<div>` menjadi dropdown mewah, itu tidak akan secara otomatis dimasukkan ke dalam urutan tab. Untuk membuat kontrol khusus menjadi fokus, berikan `tabindex = "0"`.
- Hindari kontrol dengan `tabindex > 0`. Kontrol ini akan melompati semua yang lain dalam urutan tab, terlepas dari posisi mereka di DOM. Ini dapat membingungkan bagi pengguna pembaca layar karena mereka cenderung menavigasi DOM secara linear.
- Konten non-interaktif (mis., Judul) harus menghindari fokus. Terkadang pengembang menambahkan `tabindex` ke pos karena mereka menganggapnya penting. Ini juga merupakan pola anti-karena membuat pengguna keyboard yang dapat melihat layar kurang efisien. Untuk pengguna pembaca layar, pembaca layar sudah akan mengumumkan judul-judul ini, jadi tidak perlu membuatnya fokus.
- Jika konten baru ditambahkan ke halaman, cobalah untuk memastikan bahwa fokus pengguna diarahkan ke konten itu sehingga mereka dapat mengambil tindakan terhadapnya. Lihat Mengelola Fokus di Level Halaman untuk contoh.
- Waspadalah terhadap fokus yang sepenuhnya terperangkap di titik mana pun. Perhatikan widget pelengkapan otomatis, di mana fokus keyboard mungkin macet. Fokus dapat secara sementara terperangkap dalam situasi tertentu, seperti menampilkan modal, saat Anda tidak ingin pengguna berinteraksi dengan bagian halaman lainnya - tetapi Anda harus bertujuan untuk menyediakan metode yang dapat diakses dengan

keyboard untuk keluar dari modal juga. Lihat panduan tentang Modals dan Traps Keyboard untuk contoh.

Hanya karena sesuatu itu bisa fokus, bukan berarti itu bisa digunakan

Jika Anda telah membangun kontrol khusus, maka bertujuan agar pengguna dapat mencapai semua fungsinya hanya menggunakan keyboard. Lihat Mengelola Komponen Fokus Dalam untuk teknik meningkatkan akses keyboard.

Jangan lupa konten di luar layar

Banyak situs memiliki konten di luar layar yang ada dalam DOM tetapi tidak terlihat, misalnya, tautan di dalam menu laci responsif atau tombol di dalam jendela modal yang belum ditampilkan. Meninggalkan elemen-elemen ini di DOM dapat menyebabkan pengalaman papan ketik yang membingungkan, terutama bagi pembaca layar yang akan mengumumkan konten di luar layar seolah-olah itu adalah bagian dari halaman. Lihat Menangani Konten di Layar Terbuka untuk kiat tentang cara menangani elemen-elemen ini.

Cobalah dengan pembaca layar

Meningkatkan dukungan keyboard umum memberikan beberapa dasar untuk langkah selanjutnya, yaitu memeriksa halaman untuk pelabelan dan semantik yang tepat dan segala penghalang untuk navigasi pembaca layar. Jika Anda tidak terbiasa dengan bagaimana markup semantik ditafsirkan oleh teknologi bantuan, lihat Pengantar Semantik untuk penyegaran.

Key points

- Periksa semua gambar untuk teks alt yang benar. Pengecualian untuk praktik ini adalah ketika gambar terutama untuk tujuan presentasi dan bukan bagian penting dari konten. Untuk menandakan bahwa suatu gambar harus dilewati oleh pembaca layar, setel nilai atribut alt ke string kosong, mis., `Alt = ""`.
- Periksa semua kontrol untuk label. Untuk kontrol khusus, ini mungkin memerlukan penggunaan aria-label atau aria-labelledby oleh. Lihat Label dan Hubungan ARIA untuk contoh.
- Periksa semua kontrol khusus untuk peran yang sesuai dan atribut ARIA yang diperlukan yang memberikan status mereka. Misalnya, kotak centang khusus akan memerlukan `role = "checkbox"` dan `aria-checked = "true | false"` untuk menyampaikan statusnya dengan benar. Lihat Pengantar ARIA untuk ikhtisar umum tentang bagaimana ARIA dapat menyediakan semantik yang hilang untuk kontrol khusus.
- Aliran informasi harus masuk akal. Karena pembaca layar menavigasi halaman dalam urutan DOM, jika Anda menggunakan CSS untuk memposisikan ulang elemen secara visual, mereka dapat diumumkan dalam urutan yang tidak masuk akal. Jika Anda memerlukan sesuatu untuk muncul lebih awal di halaman, cobalah untuk memindahkannya secara fisik sebelumnya di DOM.

- Bertujuan untuk mendukung navigasi pembaca layar ke semua konten di halaman. Hindari membiarkan bagian mana pun dari situs disembunyikan secara permanen atau diblokir dari akses pembaca layar.
- Jika konten harus disembunyikan dari pembaca layar, misalnya, jika itu layar atau hanya presentasi, pastikan bahwa konten diatur ke `aria-hidden = "true"`. Lihatlah panduan tentang Menyembunyikan konten untuk penjelasan lebih lanjut.

Familiar dengan “even one screen reader goes a long way”

Meskipun mungkin sulit untuk mempelajari pembaca layar, sebenarnya cukup mudah untuk mengambilnya. Secara umum, sebagian besar pengembang dapat bertahan hanya dengan beberapa perintah kunci sederhana.

Jika Anda menggunakan Mac, periksa video ini tentang menggunakan VoiceOver, pembaca layar yang menyertai Mac OS. Jika Anda menggunakan PC untuk melihat video ini menggunakan NVDA, donasi yang didukung, pembaca layar sumber terbuka untuk Windows.

aria-hidden tidak mencegah fokus keyboard

Penting untuk dipahami bahwa ARIA hanya dapat memengaruhi semantik elemen; itu tidak berpengaruh pada perilaku elemen. Meskipun Anda dapat membuat elemen tersembunyi ke pembaca layar dengan `aria-hidden = "true"`, itu tidak mengubah perilaku fokus untuk elemen itu. Untuk konten interaktif di luar layar, Anda sering harus menggabungkan `aria-hidden = "true"` dan `tabindex = "-1"` untuk memastikan itu benar-benar dihapus dari aliran keyboard. Atribut inert yang diusulkan bertujuan untuk mempermudah ini dengan menggabungkan perilaku kedua atribut.

