```
div x-data="{ open: false }" x-trap.noreturn="open">
    <input type="search" placeholder="search for something" />

    <div x-show="open">
        Search results

        <button @click="open = false">Close</button>
    </div>
</div>
```

$fokus

Plugin ini menawarkan banyak utility kecil untuk mengatur fokus di dalam halaman. Utilitas ini diekspos melalui magic `$focus`

Property	Deskripsi
focus(el)	Fokuskan elemen yang dilewati (menangani gangguan secara internal menggunakan nexttick dan lain-lain).
focusable(el)	Deteksi weather atau bukan elemen yang memiliki kemampuan fokus
focusables()	Dapatkan semua “focusable” element pada elemen saat ini
focused()	Dapatkan elemen yang difokus saat ini pada halaman
lastFocused()	Dapatkan elemen focus terakhir di halaman
within(el)	Tentukan elemen untuk cakupan magic `$focus (`elemen saat ini secara default)
first()	Fokuskan pada focusable elemen pertama
last()	Fokuskan pada focusable elemen terakhir
next()	Fokuskan pada focusable elemen selanjutnya
previous()	Fokuskan pada focusable elemen sebelumnya
noscroll()	Mencegah scrolling ke elemen yang di fokus

wrap()	ketika mengambil “selanjutnya” atau “sebelumnya” menggunakan “web round” (kecuali mengembalikan elemen pertama jika terdapat elemen “selanjutnya” pada elemen terakhir
getFirst()	Mengambil elemen pertama yang focusable
getLast()	Mengambil elemen terakhir yang focusable
getNext()	Mengambil elemen selanjutnya yang focusable
getPrevious()	Mengambil elemen sebelumnya yang focusable

Ayo lanjut dengan beberapa contoh dari penggunaan utility ini. Contoh di bawah memungkinkan pengguna untuk mengendalikan fokus di dalam kelompok tombol menggunakan arraw key (tombol panah). Anda dapat mengujinya dengan klik pada tombol kemudian menggunakan tombol panah untuk memindah fokus di sekitar.

```
<div
    @keydown.right="$focus.next()"
    @keydown.left="$focus.previous()"
>
    <button>First</button>
    <button>Second</button>
    <button>Third</button>
</div>
```

Perhatikan jika tombol terakhir yang difokus, tekan “panah kanan” tidak akan melakukan apapun. Ayo tambahkan method `.wrap()` sehingga fokus membungkus semuanya.

```
<div
    @keydown.right="$focus.wrap().next()"
    @keydown.left="$focus.wrap().previous()"
>
    <button>First</button>
    <button>Second</button>
    <button>Third</button>
</div>
```

Sekarang, ayo tambahkan 2 tombol, 1 fokus ke elemen pertama dari kelompok tombol dan fokus lain ke elemen terakhir:

```
<button @click="$focus.within($refs.buttons).first()">Focus "First"</button>
<button @click="$focus.within($refs.buttons).last()">Focus "Last"</button>

<div
    x-ref="buttons"
    @keydown.right="$focus.wrap().next()"
    @keydown.left="$focus.wrap().previous()"
>
    <button>First</button>
    <button>Second</button>
    <button>Third</button>
</div
```

Perhatikan bahwa kita butuh menambahkan method `.within()` untuk setiap tombol jadi `$focus` tahu lingkupnya sendiri ke elemen yang berbeda (`div` yang membungkus tombol).

Collapse Plugin

Collapse pada Alpine memungkinkan anda mengembangkan dan meruntuhkan elemen menggunakan transisi yang mulus.
Karena perilaku dan implementasi ini berbeda dari standar sistem transisi Alpine, fungsionalitas ini telah dibuat ke dalam plugin yang terdedikasi/dikhususkan.

Installation

Anda dapat menggunakan plugin ini baik dari tag `<script>` atau menginstalnya melalui NPM:

Via CDN

Anda dapat memasukkan CDN build dari plugin ini sebagai tag `<script>`, pastikan untuk memasukkannya sebelum file inti Alpine JS.

```
<!-- Alpine Plugins -->
<script defer
src="https://unpkg.com/@alpinejs/collapse@3.x.x/dist/cdn.min.js"></script>

<!-- Alpine Core -->
<script defer src="https://unpkg.com/alpinejs@3.x.x/dist/cdn.min.js"></script>
```

Via NPM

Anda dapat install Collapse dari NPM untuk digunakan bundel anda:

```
npm install @alpinejs/collapse
```

kemudian inisialisasi dari bundel anda:

```
import Alpine from 'alpinejs'
import collapse from '@alpinejs/collapse'

Alpine.plugin(collapse)

...
```

x-collapse

API utama untuk penggunaan plugin ini adalah direktif `x-collapse`

`x-collapse` hanya ada pada sebuah elemen yang telah memiliki directive `x-show`. Ketika ditambahkan sebuah elemen `x-show`, `x-collapse` akan secara mulus “mengembang” dan “meruntuhkan” elemen ketika visibilitasnya di toggle melalui animasi properti height.

