stencil

The Then, the Now and the Future

ionic

Manu Mtz.-Almeida

Software Engineer on the
Ionic Framework Team

@manucorporat

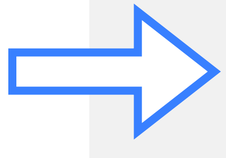
@manucorporat

“**stencil** is a toolchain for building reusable, scalable Design Systems, based in future-proof web standards.”

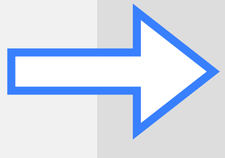

jQuery


ANGULARJS


codiqa




ionic




IONIC

Vue

Ember

React

Angular

Svelte


ionic
core

```
<script type="module" src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.esm.js"></script>
<script nomodule src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></script>

<ion-button>Button</ion-button>
<ion-range min="0" max="10" value="2"></ion-range>
```

```
import { NgModule } from '@angular/core';
import { IonicModule } from '@ionic/angular';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    IonicModule.forRoot(),
    AppRoutingModule,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

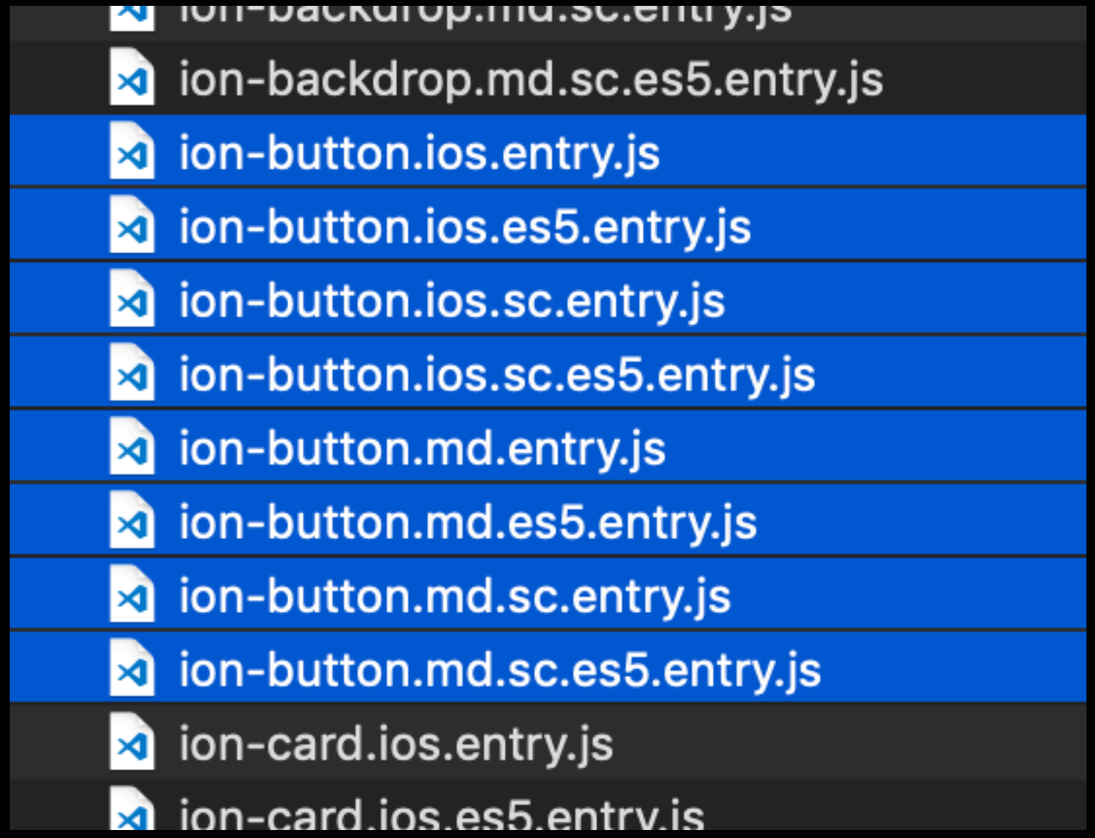
```
import React from 'react';

import { IonButton, IonIcon, IonContent } from '@ionic/react';

export const Page = () => (
  <IonContent>

    <IonButton>
      <IonIcon slot="start name="rocket" />
      Default
    </IonButton>

  </IonContent>
);

ReactDOM.render(<Page />, document.getElementById('app'));
```

```
import Vue from 'vue';
import Ionic from '@ionic/vue';
import App from './App.vue';