Elemen interaktif seperti tautan dan tombol harus menunjukkan tujuan dan statusnya

Memberikan petunjuk visual tentang apa yang akan dilakukan kontrol membantu orang mengoperasikan dan menavigasi situs Anda. Petunjuk ini disebut keterjangkauan. Menyediakan biaya memungkinkan orang menggunakan situs Anda di berbagai perangkat.

Key points

- Elemen interaktif, seperti tautan dan tombol, harus dapat dibedakan dari elemen non-interaktif. Sulit bagi pengguna untuk menavigasi situs atau aplikasi ketika mereka tidak tahu apakah suatu elemen dapat diklik. Ada banyak metode yang valid untuk mencapai tujuan ini. Salah satu praktik umum adalah tautan mendasar untuk membedakannya dari teks di sekitarnya.
- Mirip dengan persyaratan fokus, elemen-elemen interaktif seperti tautan dan tombol memerlukan keadaan mengambang untuk pengguna mouse sehingga mereka tahu jika mereka melayang di atas sesuatu yang dapat diklik. Namun, elemen interaktif masih harus dibedakan sendiri. Mengandalkan kondisi mengambang saja untuk menunjukkan elemen yang dapat diklik tidak membantu perangkat layar sentuh.

Manfaatkan headings dan landmarks

Heading dan elemen landmark mengilhami halaman Anda dengan struktur semantik, dan sangat meningkatkan efisiensi navigasi pengguna teknologi bantu. Banyak pengguna pembaca layar melaporkan bahwa, ketika mereka pertama kali mendarat di halaman yang tidak dikenal, mereka biasanya mencoba menavigasi dengan judul. Demikian pula, pembaca layar juga menawarkan kemampuan untuk melompat ke landmark penting seperti <main> dan <nav>. Untuk alasan ini, penting untuk mempertimbangkan bagaimana struktur halaman Anda dapat digunakan untuk memandu pengalaman pengguna.

Key points

- Manfaatkan hierarki h1-h6 dengan semestinya. Pikirkan pos sebagai alat untuk membuat garis besar halaman Anda. Jangan mengandalkan gaya tajuk bawaan; alih-alih, pertimbangkan semua judul seolah-olah ukurannya sama dan gunakan level semantik yang sesuai untuk konten primer, sekunder, dan tersier. Kemudian gunakan CSS untuk membuat judul sesuai dengan desain Anda.
- Gunakan elemen dan peran tengara sehingga pengguna dapat mem-bypass konten berulang. Banyak teknologi bantuan menyediakan pintasan untuk melompat ke bagian halaman tertentu, seperti yang ditentukan oleh elemen <main> atau <nav>. Elemen-elemen ini memiliki peran tengara yang tersirat. Anda juga dapat menggunakan atribut peran ARIA untuk secara eksplisit mendefinisikan kawasan pada halaman, mis., <Div role = "search">. Lihat panduan pada judul dan landmark untuk contoh lebih lanjut.
- Hindari role = "aplikasi" kecuali Anda memiliki pengalaman sebelumnya bekerja dengannya. Peran tengara aplikasi akan memberi tahu teknologi bantu untuk menonaktifkan pintasannya dan melewati semua penekanan tombol ke halaman. Ini berarti bahwa tombol yang biasanya digunakan pembaca layar untuk bergerak di sekitar halaman tidak akan berfungsi lagi, dan Anda harus menerapkan sendiri semua keyboard yang menanganinya.

Tinjau headings dan landmarks dengan cepat dengan pembaca layar

Pembaca layar seperti VoiceOver dan NVDA menyediakan menu konteks untuk melompat-lompat ke wilayah penting pada halaman. Jika Anda melakukan pemeriksaan aksesibilitas, Anda dapat menggunakan menu ini untuk mendapatkan ikhtisar cepat laman dan menentukan apakah tingkat tajuk sesuai dan tengara mana yang digunakan. Untuk mempelajari lebih lanjut, lihat video instruksional ini tentang dasar-dasar VoiceOver dan NVDA.

Otomatisasi proses

Menguji aksesibilitas suatu situs secara manual bisa membosankan dan rawan kesalahan. Akhirnya, Anda akan ingin mengotomatiskan proses sebanyak mungkin. Ini dapat dilakukan melalui penggunaan ekstensi browser, dan suite tes aksesibilitas baris perintah.

Key points

- Does the page pass all the tests from either the [aXe](#) or [WAVE](#) browser extensions? These extensions are just two available options and can be a useful addition to any manual test process as they can quickly pick up on subtle issues like failing contrast ratios and missing ARIA attributes. If you prefer to do things from the command line, [axe-cli](#) provides the same functionality as the aXe browser extension, but can be easily run from your terminal.
- To avoid regressions, especially in a continuous integration environment, incorporate a library like [axe-core](#) into your automated test suite. axe-core is the same engine that powers the aXe chrome extension, but in an easy-to-run command line utility.
- If you're using a framework or library, does it provide its own accessibility tools? Some examples include [protractor-accessibility-plugin](#) for Angular, and [a11ysuite](#) for Polymer and Web Components. Take advantage of available tools whenever possible to avoid reinventing the wheel.

Jika Anda membuat Aplikasi Web Progresif, pertimbangkan untuk mencoba Lighthouse Jika Anda membuat Aplikasi Web Progresif, pertimbangkan untuk mencoba Lighthouse

Lighthouse adalah alat untuk membantu mengukur kinerja aplikasi web progresif Anda, tetapi juga menggunakan pustaka inti untuk memberi daya pada serangkaian uji aksesibilitas. Jika Anda sudah menggunakan Mercusuar, awasi kegagalan tes aksesibilitas dalam laporan Anda. Memperbaiki ini akan membantu meningkatkan pengalaman pengguna keseluruhan situs Anda.

Wrapping Up

Membuat ulasan aksesibilitas menjadi bagian rutin dari proses tim Anda, dan melakukan pemeriksaan ini lebih awal dan sering, dapat membantu meningkatkan pengalaman keseluruhan menggunakan situs Anda. Ingat, aksesibilitas yang baik sama dengan UX yang baik!