Sebagai contoh:

```
<div x-data="{ expanded: false }">
    <button @click="expanded = ! expanded">Toggle Content</button>

    <p x-show="expanded" x-collapse>

        ...
    </p>
</div>
```

Modifier

.duration

Anda dapat menyesuaikan durasi dari collapse/expand dengan menambahkan modifier `.duration`:

```
<div x-data="{ expanded: false }">
    <button @click="expanded = ! expanded">Toggle Content</button>
```

```
    <p x-show="expanded" x-collapse.duration.1000ms>
        ...
    </p>
</div>
```

.min

Secara bawaan, state “collapsed” mengatur tinggi dari elemen ke `0px` dan juga mengatur `display: none;`

Terkadang, ini berguna untuk “memotong” sebuah elemen daripada menyembunyikannya sepenuhnya. Dengan menggunakan modifier `.min` Anda dapat mengatur tinggi minimal dari state “collapsed”nya `x-collapse` Sebagai contoh:

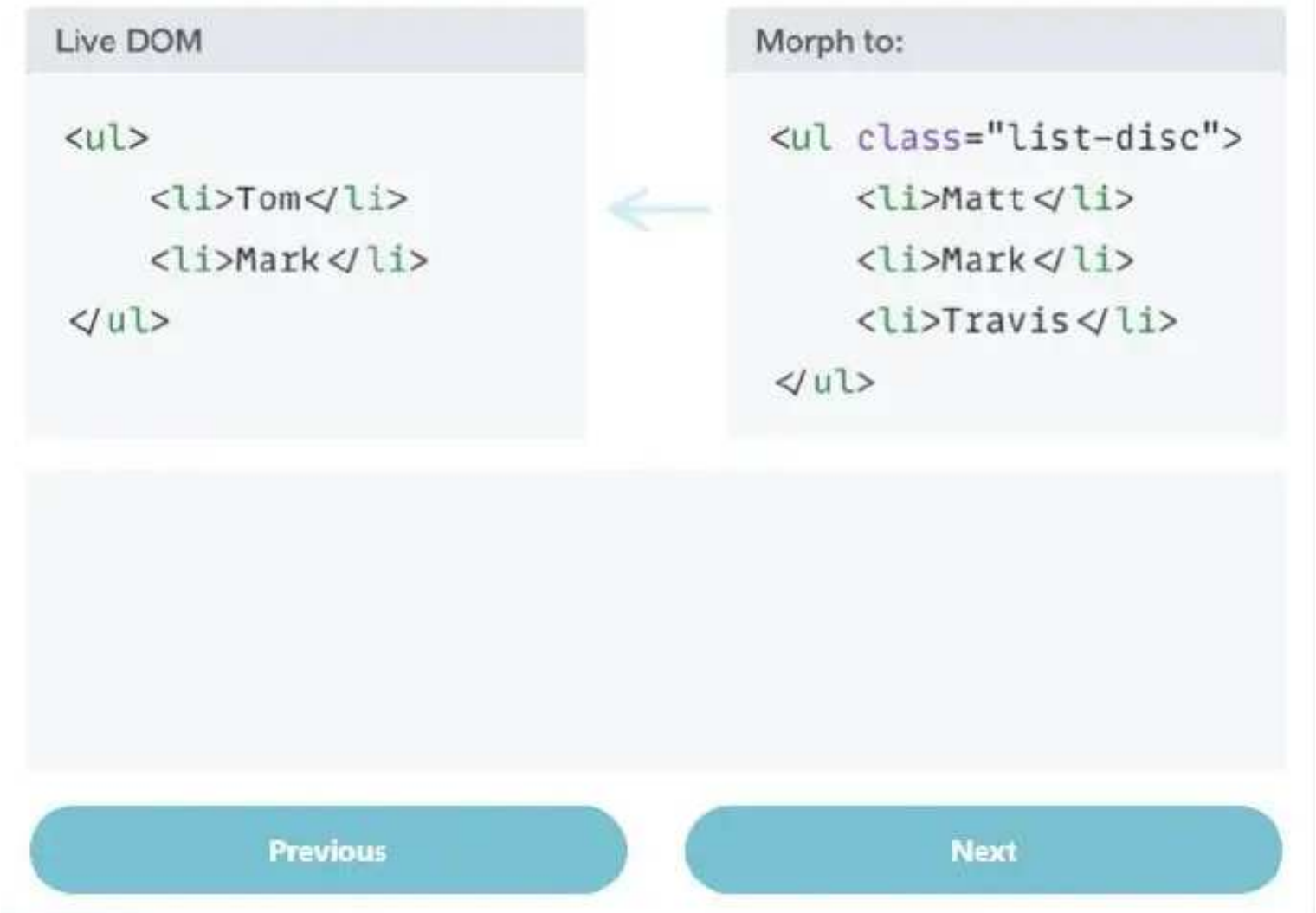
```
<div x-data="{ expanded: false }">
    <button @click="expanded = ! expanded">Toggle Content</button>

    <p x-show="expanded" x-collapse.min.50px>
        ...
    </p>
</div>
```

Morph Plugin

Plugin Morph memungkinkan anda untuk “mengubah” elemen pada halaman menjadi template html yang telah disediakan, sambil mempertahankan peramban (browser) atau state Alpine apapun di dalam elemen yang “diubah”.