Vue.use(Ionic);

new Vue({
  render: h => h(App)
}).$mount('#app');
```

Output Targets

- Bindings
- SSR
- Docs
- Lazy loading
- Raw Web Components

Differential Builds



- Build with iOS and MD styles
- Build in ESM (modern)
- Build in ES5 (legacy, IE11)
- Build SC: browser that doesn't support native shadow-dom

Hot Module Replacement

Service Worker generation

Deep TS integration

Prerendering

DX

Fast incremental builds

Familiar API

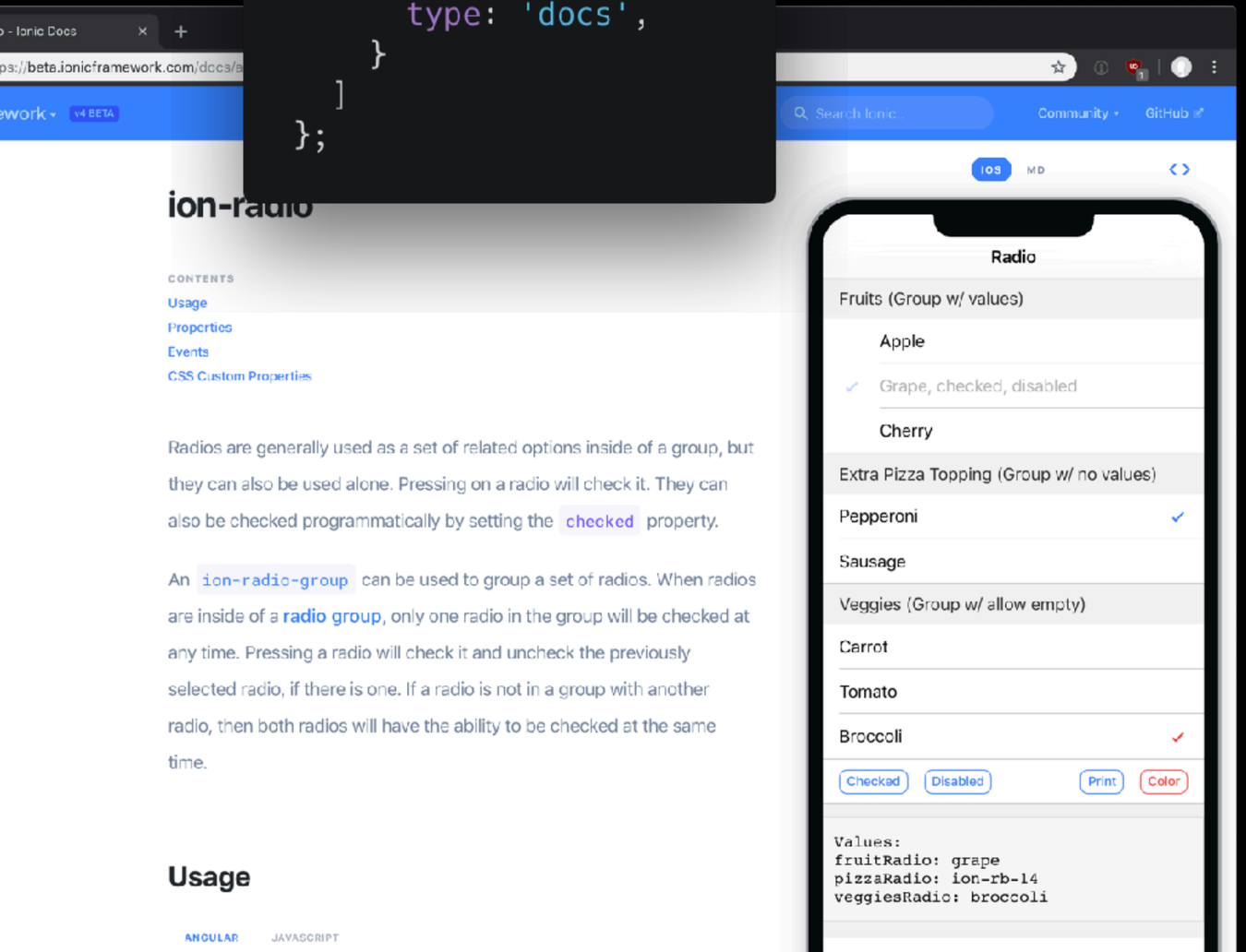
Docs generation

Visual-diff testing

E2E testing

Docs generation