7.5 Aksesibilitas untuk tim

Membuat situs Anda lebih mudah diakses bisa menjadi tugas yang menakutkan. Jika Anda mendekati aksesibilitas untuk pertama kalinya, luasnya topik dapat membuat Anda bertanya-tanya harus mulai dari mana. Bagaimanapun, bekerja untuk mengakomodasi beragam kemampuan berarti ada berbagai isu yang perlu dipertimbangkan.

Ingat, aksesibilitas adalah upaya tim. Setiap orang memiliki peran untuk dimainkan. Artikel ini menguraikan kriteria untuk masing-masing disiplin ilmu utama (manajer proyek, perancang UX, dan pengembang) sehingga mereka dapat bekerja untuk memasukkan praktik terbaik aksesibilitas ke dalam proses mereka.

Project manager

Tujuan utama untuk setiap manajer proyek adalah mencoba memasukkan pekerjaan aksesibilitas dalam setiap tonggak sejarah; memastikan itu sama prioritasnya dengan topik lain seperti kinerja, dan pengalaman pengguna. Di bawah ini adalah beberapa item daftar periksa yang perlu diingat saat mengerjakan proses Anda.

- Jadikan pelatihan aksesibilitas tersedia bagi tim.
- Identifikasi perjalanan pengguna yang kritis di situs atau aplikasi.
- Cobalah untuk memasukkan daftar periksa aksesibilitas ke dalam proses tim.
- Jika memungkinkan, evaluasi situs atau aplikasi dengan studi pengguna.

Pelatihan aksesibilitas

Ada sejumlah sumber daya gratis yang bagus untuk belajar tentang aksesibilitas web. Menyisihkan waktu untuk tim Anda untuk mempelajari topik dapat membuatnya lebih mudah untuk memasukkan aksesibilitas di awal proses.

Beberapa sumber daya yang disediakan oleh Google meliputi:

Aksesibilitas Web oleh Google - kursus pelatihan interaktif multi-minggu.

Fundamental Aksesibilitas - panduan aksesibilitas tertulis dan praktik terbaik.

Pedoman Bahan: Aksesibilitas - seperangkat praktik terbaik UX untuk desain inklusif.

Mengidentifikasi perjalanan pengguna yang kritis

Setiap aplikasi memiliki beberapa tindakan utama yang perlu dilakukan pengguna. Misalnya, jika Anda membuat aplikasi e-commerce, maka setiap pengguna harus dapat menambahkan item ke keranjang belanja mereka.

Primary user journey:

"A user can add an item to their shopping cart."

Beberapa tindakan mungkin memiliki kepentingan sekunder, dan mungkin hanya dilakukan sesekali. Misalnya, mengubah foto avatar Anda adalah fitur yang bagus, tetapi mungkin tidak penting untuk setiap pengalaman.

Mengidentifikasi tindakan utama dan sekunder dalam aplikasi Anda akan membantu Anda memprioritaskan pekerjaan aksesibilitas di depan. Kemudian, Anda dapat menggabungkan tindakan ini dengan daftar periksa aksesibilitas untuk melacak kemajuan Anda dan menghindari regresi.

Memasukkan daftar periksa aksesibilitas

Topik aksesibilitas cukup luas, sehingga memiliki daftar periksa bidang-bidang penting untuk dipertimbangkan dapat membantu Anda memastikan Anda memenuhi semua basis Anda.

Ada sejumlah daftar aksesibilitas di luar sana, beberapa contoh industri termasuk:

Daftar Periksa WebAIM WCAG

Pedoman Aksesibilitas Vox

Dengan daftar periksa di tangan, Anda dapat melihat tindakan primer dan sekunder Anda untuk mulai menentukan apa pekerjaan yang masih perlu dilakukan. Anda bisa menjadi sangat taktis tentang proses ini dan bahkan membangun matriks tindakan primer dan sekunder dan menentukan untuk setiap langkah dalam proses tersebut, apakah ada bit aksesibilitas yang hilang.

	<i>Checklist Item 1</i>	<i>Checklist Item 2</i>	<i>Checklist Item 3</i>
<i>Primary Use Case 1</i>	X	X	
<i>Primary Use Case 2</i>		X	
<i>Secondary Use Case 1</i>			
<i>Secondary Use Case 2</i>	X		

Mengevaluasi dengan studi pengguna

Tidak ada yang mengalahkan saat duduk dengan pengguna sebenarnya dan mengamati mereka mencoba menggunakan aplikasi Anda. Jika Anda menyesuaikan aksesibilitas menjadi pengalaman yang ada, proses ini dapat membantu Anda dengan cepat mengidentifikasi area

yang perlu diperbaiki. Dan jika Anda memulai proyek baru, studi pengguna awal dapat membantu Anda menghindari menghabiskan terlalu banyak waktu mengembangkan fitur yang sulit digunakan.

Bertujuan untuk memasukkan umpan balik dari beragam populasi pengguna sebanyak mungkin. Pertimbangkan pengguna yang terutama bernavigasi dengan keyboard, atau mengandalkan teknologi bantu seperti pembaca layar atau kaca pembesar layar.

UX designer

Karena orang cenderung mendesain menggunakan bias mereka sendiri, jika Anda tidak memiliki cacat dan tidak memiliki kolega dengan cacat, Anda mungkin secara tidak sengaja merancang hanya untuk beberapa pengguna Anda. Saat Anda bekerja, tanyakan pada diri sendiri "apa saja tipe pengguna yang mungkin mengandalkan desain ini?" Berikut adalah beberapa teknik yang dapat Anda coba untuk membuat proses Anda lebih inklusif.

- Konten memiliki kontras warna yang cukup.
- Urutan tab ditentukan.
- Kontrol memiliki label yang dapat diakses.
- Ada beberapa cara untuk berinteraksi dengan UI.

Konten memiliki kontras warna yang baik

Tujuan utama sebagian besar situs adalah untuk menyampaikan beberapa informasi kepada pengguna, baik melalui teks tertulis atau gambar. Namun, jika konten ini kontras rendah, mungkin sulit bagi sebagian pengguna (terutama mereka yang memiliki gangguan penglihatan) untuk membaca. Ini dapat memengaruhi pengalaman pengguna mereka secara negatif. Untuk mengatasi masalah ini, bertujuan agar semua teks dan gambar memiliki kontras warna yang cukup.