Ini berguna untuk memperbarui HTML dari sebuah request peladen tanpa kehilangan state pada halaman Alpine. Utility seperti inti adalah ini dari Framework s full-stack seperti Laravel Livewire dan Phoenix Live View.

Cara terbaik untuk memahami tujuan ini adalah dengan mengikuti Interactive visual berikut. Mari kita coba.



Installation

anda dapat menggunakan plugin ini baik menyertakannya melalui script `<script>` atau menginstalnya melalui NPM.

Via CDN

Anda dapat memmasukkan CDN build plugin ini sebagai sebuah tag `<script>`, pastikan memasukkannya sebelum file inti Alpine JS.

```
<!-- Alpine Plugins -->
<script defer
src="https://unpkg.com/@alpinejs/morph@3.x.x/dist/cdn.min.js"></script>

<!-- Alpine Core -->
<script defer src="https://unpkg.com/alpinejs@3.x.x/dist/cdn.min.js"></script>
```

Via NPM

Anda dapat menginstal Morph dari NPM untuk digunakan di bundel anda

```
npm install @alpinejs/morph
```

lalu inisialisasi dari bundel anda.

```
import Alpine from 'alpinejs'
import morph from '@alpinejs/morph'

window.Alpine = Alpine
Alpine.plugin(morph)
```

...

Alpine.morph()

`Alpine.morph(el, newHtml)` memungkinkan anda secara integratif mengubah sebuah node DOM berdasarkan pada HTML yang dioper. `Alpine.morph(el, newHtml)` menerima parameter berikut:

Dom Element pada halaman

Parameter	Deskripsi
el	DOM element pada halaman
newHtml	sebuah string dari HTML untuk digunakan sebagai template yang mengubah elemen
options (optional)	sebuah objek opsional digunakan utamanya untuk injecting life cycle hooks

Ini adalah contoh dari `Alpine.morph()` untuk memperbarui sebuah komponen Alpine dengan html baru:(pada aplikasi nyata, html baru ini akan datang dari peladen)

```
<div x-data="{ message: 'Change me, then press the button!' }">
    <input type="text" x-model="message">
    <span x-text="message"></span>
</div>

<button>Run Morph</button>

<script>
    document.querySelector('button').addEventListener('click', () => {
        let el = document.querySelector('div')

        Alpine.morph(el, `
            <div x-data="{ message: 'Change me, then press the button!' }">
```

```
                <h2>See how new elements have been added</h2>

                <input type="text" x-model="message">
                <span x-text="message"></span>

                <h2>but the state of this component hasn't changed?

Magical.</h2></div>
        `)
    })
</script>
```

Lifecycle Hooks

Plugin “Morph” bekerja dengan membandingkan 2 DOM tree, Live element dan mengoper nya ke dalam HTML.

Morph menyusuri 2 DOM tree secara simultan dan membandingkan setiap node dan childrennya. Jika ditemukan perbedaan, Morph akan “menggabungkan” (perubahan) ke DOM tree saat ini untuk pencocokan dengan HTML yang di oper ke DOM tree.

Sementara alogaritma defaut sangat mampu. Ada kasus di mana anda ingin hooks ke dalam life cycle dan dan mengamati atau mengubah perilaku.

Sebelum kita melompat ke lifecycle hoos yang tersedia, ayo pertama daftar semua parameter potensial yang mereka terima dan penjelasan dan masing-masing:

`el`	ini adalah selalu aktual, saat ini, dan elemen pada halaman akan “ditambal“ (diubah oleh Morph)
`toEl`	Ini adalah sebuah “template elemen”. Ini merupakan elemen sementara yang mencerminkan apa yang sedang `el` akan tambal. `el` tidak secara aktual tampil dihalaman dan hanya digunakan untuk tujuan referensi.
`childrenOnly()`	Fungsi ini dapat dipanggil di dalam hook untuk memberitahu Morph untuk melewati elemen saat ini dan hanya “menambal” (patch) dan elemen childrennya.

`skip()`	Sebuah fungsi yang dapat ketika dipanggil dalam “hook” akan melewati pembandingan atau penambalannya sendiri dan children dari Element saat ini.