Kontras diukur dengan membandingkan pencahayaan warna latar depan dan latar belakang. Untuk teks yang lebih kecil (apa pun di bawah 18pt atau 14pt tebal) disarankan rasio minimum 4,5: 1. Untuk teks yang lebih besar, rasio ini dapat disesuaikan menjadi 3: 1.

Pada gambar di bawah ini, teks di sisi kiri memenuhi minimum kontras ini, sedangkan teks di sisi kanan adalah kontras rendah.

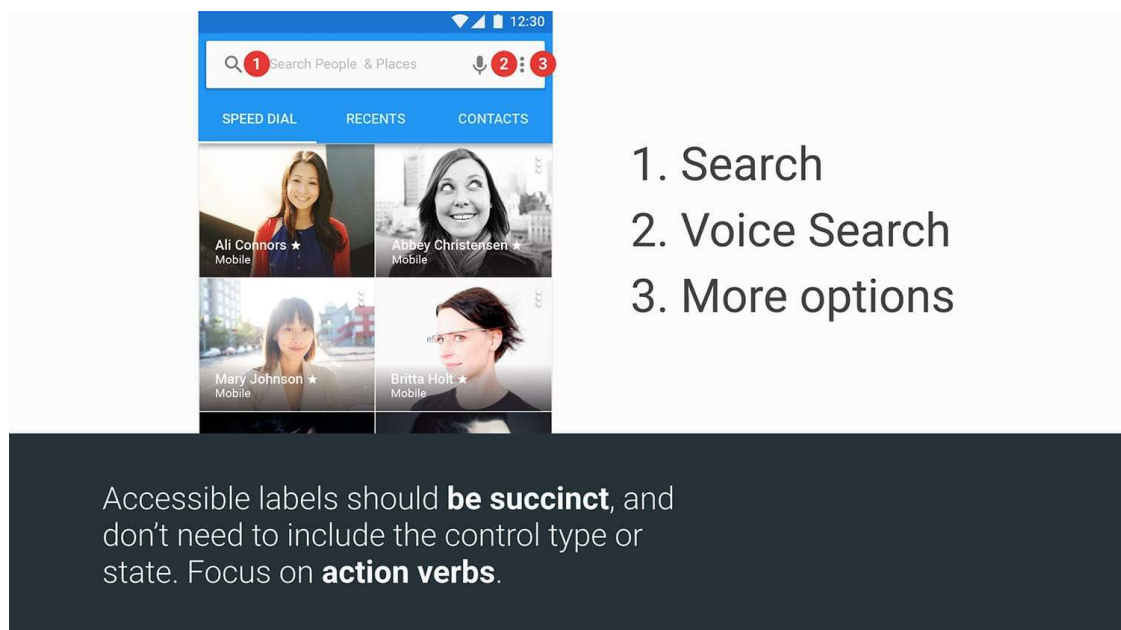
Antarmuka tiruan di atas diberi nomor untuk menampilkan urutan tab. Membuat tiruan seperti ini dapat membantu dengan mengidentifikasi urutan tab yang dimaksud. Ini kemudian dapat dibagikan dengan pengembang dan penguji QA untuk memastikan itu diterapkan dan diuji dengan benar.

Kontrol memiliki label yang dapat diakses

Untuk pengguna teknologi bantuan seperti pembaca layar, label memberikan informasi yang seharusnya hanya visual. Misalnya tombol pencarian yang hanya berupa ikon kaca pembesar dapat memiliki label "Cari" yang dapat diakses untuk membantu mengisi kekurangan visual yang hilang.

Berikut adalah beberapa saran sederhana untuk diikuti ketika mendesain label yang dapat diakses:

- Bersikap singkat - Membosankan mendengarkan deskripsi panjang bisa membosankan.
- Cobalah untuk tidak memasukkan tipe atau status kontrol - Jika kontrol dikodekan dengan benar maka pembaca layar akan mengumumkan ini secara otomatis.
- Fokus pada kata kerja tindakan - Gunakan "cari" bukan "kaca pembesar".



Anda mungkin mempertimbangkan untuk membuat tiruan dengan semua kontrol Anda berlabel. Ini dapat dibagikan dengan tim pengembangan Anda dan tim QA untuk implementasi dan pengujian.

Berbagai cara untuk berinteraksi dan memahami UI

Sangat mudah untuk mengasumsikan bahwa semua pengguna berinteraksi dengan halaman terutama menggunakan mouse. Saat mendesain, pertimbangkan bagaimana seseorang akan berinteraksi dengan kontrol menggunakan keyboard sebagai gantinya.

Rencanakan status fokus Anda! Ini berarti menentukan seperti apa sebuah kontrol ketika pengguna memfokuskannya dengan tab atau menekan tombol panah. Ini berguna untuk membuat negara-negara ini merencanakan lebih awal, daripada mencoba menyemir mereka ke dalam desain nanti.

Akhirnya, untuk setiap titik interaksi, Anda ingin memastikan bahwa pengguna memiliki banyak cara untuk memahami konten. Cobalah untuk tidak menggunakan warna sendirian untuk menyampaikan informasi, karena isyarat halus ini mungkin terlewatkan oleh pengguna dengan kekurangan penglihatan warna. Contoh klasik adalah bidang teks yang tidak valid. Alih-alih hanya menggarisbawahi merah untuk menandakan masalah, pertimbangkan juga menambahkan beberapa teks pembantu. Dengan begitu Anda mencakup lebih banyak pangkalan dan meningkatkan kemungkinan bahwa pengguna akan melihat masalah.

Developer

Peran pengembang adalah tempat manajemen fokus dan semantik bergabung untuk membentuk pengalaman pengguna yang kuat. Berikut adalah beberapa item yang dapat diingat pengembang saat mereka bekerja di situs atau aplikasi mereka:

- Urutan tab logis.
- Fokus dikelola dengan baik dan terlihat.
- Elemen interaktif memiliki dukungan keyboard.
- Peran dan atribut ARIA diterapkan sesuai kebutuhan.
- Elemen diberi label dengan benar.
- Pengujian otomatis.

Urutan tab logis

Elemen asli seperti input, tombol, dan pilih bisa ikut serta dalam urutan tab secara gratis dan secara otomatis dapat fokus dengan keyboard. Tetapi tidak semua elemen menerima perilaku yang sama ini! Secara khusus, elemen generik seperti div, dan span, tidak diikuti dalam urutan tab. Ini berarti jika Anda menggunakan div untuk membuat kontrol interaktif, Anda harus melakukan pekerjaan tambahan agar keyboard dapat diakses.

Dua opsi adalah:

- Berikan kontrol `tabindex = "0"`. Ini setidaknya akan membuatnya menjadi fokus, meskipun Anda mungkin perlu melakukan pekerjaan tambahan untuk menambahkan dukungan untuk penekanan tombol.
- Jika memungkinkan, pertimbangkan untuk menggunakan tombol, bukan div atau rentang untuk kontrol seperti tombol. Elemen tombol asli sangat mudah untuk ditata dan mendapat dukungan keyboard secara gratis.

Mengelola fokus

Saat Anda mengubah konten halaman, penting untuk mengarahkan perhatian pengguna dengan memindahkan fokus. Contoh klasik kapan teknik ini berguna adalah ketika membuka dialog modal. Jika pengguna yang mengandalkan keyboard menekan tombol untuk membuka dialog dan fokus mereka tidak dipindahkan ke elemen dialog, maka satu-satunya tindakan mereka adalah menabrak seluruh situs sampai mereka akhirnya menemukan kontrol baru. Dengan memindahkan fokus ke konten baru segera setelah muncul, Anda dapat meningkatkan efisiensi pengalaman pengguna ini.

Dukungan keyboard untuk elemen interaktif

Jika Anda membuat kontrol khusus seperti carousel atau dropdown, maka Anda perlu melakukan beberapa pekerjaan tambahan untuk menambahkan dukungan keyboard. Panduan Praktek Penulisan ARIA adalah sumber daya yang berguna yang mengidentifikasi berbagai pola UI dan jenis tindakan papan ketik yang diharapkan akan mereka dukung.

2.16 Radio Group

\$

A radio group is a set of checkable buttons, known as radio buttons, where only one button in the set may be in a checked state.

Keyboard Interaction

- When a radio group receives focus:
 - If a radio button is checked, focus is set on the checked button.
 - If none of the radio buttons are checked, focus is set on the first radio button in the group.
- Space: checks the focussed radio button if it is not already checked.
- Right Arrow and Down Arrow: move focus to the next radio button in the group, uncheck the previously focused button, and check the newly focused button. If focus is on the last button, focus moves to the first button.
- Left Arrow and Up Arrow: move focus to the previous radio button in the group, uncheck the previously focused button, and check the newly focused button. If focus is on the first button, focus moves to the last button.

ARIA Authoring Practices. An excellent **cheat sheet** for component accessibility!

Untuk mempelajari lebih lanjut tentang menambahkan dukungan keyboard ke suatu elemen, lihat bagian tabindex keliling di dokumen Fundamentals Aksesibilitas Google.

Peran dan atribut ARIA diterapkan sesuai kebutuhan

Kontrol kustom tidak hanya membutuhkan dukungan keyboard yang tepat, tetapi juga memerlukan semantik yang baik. Bagaimanapun, div, secara semantik, hanyalah wadah pengelompokan generik. Jika Anda menggunakan div sebagai dasar untuk menu dropdown Anda, Anda harus bergantung pada ARIA untuk melapisi semantik tambahan sehingga jenis kontrol dapat disampaikan ke teknologi bantu. Di sini, sekali lagi, Panduan Praktik Penulisan ARIA dapat membantu dengan mengidentifikasi peran, status, dan properti yang harus Anda

gunakan. Sebagai bonus tambahan, banyak penjelasan dalam panduan ARIA juga disertai dengan kode sampel!

Labeling elements

Untuk memberi label pada input asli, Anda dapat menggunakan elemen `<label>` bawaan seperti yang dijelaskan pada MDN. Ini tidak hanya akan membantu Anda membuat kemampuan visual pada layar, tetapi juga memberi input nama yang dapat diakses di pohon aksesibilitas. Nama ini kemudian diambil oleh teknologi bantu (seperti pembaca layar) dan diumumkan kepada pengguna.

Sayangnya `<label>` tidak mendukung pemberian nama yang dapat diakses ke kontrol khusus (seperti yang dibuat menggunakan Elemen Kustom atau dari `div` dan bentang sederhana). Untuk kontrol semacam ini Anda harus menggunakan atribut `aria-label` dan `aria-labelledby` oleh.

Pengujian otomatis

Menjadi malas bisa menjadi hal yang baik, terutama dalam hal pengujian. Sedapat mungkin berusaha mengotomatiskan tes aksesibilitas Anda sehingga Anda tidak perlu melakukan semuanya secara manual. Ada sejumlah alat pengujian industri hebat yang ada saat ini untuk memudahkan dan cepat memeriksa masalah aksesibilitas umum:

kapak, dibuat oleh sistem Deque, tersedia sebagai ekstensi Chrome dan modul Node (baik untuk lingkungan integrasi berkelanjutan). A11ycast singkat ini menjelaskan beberapa cara berbeda untuk memasukkan kapak ke dalam proses pengembangan Anda.

Lighthouse adalah proyek sumber terbuka Google untuk mengaudit kinerja Aplikasi Web Progresif Anda. Selain memeriksa apakah PWA Anda memiliki dukungan untuk hal-hal seperti Pekerja Layanan dan Manifes Aplikasi Web, Lighthouse juga akan menjalankan serangkaian tes praktik terbaik, termasuk tes untuk masalah aksesibilitas.

BAB VIII ANIMASI

Sasaran Pembelajaran

Mahasiswa mampu membuat antarmuka pengguna yang sangat responsif dan interaktif

Kemampuan mahasiswa yang menjadi prasyarat

Mahasiswa sudah menguasai materi Interaksi Manusia Komputer, menguasai tools untuk mendesain, Javascript dan CSS

Keterkaitan bahan pembelajaran dengan pokok bahasan lainnya

Materi ini sebagai dasar untuk mempelajari pokok bahasan lainnya

Manfaat atau pentingnya bahan pembelajaran ini

Membuat aplikasi web/situs dan mobile yang menarik

Animasi adalah bagian penting saat membuat aplikasi web dan situs yang menarik. Pengguna mengharapkan antarmuka pengguna yang sangat responsif dan interaktif. Namun, menganimasikan antarmuka Anda bukanlah hal yang mudah. Apa yang harus dianimasikan, kapan, dan apa jenis nuansa yang dimiliki animasi tersebut?