Ini tersedia lifecycle hooks (dioper sebagai parameter ketiga `Alpine.morph(..., options)`):

Option	Deskripsi
`updating(el, toEl, childrenOnly, skip)`	dipanggil sebelum patching `el` dengan perbandingan `toEl`
`updated(el, toEl)`	dipanggil setelah Morph di patch `el`
`removing(el, skip)`	dipanggil sebelum Morph menghapus sebuah elemen dari live DOM
`removed(el)`	Dipanggil setelah Morph menghapus sebuah elemen dari live DOM
`adding(el, skip)`	dipanggil sebelum penambahan elemen baru
`added(el)`	dipanggil setelah penambahan elemen baru ke live DOM tree
`key(el)`	sebuah fungsi re-useable untuk menentukan cara “keys” Morph elemen dalam DOM tree sebelum di bandingkan atau tambal.
`lookahead`	sebuah nilai boolean yang memberitahu Morph untuk mengaktifkan sebuah ekstraksi fitur dalam algoritma yang “melihat kedepan” untuk memastikan sebuah DOM elemen seharusnya dihapus alih-alih hanya “dipindahkan” ke sibling nanti.

.

Ini adalah kode dari semua lifecycle untuk referensi yang lebih konkrit:

```
Alpine.morph(el, newHtml, {
  updating(el, toEl, childrenOnly, skip) {
    //
  },

  updated(el, toEl) {
    //
  },

  removing(el, skip) {
    //
  },

  removed(el) {
    //
  },

  adding(el, skip) {
    //
  },

  added(el) {
    //
  },

  key(el) {
    // By default Alpine uses the `key=""` HTML attribute.
    return el.id;
  },

  lookahead: true, // Default: false
});
```

Keys

Utility pembeda-DOM seperti Morph mencoba hal terbaik mereka untuk secara akurat “mengubah” original DOM ke dalam html baru. Meskipun ada kasus di mana tidak mungkin untuk menentukan jika sebuah elemen harus diubah atau dihapus seluruhnya.

Karena batasan ini, Morph memiliki sebuah sistem “key” yang memungkinkan developer untuk memaksa elemen tertentu ditahan daripada menggantinya.

Penggunaan umum kasus ini Ini adalah sebuah daftar sibling di dalam sebuah pengulangan. Dibawah ini adalah contoh alasan key terkadang dibutuhkan.

```
<!-- "Live" Dom on the page: -->
<ul>
    <li>Mark</li>
    <li>Tom</li>
    <li>Travis</li>
</ul>

<!-- New HTML to "morph to": -->
<ul>
    <li>Travis</li>

    <li>Mark</li>
    <li>Tom</li>
</ul>
```

Mengingat situasi di atas, Morph tidak memiliki cara untuk tahu bahwa node “Travis” telah dihapus dari DOM tree. Pikirkan bahwa “Mark” telah berubah ke “Travis” dan “Travis” berubah ke “Tom”.

Ini bukan yang sebenarnya kita inginkan, kita ingin untuk menyimpan elemen original alih-alih mengubahnya, MEMINDAHKAN mereka ke dalam `<ul>`.

Dengan menambahkan keys di setiap node kita dapat mencapai ini:

```
<!-- "Live" Dom on the page: -->
<ul>
    <li key="1">Mark</li>
    <li key="2">Tom</li>
    <li key="3">Travis</li>
</ul>

<!-- New HTML to "morph to": -->
<ul>
    <li key="3">Travis</li>

    <li key="1">Mark</li>
    <li key="2">Tom</li>
</ul>
```

Sekarang terdapat “keys” di `<li>`, Morph mencocokkan mereka ke-2 DOM tree dan memindah mereka.

Anda dapat melakukan konfigurasi Morph untuk pertimbangkan sebuah “key” dengan konfigurasi opsional `key:`.

Advanced

Reactivity

Alpine merupakan “reaktif” dalam arti ketika anda mengubah bagian dari data, semua yang bergantung pada data tersebut secara otomatis

bereaksi pada perubahan tersebut.
Setiap reaktivitas kecil yang yang terjadi di Alpine, terjadi karena 2 fungsi reaktif penting di inti Alpine: `Alpine.reactive()` dan `Alpine.effect()`

**Alpine menggunakan engine reactivity-nya VueJS di belakang layar untuk menyediakan fungsi ini.*

Memahami 2 fungsi yang diatas memberi anda kekuatan super sebagai developer Alpine tapi juga sebagai seorang web Developer secara umum.

Alpine.reactive()

Mari kita lihat pada `Alpine.reactive()`. Fungsi ini menerima sebuah object JavaScript sebagai parameter dan mengembalikan sebuah versi “reaktif” dari objek tersebut. Sebagai contoh:

```
let data = { count: 1 };

let reactiveData = Alpine.reactive(data);
```

Di belakang layar, ketika `Alpine.reactive` menerima `data`. Data tersebut akan dibungkus dalam sebuah proxy custom JavaScript.

Sebuah proxy merupakan objek yang spesial di JavaScript yang dapat mencegat “get” dan “set” untuk memanggil sebuah object JavaScript.

Pada nilai nominal, `reactiveData` harusnya berperilaku sama seperti `data` Sebagai contoh:

```
onsole.log(data.count); // 1
console.log(reactiveData.count); // 1

reactiveData.count = 2;

console.log(data.count); // 2
console.log(reactiveData.count); // 2
```

Apa yang Anda lihat disini ini terjadi karena `reactiveData` ialah pembungkus kecil disekitar `data`, setiap upaya untuk get atau set sebuah properti akan berperilaku tepat seperti ini seperti jika anda berinteraksi dengan `data` secara langsung.
Perbedaan utama di sini adalah kapanpun anda memodifikasi dan mengambil sebuah value dari `reactiveData`, Alpine menyadarinya dan akan mengeksekusi logic lain manapun yang bergantung pada data itu.