TL;DR

- Gunakan animasi sebagai cara untuk menambahkan jiwa ke proyek Anda.
- Animasi harus mendukung interaksi pengguna.
- Hati-hati dengan properti yang Anda animasikan; beberapa lebih berat dibanding yang lain.

Memilih hal yang tepat untuk dianimasikan

Animasi yang bagus menambahkan selapis kegembiraan dan rasa keterlibatan pengguna untuk proyek Anda. Anda bisa menganimasikan hampir semuanya, apakah itu lebar, ketinggian, posisi, warna, atau latar belakang, namun Anda harus menyadari potensi bottleneck kinerja dan bagaimana animasi memengaruhi personalitas aplikasi Anda. Animasi yang tersendat atau salah pilih bisa berdampak negatif terhadap pengalaman pengguna, sehingga animasi harus berkinerja baik dan tepat.

Gunakan animasi untuk mendukung interaksi

Jangan menganimasikan sesuatu hanya karena Anda bisa; itu hanya akan membuat pengguna jengkel dan mengganggu. Namun, gunakan animasi yang ditempatkan secara tepat untuk memperkuat interaksi pengguna. Jika mereka mengetuk ikon menu, menggesek untuk menampilkan panel samping navigasi, atau mengetuk tombol, gunakan cahaya lembut atau efek

memantul untuk menerima interaksi. Hindari animasi yang mengganggu atau menghalangi aktivitas pengguna secara tidak perlu.

Hindari menganimasikan properti yang berat

Satu-satunya hal yang lebih buruk daripada animasi yang salah tempat adalah yang menyebabkan laman menjadi tersendat. Tipe animasi ini membuat pengguna merasa frustrasi dan tak senang, dan berharap Anda tidak menganimasikannya sama sekali.

Beberapa properti lebih sulit untuk diubah dibanding yang lain, dan hal inilah yang lebih mungkin membuat animasi tersendat. Jadi, misalnya, mengubah `box-shadow` sebuah elemen memerlukan operasi pewarnaan yang jauh lebih sulit dibandingkan mengubah, katakan, warna teksnya. Demikian pula, mengubah `width` sebuah elemen cenderung lebih sulit dibandingkan mengubah `transform`.

Anda bisa membaca selengkapnya mengenai perhitungan kinerja animasi dalam panduan [Animasi dan Kinerja](#), namun jika Anda menginginkan TL;DR, tetaplah konsisten pada ubahan `transform` dan `opacity`, serta menggunakan `will-change`. Jika Anda ingin mengetahui secara pasti pekerjaan mana yang dipicu karena menganimasikan properti yang diberikan, lihat [Pemicu CSS](#).

8.1 Animasi CSS Versus JavaScript

Ada dua cara utama untuk membuat animasi di web: dengan CSS dan dengan JavaScript. Mana yang Anda pilih sangat tergantung pada dependensi lain dari proyek, dan jenis efek yang ingin coba dicapai.

TL;DR

- Gunakan animasi CSS untuk transisi "satu-kali" yang lebih sederhana, seperti mengubah status elemen UI.
- Gunakan animasi JavaScript ketika Anda ingin menggunakan efek lanjutan seperti memantul, berhenti, jeda, memutar mundur atau memperlambat.
- Jika Anda memilih untuk membuat animasi dengan JavaScript, gunakan Web Animations API atau kerangka kerja modern yang sesuai bagi Anda.

Kebanyakan animasi dasar bisa dibuat dengan CSS atau JavaScript, namun besaran tenaga dan waktu bisa berbeda (lihat juga [Kinerja CSS vs JavaScript](#)). Masing-masing memiliki kelebihan dan kekurangan, tetapi ini bisa dijadikan panduan yang bagus:

- **Gunakan CSS ketika Anda memiliki status mandiri yang lebih kecil untuk elemen UI.** Transisi dan animasi CSS ideal untuk menampilkan menu navigasi dari samping, atau menampilkan keterangan alat. Anda mungkin akhirnya menggunakan JavaScript untuk mengontrol status, namun animasinya akan ada di CSS.

- **Gunakan JavaScript ketika Anda membutuhkan kontrol yang signifikan atas animasi Anda.** Web Animations API adalah pendekatan berbasis standar, saat ini tersedia di Chrome dan Opera. Ini memberikan objek nyata, ideal untuk aplikasi berorientasi objek yang kompleks. JavaScript juga berguna ketika Anda perlu berhenti, jeda, melambat atau membalikkan.
- **Gunakan `requestAnimationFrame` secara langsung ketika Anda ingin mengatur seluruh kejadian dengan tangan.** Ini adalah pendekatan JavaScript lanjutan, namun bisa bermanfaat jika Anda membangun sebuah game atau menggambar pada kanvas HTML.

Atau, jika Anda sudah menggunakan kerangka kerja JavaScript yang dilengkapi fungsionalitas animasi, seperti melalui metode jQuery `.animate()` atau [GreenSock TweenMax](#), maka Anda mungkin merasa lebih nyaman dengan tetap menggunakannya untuk animasi Anda.

Menganimasikan dengan CSS

Menganimasikan dengan CSS adalah cara paling sederhana untuk menggerakkan sesuatu di layar. Pendekatan ini digambarkan sebagai *deklaratif*, karena Anda menentukan apa yang akan terjadi.

Di bawah ini adalah beberapa CSS yang memindahkan elemen 100 px pada sumbu X dan Y. Hal ini dilakukan dengan menggunakan transisi CSS yang disetel 500 ms. Ketika kelas `move` ditambahkan, nilai `transform` berubah dan transisi dimulai.

```
.box {
  -webkit-transform: translate(0, 0);
  -webkit-transition: -webkit-transform 500ms;

  transform: translate(0, 0);
  transition: transform 500ms;
}

.box.move {
  -webkit-transform: translate(100px, 100px);
  transform: translate(100px, 100px);
}
```

[Cobalah](#)

Selain durasi transisi, ada beberapa pilihan untuk *easing*, yaitu bagaimana animasi terasa. Untuk informasi selengkapnya tentang easing, lihat panduan [Dasar-Dasar Easing](#).