`Alpine.reactive` hanya bagian pertama dari cerita. `Alpine.effect` adalah bagian yang lain mari kita dalami.

Alpine.effect()

`Alpine.effect` menerima sebuah fungsi callback tunggal. Begitu `Alpine.effect` dipanggil, Dia akan menjalankan fungsi yang disediakan, tapi secara aktif melihat interaksi manapun dengan data reaktif. Jika terdeteksi sebuah interaksi (sebuah get atau set dari proxy reaktif) `Alpine.effect` akan menjaganya dan memastikan callback dijalankan ulang jika ada data reaktif apapun yang berubah di masa depan. Sebagai contoh:

```
let data = Alpine.reactive({ count: 1 });

Alpine.effect(() => {
  console.log(data.count);
});
```

Ketika kode ini dijalankan, "1" akan di log ke console. Kapanpun `data.count` berubah nilai akan di log ke console lagi.

Ini merupakan mekanisme yang membuka kunci semua reactivity pada inti dari Alpine.

Untuk menyambungkan ke titik selanjutnya. Ayo perhatikan sebuah component "counter" sederhana tanpa menggunakan syntax Alpine sama sekali, hanya menggunakan `Alpine.reactive` dan `Alpine.effect`

```
<button>Increment</button>;
```

```
Count: <span></span>;
```

```
let button = document.querySelector("button");

let span = document.querySelector("span");
let data = Alpine.reactive({ count: 1 });

Alpine.effect(() => {
  span.textContent = data.count;
});

button.addEventListener("click", () => {
  data.count = data.count + 1;
});
```

Seperti yang dapat Anda lihat, anda dapat membuat data reaktif apapun. Dan dapat juga dapat membungkus kedalam fungsionalitas manapun didalam `Alpine.effect`

Kombinasi ini membuka kunci paradigma programming yang sangat powerfull untuk web development. Berlari liar dan bebas.

Extending

Alpine memiliki basis code yang dibuka yang memungkinkan ekstensi dalam beberapa cara. Faktanya, setiap directive dan magic yang tersedia di Alpine sendiri menggunakan API yang tepat. Dalam teori anda dapat membangun ulang semua fungsionalitas Alpine dan menggunakannya untuk anda sendiri.

Lifecycle concerns

Sebelum kita mendalami setiap individual API, pertama mari kita bahas tentang lokasi basis kode anda harus API.

Karena API memiliki dampak pada cara Alpine menginisialisasi halaman, mereka harus didaftarkan SETELAH Alpine di download dan tersedia di halaman, tapi SEBELUM halaman tersebut diinisialisasi.

Ada dua teknik berbeda bergantung pada ada jika anda mengimpor Alpine kedalam bundel atau masukkannya melalui tag `<script>`. Mari lihat keduanya:

Via a script tag

Jika anda memasukkan Alpine melalui tag `<script>`, anda butuh mendaftarkan extensi custom lainnya di dalam event listener Alpine `alpine:init`

Ini contohnya:

```
<html>
    <script src="/js/alpine.js" defer></script>

    <div x-data x-foo></div>

    <script>
        document.addEventListener('alpine:init', () => {
            Alpine.directive('foo', ...)
        })
    </script>
</html>
```

Jika anda ingin mengekstrak ekstensi kode anda sendiri kedalam sebuah file eksternal, anda perlu memastikan bahwa file tag `<script>` tersebut berlokasi SEBELUM file Alpine.

```
<html>
  <script src="/js/foo.js" defer></script>

  <script src="/js/alpine.js" defer></script>

  <div x-data x-foo></div>
</html>;
```

Via an NPM module

jika anda mengimpor Alpine kedalam sebuah bundel, Anda harus memastikan Anda meregistrasi kode ekstensi apa pun DI ANTARANYA ketika anda mengimpor objek global `Alpine` dan ketika anda menginisialisasi Alpine dengan memanggil `Alpine.start()`. Sebagai contoh:

```
import Alpine from 'alpinejs'
```

```
Alpine.directive('foo', ...)

window.Alpine = Alpine
window.Alpine.start()
```

Sekarang kita tahu dimana menggunakan API extension ini, mari lihat lebih dekat bagaimana penggunaan masing-masing:

Custom Directive

Alpine memungkinkan anda untuk mendaftarkan directive custom anda sendiri menggunakan API `Alpine.directive()`