Jika, seperti dalam cuplikan di atas, Anda membuat kelas CSS yang terpisah untuk mengelola animasi, maka Anda bisa menggunakan JavaScript untuk menghidupkan dan mematikan setiap animasi:

```
box.classList.add('move');
```

Melakukan hal ini memberikan keseimbangan yang baik bagi aplikasi Anda. Anda bisa fokus dalam mengelola status dengan JavaScript, dan hanya menyetel kelas yang sesuai pada elemen sasaran, mempercayakan browser untuk menangani animasi. Jika menggunakan cara ini, Anda bisa mendengarkan kejadian `transitionend` pada elemen, namun hanya jika Anda mampu melepaskan dukungan pada versi Internet Explorer lama; versi 10 adalah versi pertama yang mendukung kejadian ini. Semua browser lain telah lama mendukung kejadian ini.

JavaScript yang diperlukan untuk mendengarkan akhir transisi terlihat seperti ini:

```
var box = document.querySelector('.box');
box.addEventListener('transitionend', onTransitionEnd, false);

function onTransitionEnd() {
  // Handle the transition finishing.
}
```

Selain menggunakan transisi CSS, Anda juga bisa menggunakan animasi CSS, yang memberikan Anda kontrol lebih banyak terhadap keyframe, durasi dan iterasi animasi individu.

Note: Jika Anda baru dalam dunia animasi, keyframe adalah istilah lama dari animasi yang digambar tangan. Animator akan membuat bingkai khusus untuk bagian dari tindakan, disebut bingkai kunci, yang akan merekam beberapa hal seperti bagian paling ekstrem dari beberapa gerakan, dan kemudian mereka akan menggambar semua bingkai secara tersendiri di antara keyframe. Kita menjalankan proses yang sama saat ini dengan animasi CSS, kita menginstruksikan browser tentang nilai yang harus dimiliki properti CSS pada titik tertentu, dan keyframe mengisi celahnya.

Misalnya, Anda bisa menganimasikan kotak dengan cara yang sama seperti transisi, namun menganimasikannya tanpa interaksi pengguna seperti klik, dan dengan pengulangan tak terbatas. Anda juga bisa mengubah beberapa properti sekaligus:

```
/**
 * This is a simplified version without
 * vendor prefixes. With them included
 * (which you will need), things get far
 * more verbose!
 */
.box {
  /* Choose the animation */
  animation-name: movingBox;

  /* The animation's duration */
}
```

```

animation-duration: 1300ms;

/* The number of times we want
   the animation to run */
animation-iteration-count: infinite;

/* Causes the animation to reverse
   on every odd iteration */
animation-direction: alternate;
}

@keyframes movingBox {
  0% {
    transform: translate(0, 0);
    opacity: 0.3;
  }

  25% {
    opacity: 0.9;
  }

  50% {
    transform: translate(100px, 100px);
    opacity: 0.2;
  }

  100% {
    transform: translate(30px, 30px);
    opacity: 0.8;
  }
}

```

Cobalah

Dengan animasi CSS, Anda menentukan animasi secara independen dari elemen target, dan menggunakan properti nama animasi untuk memilih animasi yang diperlukan.

Animasi CSS tetap diberi awalan oleh vendor, dengan awalan `-webkit-` yang digunakan di Safari, Safari Mobile, dan Android. Chrome, Opera, Internet Explorer dan Firefox semua diluncurkan tanpa awalan. Banyak alat yang bisa membantu Anda membuat versi awalan dari CSS yang Anda butuhkan, sehingga Anda bisa menulis versi tanpa awalan dalam file sumber Anda.

Menganimasikan dengan JavaScript dan Web Animations API

Membuat animasi dengan JavaScript, jika dibandingkan, lebih kompleks daripada menulis transisi atau animasi CSS, tapi memberikan lebih banyak kendali kepada developer. Anda bisa menggunakan [Web Animations API](#) untuk menganimasikan properti CSS tertentu atau membangun efek objek yang bisa disusun.

Animasi JavaScript *sangat penting*, ketika Anda menulisnya secara inline sebagai bagian dari kode Anda. Anda juga bisa membungkusnya dalam objek lain. Berikut adalah JavaScript yang harus Anda tulis untuk membuat ulang transisi CSS yang sudah dijelaskan sebelumnya:

```
var target = document.querySelector('.box');
var player = target.animate([
  {transform: 'translate(0)'},
  {transform: 'translate(100px, 100px)'}
], 500);
player.addEventListener('finish', function() {
  target.style.transform = 'translate(100px, 100px)';
});
```

Secara default, Web Animasi hanya memodifikasi presentasi dari elemen. Jika Anda ingin agar objek tetap berada di lokasi pindahannya, maka Anda harus mengubah gaya dasarnya ketika animasi selesai, seperti contoh kami.

[Cobalah](#)

Web Animations API adalah standar baru dari W3C. Didukung secara native di Chrome dan Opera, dan dalam [proses development aktif untuk Firefox](#). Untuk browser modern lainnya, [tersedia polyfill](#).

Dengan animasi JavaScript, Anda memiliki kontrol penuh dari gaya elemen ini dalam setiap langkahnya. Ini berarti Anda bisa memperlambat animasi, melakukan jeda, menghentikan, membalikkan, dan memanipulasi elemen animasi sesuai keinginan Anda. Hal ini sangat berguna jika membangun aplikasi berorientasi objek yang kompleks, karena Anda bisa membungkus perilaku dengan baik.

8.2 Dasar-Dasar Easing

Di dunia ini tidak ada yang bergerak secara linear dari satu titik ke titik lainnya. Dalam kenyataannya, sesuatu cenderung mengalami percepatan atau perlambatan ketika mereka bergerak. Otak kita sudah diprogram untuk mengharapkan jenis gerakan ini, sehingga ketika membuat animasi, Anda harus menggunakannya untuk keuntungan Anda. Gerakan alami membuat pengguna merasa lebih nyaman dengan aplikasi Anda, yang pada akhirnya memberikan pengalaman yang lebih baik secara keseluruhan.

TL;DR

- Easing membuat animasi Anda terasa lebih alami.
- Pilih animasi ease-out untuk elemen UI.
- Hindari animasi ease-in atau ease-in-out kecuali Anda bisa membuatnya berdurasi pendek; mereka terasa lambat bagi pengguna akhir.

Dalam animasi klasik, istilah untuk gerak yang dimulai perlahan kemudian dipercepat adalah "slow in," dan gerakan yang dimulai dengan cepat kemudian berkurang kecepatannya adalah "slow out." Istilah yang paling umum digunakan di web untuk animasi ini adalah "ease in" dan "ease out". Kadang-kadang keduanya digabungkan, yang disebut "ease in out." Kemudian, easing adalah proses untuk membuat animasi lebih gampang untuk diucapkan.

Kata kunci easing

Transisi dan animasi CSS memperbolehkan Anda [memilih jenis easing yang ingin Anda gunakan untuk animasi Anda](#). Anda bisa menggunakan kata kunci yang memengaruhi easing (atau kadang-kadang disebut *timing*) dari animasi tersebut. Anda juga bisa [sepenuhnya menyesuaikan easing Anda](#), yang memberi Anda lebih banyak kebebasan untuk mengekspresikan kepribadian aplikasi.

Berikut adalah beberapa kata kunci yang bisa Anda gunakan dalam CSS:

- `linear`
- `ease-in`
- `ease-out`
- `ease-in-out`

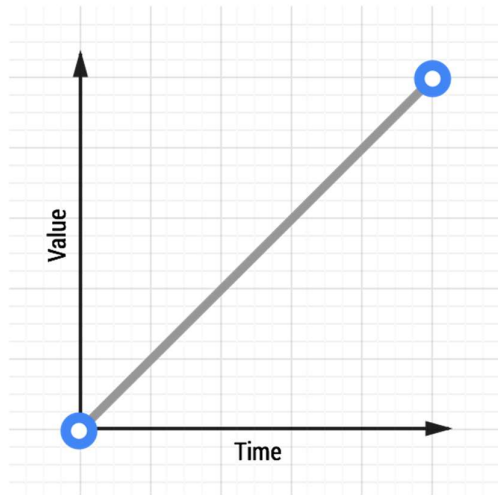
Sumber: [Transisi CSS, W3C](#)

Anda juga bisa menggunakan kata kunci `steps`, yang memungkinkan Anda untuk membuat transisi yang memiliki langkah-langkah terpisah, namun kata kunci yang tercantum di atas adalah yang paling berguna untuk membuat animasi yang terasa alami.

Animasi linear

Animasi tanpa menggunakan easing disebut sebagai **linear**. Ketika waktu berjalan, nilai juga mengalami pertambahan dalam jumlah yang setara. Dengan gerakan linear, sesuatu cenderung terasa kaku dan tidak wajar, dan hal ini bisa membuat janggal bagi pengguna. Pada intinya, Anda harus menghindari gerakan linear.

Apakah Anda sedang melakukan pengkodean animasi menggunakan CSS atau JavaScript, Anda akan mendapati bahwa selalu ada pilihan untuk gerakan linear. Grafik transisi linear terlihat sebagai berikut:

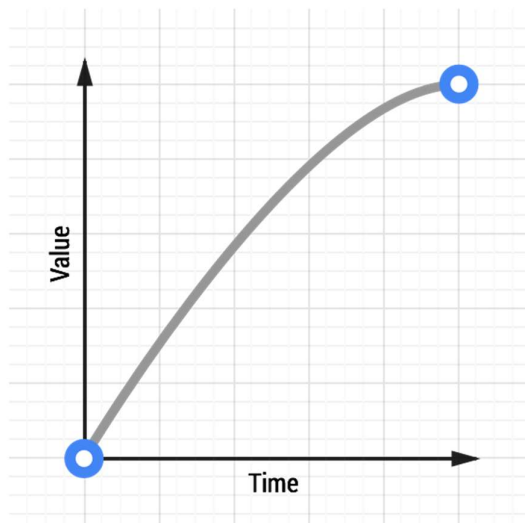


[Lihat animasi linear](#)

Untuk mendapatkan efek di atas dengan CSS, kodenya akan terlihat seperti berikut ini:

```
transition: transform 500ms linear;
```

Animasi ease-out



Easing out menyebabkan animasi dimulai lebih cepat dari yang linear, dan juga melambat di akhir.

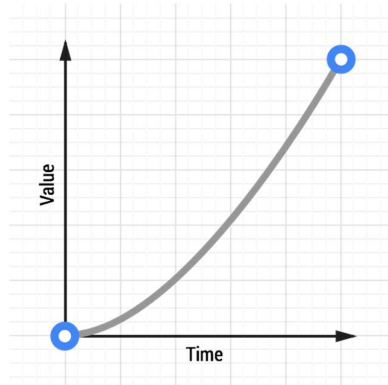
Easing out biasanya paling cocok digunakan untuk pekerjaan antarmuka pengguna, karena awal yang cepat dari animasi Anda memberikan kesan responsif, dan masih ada ruang untuk perlambatan alami di bagian akhir.

[Lihat animasi ease-out](#)

Ada banyak cara untuk mendapatkan efek ease out, tapi yang paling sederhana adalah kata kunci ease-out di CSS:

```
transition: transform 500ms ease-out;
```

Animasi ease-in



Animasi ease-in mulai dengan perlahan dan berakhir cepat, yang merupakan kebalikan dari animasi ease-out.

Jenis animasi ini seperti sebuah batu berat yang jatuh, batu itu awalnya jatuh perlahan-lahan lalu menghunjam tanah secara cepat dengan dentuman mematikan.

Namun, dari sudut pandang interaksi, ease-in terasa sedikit janggal karena akhir yang tiba-tiba; sesuatu yang bergerak di dunia nyata mengalami perlambatan, bukannya langsung berhenti tiba-tiba. Ease-in juga memiliki efek yang merugikan yaitu terasa lambat ketika dimulai, yang berdampak negatif terhadap persepsi daya respons dalam situs atau aplikasi Anda.

[Lihat animasi ease-in](#)

Untuk menggunakan animasi ease-in, sama seperti animasi ease-out dan linear, Anda bisa menggunakan kata kuncinya:

```
transition: transform 500ms ease-in;
```

Animasi ease-in-out

Easing-in dan easing-out mirip dengan mobil yang berakselerasi lalu melambat, dan jika digunakan dengan bijak, bisa memberikan efek yang lebih dramatis dari sekadar easing-out.