Method Signature

```
Alpine.directive(
  "[name]",
  (el, { value, modifiers, expression }, { Alpine, effect, cleanup }) => {});
```

name	Name nama dari direktif. Nama “foo” untuk contoh akan dikonsumsi sebagai `x-foo`
el	DOM element directive yang ditambahkan ke
value	ke jika disediakan, bagian direktif setelah titik dua. Contoh `'bar'` di dalam `x-foo:bar`
modifiers	Array dari tambahan trailing yang dipisah dengan titik ke directive. Contoh `['baz', 'lob']` dari `x-foo.baz.lob`
expression	Bagian nilai atribut dari directive. Contoh: `law` dari `x-foo=”law”`
Alpine	global objek Alpine
effect	sebuah fungsi untuk membuat efek reaktif yang akan otomatis membersihkan setelah direktif ini dihapus dari DOM
cleanup	sebuah fungsi yang dapat mengoperkan callback yang akan terlebih dijalankan ketika directive unu dihapus dari DOM

Simple Example

ini merupakan sebuah contoh sederhana directive kita akan membuat direksi bernama `x-uppercase`

```
Alpine.directive('uppercase', el => {
    el.textContent = el.textContent.toUpperCase()
})
```

```
<div x-data>
    <span x-uppercase>Hello World!</span>
</div>
```

Evaluating expressions

Ketika mendaftarkan sebuah directive custom, anda mungkin ingin untuk mengevaluasi expresi JavaScript yang di sediakan oleh pengguna:

sebagai contoh, Katakanlah anda ingin membuat sebuah custom directive sebagai sebuah shortcut untuk console.log().

Seperti berikut:

```
<div x-data="{ message: 'Hello World!' }">
  <div x-log="message"></div>
</div>;
```

Anda butuh untuk mengambil nilai asli dari `message` dengan mengevaluasinya sebagai ekspresi JavaScript dengan lingkup `x-data`.

Untungnya, Alpine mengekspose sistem untuk evaluasi ekspresi JavaScript dengan sebuah API `evaluate()`, Ini adalah sebuah contoh:

```
Alpine.directive("log", (el, { expression }, { evaluate }) => {
  // expression === 'message'

  console.log(evaluate(expression));
});
```

Sekarang ketika Alpine menginisialisasi `<div x-log...>`, Alpine akan mengambil ekspresi memasukkannya ke dalam direktif (“message” pada kasus ini), dan mengevaluasinya dalam context dari komponen elemen Alpine saat ini.

Introduksi reactivity

Membangun contoh `x-log` seperti sebelumnya, katakanlah kita ingin `x-log` untuk log nilai dari `message` dan juga log jika nilai berubah.

Berikan contoh template berikut:

```
<div x-data="{ message: 'Hello World!' }">
    <div x-log="message"></div>

    <button @click="message = 'yolo'">Change</button>
</div>
```

Anda ingin “Hellow World!” di log pada awalnya, kemudian kita ingin “yolo” di log setelah menekan tombol `<button>`.

Kita dapat menyesuaikan implementasi `x-log` dan memperkenalkan dua API baru untuk mencapai ini: `evaluateLater()` dan `effect()`

```
Alpine.directive("log", (el, { expression }, { evaluateLater, effect }) => {
  let getThingToLog = evaluateLater(expression);

  effect(() => {
    getThingToLog((thingToLog) => {
      console.log(thingToLog);
    });
  });
});
```

Mari kita telusuri kode diatas baris demi baris.

```
let getThingToLog = evaluateLater(expression);
```

Disini, alih-alih secara langsung mengevaluasi `message` dan mengambil hasilnya, Kita convert string ekspresi (“message”) ke dalam sebuah fungsi JavaScript yang kita akan jalankan kapanpun. Jika anda akan mengevaluasi ekspresi JavaScript lebih dari sekali, sangat direkomendasikan untuk generate sebuah fungsi JavaScript pertama kali dan menggunakannya daripada memanggil `evaluate()` secara langsung. Alasan dibalik itu adalah proses untuk menafsirkan sebuah string sebagai fungsi javascript sangat mahal dan harus dihindari ketika tidak diperlukan.

```
effect(() => {
    ...
})
```

Dengan mengoper sebuah callback ke `effect()`, kita memberitahu Alpine untuk menjalankan secepatnya, kemudian melacak dependensi apapun yang digunakan (properti `x-data` seperti `message` pada kasus kita). Sekarang segera setelah 1 dependency berubah callback ini akan dijalankan ulang ini memberi kita "reactivity".

Anda mungkin menyadari fungsi ini dari `x-effect`. Ini memiliki mekanisme yang sama di belakang layar.

Anda mungkin juga menyadari bahwa `Alpine.effect()` ada dan heran kenapa kita tidak menggunakannya di sini. Alasannya fungsi `effect` disediakan melalui parameter method memiliki fungsional yang khusus yang membersihkan dirinya sendiri ketika directive dihapus dari halaman untuk alasan apapun.

Sebagai contoh, jika untuk beberapa alasan elemen dengan `x-log` dihapus dari halaman, dengan menggunakan `effect()` alih-alih menggunakan `Alpine.effect()` ketika properti `message` berubah, nilai tidak lagi di di log ke console.

```
getThingToLog((thingToLog) => {
  console.log(thingToLog);
});
```

Sekarang kita dapat memanggil `getThingToLo` yang jika anda panggil ulang merupakan fungsi JavaScript versi aktual dari ekspresi string: "message".

Anda mungkin berekspektasi `getThingToCall()` mengembalikan hasilnya secara langsung, tapi alih-alih Alpine mensyaratkan anda untuk mengoper sebuah callback untuk menerima hasil.

Alasan dari ini adalah untuk mendukung ekspresi asinkronus seperti `await getMessage()`. Dengan mengoper sebuah callback "receiver". Alih-alih mendapatkan hasil secara langsung, anda ada diperbolehkan menggunakan direktif untuk bekerja dengan ekspresi asinkronus.

Cleaning Up

Katakanlah anda butuh untuk mendaftarkan sebuah event listener dari sebuah custom directive. Setelah direktif dihapus dari halaman untuk alasan apapun, anda mungkin ingin menghapus event listener juga.

Alpine membuatnya mudah dengan menyediakan sebuah fungsi `cleanup`

ketika mendaftarkan directive custom.
Ini adalah sebuah contoh:

```
Alpine.directive("...", (el, {}, { cleanup }) => {
  let handler = () => {};

  window.addEventListener("click", handler);

  cleanup(() => {
    window.removeEventListener("click", handler);
  });
});
```

Sekarang jika direktif dihapus dari elemen ini atau elemennya sendiri dihapus, event listener akan dihapus juga.

Custom magics

Alpine memungkinkan anda untuk mendaftarkan sebuah custom “magic” (properti atau method) menggunakan `Alpine.magic()`. Magic apapun yang anda daftarkan tersedia ke semua aplikasi anda dengan an awalan `$`.

Method signature

```
Alpine.magic("[name]", (el, { Alpine }) => {});
```

name	Nama magic, nama “foo” untuk contoh akan di konsumsi sebagai `$foo`
el	DOM Element Magic yang terterpicu dari
Alpine	objek Global Alpine

Magic Property

Ini adalah sebuah contoh dari magic helper "$now" yang memudahkan untuk mendapat waktu saat ini di mana pun di Alpine:

```
Alpine.magic("now", () => {
  return new Date().toLocaleTimeString();
});
```

```
<span x-text="$now"></span>;
```

Sekarang tag `<span>` akan berisi waktu saat ini, menyerupai sesuatu seperti "12:00:00 PM".

Seperti yang dapat anda lihat `$now` berperilaku seperti properti statistik, tapi di belakang layar sebenarnya sebuah getter yang mengevaluasi setiap kali properti diakses.

karena itu, anda dapat menerapkan "fungsi" magic dengan mengembalikan sebuah fungsi dari getar.

Magic functions

Sebagai contoh, Jika anda ingin membuat fungsi magic `$clipboard()` yang menerima sebuah string untuk meng-copy ke Clipboard, kita akan mengimplementasikannya seperti ini:

```
Alpine.magic("clipboard", () => {
  return (subject) => navigator.clipboard.writeText(subject);
});
```

```
<button @click="$clipboard('hello world')">Copy "Hello World"</button>
```

Sekarang pengaksesan `$clipboard` mengembalikan fungsi clipboard itu sendiri, kita dapat segera memanggilnya dan mengoper sebuah argumen seperti yang kita lihat di template dengan `$clipboard('hello world')`.

Anda dapat menggunakan syntax yang lebih singkat (sebuah double

arrow function) untuk mengembalikan sebuah fungsi dari sebuah fungsi jika anda lebih suka:

```
Alpine.magic("clipboard", () => (subject) => {
  navigator.clipboard.writeText(subject);
});
```

Writing and sharing plugins

Sampai saat ini anda seharusnya melihat betapa mudah dan sederhananya mendaftarkan custom directive dan magic anda di aplikasi anda, tapi bagaimana dengan sharing fungsi melalui NPM atau yang lain?

Anda dapat memulainya dengan package resmi Alpine “plugin-blueprint”. Ini semudah meng clone repository dan menjalankan `npm install && npm run buil` untuk mendapat otorisasi in plugin.

Untuk mendemonstrasikan proses, Ayo buat sebuah plugin Alpine dari awal bernama `Foo` yang ada directive (`x-foo`) dan magic (`x-foo`).

Kita akan memulai membuat ini untuk konsumsi tag `<script>` sederhana di Alpine. Lalu meningkatkannya ke sebuah modul untuk diimpor ke dalam bundel:

Script include

Mari kita mulai secara terbalik dengan melihat bagaimana plugin kita akan dimasukkan ke dalam sebuah project.

```
<html>
    <script src="/js/foo.js" defer></script>
    <script src="/js/alpine.js" defer></script>
    <div x-data x-init="$foo()">
        <span x-foo="'hello world'">
    </div>
</html>
```

Perhatikan bagaimana script kita dimasukkan SEBELUM Alpine itu sendiri. Ini penting, jika tidak, Alpine telah diinisialisasi ketika kita plugin kita di muat.

Sekarang lihat ke dalam content `/js/foo.js`:

```
document.addEventListener('alpine:init', () => {
    window.Alpine.directive('foo', ...)
```

```
    window.Alpine.magic('foo', ...)
})
```

itu saja! membuat plugin untuk dimasukkan via tag script sangat sederhana dengan Alpine.

Bundle module

Sekarang katakanlah anda ingin membuat plugin yang seseorang dapat diinstal melalui NPM dan memasukkan kedalam bundel mereka.

Seperti contoh terakhir, kita akan memulainya dengan terbalik, memulai dengan bagaimana plugin akan dikonsumsi:

```
import Alpine from "alpinejs";

import foo from "foo";
Alpine.plugin(foo);

window.Alpine = Alpine;
window.Alpine.start();
```

anda akan menyadari bahwa ada API baru di sini: `Alpine.plugin().` Ini adalah method Alpine yang berguna untuk expose dan mencegah konsumen dari plugin anda dari harus mendaftar beberapa direktif dan magic berbeda sendiri.

Sekarang lihat sumber dari plugin yang akan kita eksport dari `foo`

```
export default function (Alpine) {
    Alpine.directive('foo', ...)
    Alpine.magic('foo', ...)
}
```

Kini anda dapat melihat `Alpine.plugin` sangatlah sederhana. Plugin menerima sebuah callback dan secepatnya memanggilnya ketika `Alpine` global menyediakan sebuah parameter untuk digunakan di dalamnya.

Kemudian anda dapat memperluas Alpine sesuka anda.

Async

Alpine dibangun untuk mendukung fungsi asinkronus di banyak tempat ini mendukung yang standar.

Sebagai contoh, katakanlah anda memiliki sebuah fungsi sederhana yang dinamakan `getLabel()` yang anda gunakan sebagai input ke direktif `x-text`:

```
function getLabel() {
  return "Hello World!";
}
```

```
<span x-text="getLabel()"></span>;
```

karena `getLabel` sinkronus, semua berfungsi seperti yang diperkirakan. Sekarang, mari kita berpura-pura `getLabel` membuat sebuah network request untuk mengambil tabel dan tidak mengembalikannya secara seketika (asinkronus). Dengan membuat `getLabel` fungsi asinkronus, anda dapat memanggilnya dari Alpine menggunakan syntax JavaScript `await`.

```
async function getLabel() {
  let response = await fetch("/api/label");

  return await response.text();
}
<span x-text="await getLabel()"></span>;
```

Selain itu, jika anda lebih suka memanggil method di dalam Alpine tanpa diikuti tanda kurung, Anda dapat membiarkannya tanpa tanda kurung dan Alpine akan mendeteksi fungsi yang disediakan sebagai fungsi asinkronus dan menanganinya. Sebagai contoh:

```
<span x-text="getLabel"></span>;
```

CSP (content security policy)

Supaya Alpine dapat mengeksekusi string biasa dari atribut html sebagai ekspresi JavaScript, sebagai contoh `x-on:click="console.log()"`, Alpine membutuhkan untuk bergantung pada sebuah utility yang yang yang melanggar kebijakan keamanan konten “unsafe-eval”.

*di belakang layar, Alpine sebenarnya tidak menggunakan `eval()` sendiri karena lambat dan bermasalah. Alih-alih menggunakan fungsi deklarasi, yang lebih baik, tapi masih melanggar “unsafe-eval”.

Agar bisa mengakomodasi environment di mana content security policy ini diperlukan, Alpine menawarkan sebuah alternatif yang tidak melanggar “unsafe-eval” tapi memiliki sintak yang lebih dibatasi.

Installation

Seperti semua ekstensi Alpine, anda dapat memasukkan melalui tag `<script>` atau module import:

Script tag

```
<html>

  <script src="alpinejs/alpinejs-csp/cdn.js" defer></script>
</html>;
```

Module import

```
import Alpine from "@alpinejs/csp";

window.Alpine = Alpine;
window.Alpine.start();
```

Restrictions

Karena Alpine tidak lagi menafsirkan string sebagai JavaScript biasa, maka harus diuraikan (parse) dan di construct oleh fungsi javascript dari mereka secara manual.

Karena keterbatasan ini, anda harus menggunakan `Alpine.data` untuk mendaftar objek `x-data` anda, dan harus mereferensikan property dan method hanya dengan key saja.

Sebagai contoh, sebuah komponen inline seperti ini tidak akan berfungsi.

```
<!-- Bad -->
<div x-data="{ count: 1 }">

    <button @click="count++">Increment</button>

    <span x-text="count"></span>
</div>
```

Meskipun, memecah ekspresi ke dalam API eksternal, contoh berikut valid dengan CSP build:

```
<!-- Good -->
<div x-data="counter">
    <button @click="increment">Increment</button>

    <span x-text="count"></span>
</div>
```

```
Alpine.data("counter", () => ({
```

```
  count: 1,

  increment() {
    this.count++;
  },
}));
```

SELESAI

Diterjemahkan oleh @hari_kun