```
export const config = {
  outputTargets: [
    {
      type: 'docs',
    }
  ]
};
```



readme.md

ion-radio

Radios are generally used as a set of related options inside of a group, but they can also be used alone. Pr
radio will check it. They can also be checked programmatically by setting the `checked` property.

An `ion-radio-group` can be used to group a set of radios. When radios are inside of a radio group, only or
group will be checked at any time. Pressing a radio will check it and uncheck the previously selected radio,
one. If a radio is not in a group with another radio, then both radios will have the ability to be checked at th

Properties

Property	Attribute	Description
`checked`	`checked`	If `true`, the radio is selected. Defaults to `false`.
`color`	`color`	The color to use from your application's color palette. Default options are: `"primary"`, `"secondary"`, `"tertiary"`, `"success"`, `"warning"`, `"danger"`, `"light"`, `"medium"`, and `"dark"`. For more information on colors, see theming.
`disabled`	`disabled`	If `true`, the user cannot interact with the radio. Defaults to `false`.
`mode`	`mode`	The mode determines which platform styles to use. Possible values are: `"ios"` or `"md"`.
`name`	`name`	The name of the control, which is submitted with the form data.
`value`	--	the value of the radio.

Events

stencil one

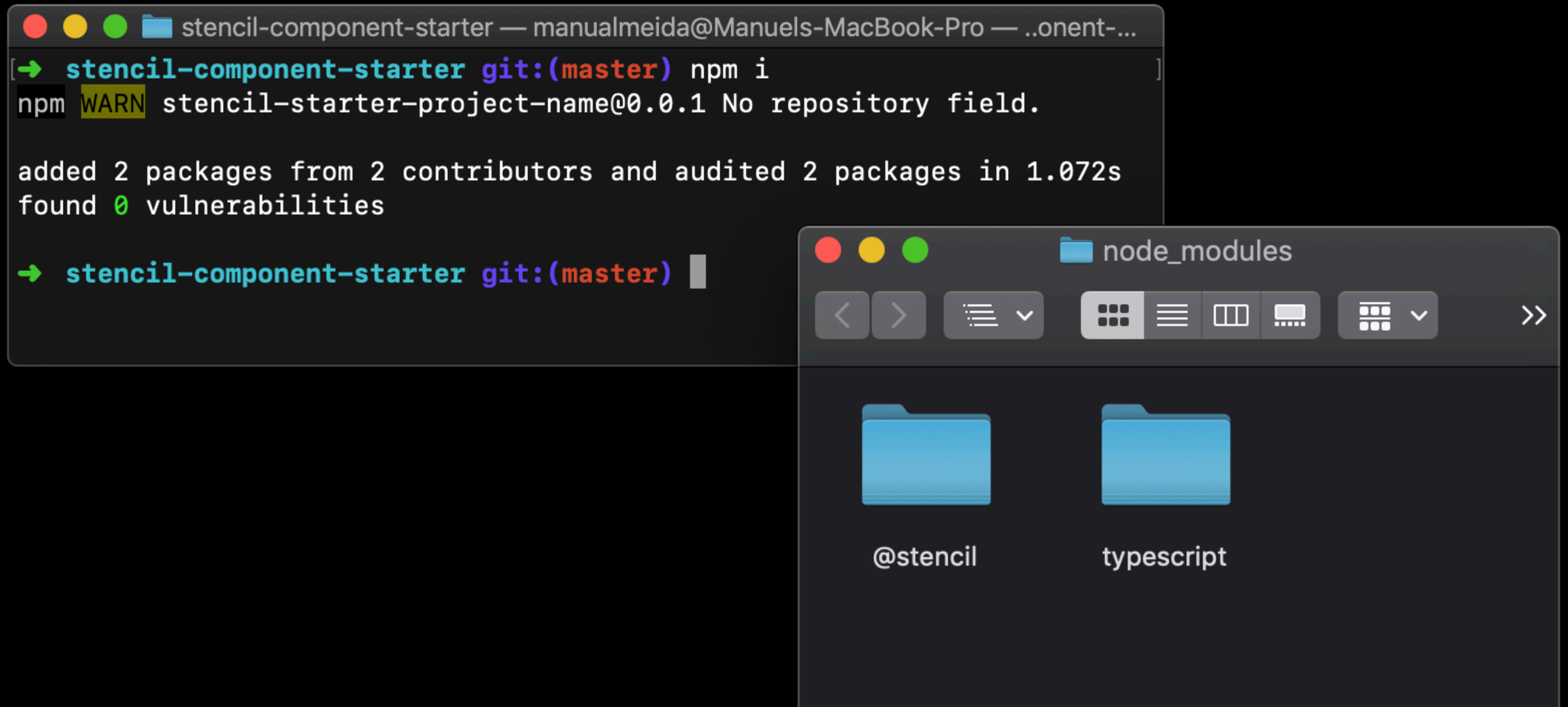
Consolidation of the API

New compiler architecture

Zero dependencies

Faster/smaller runtime

Dependency free

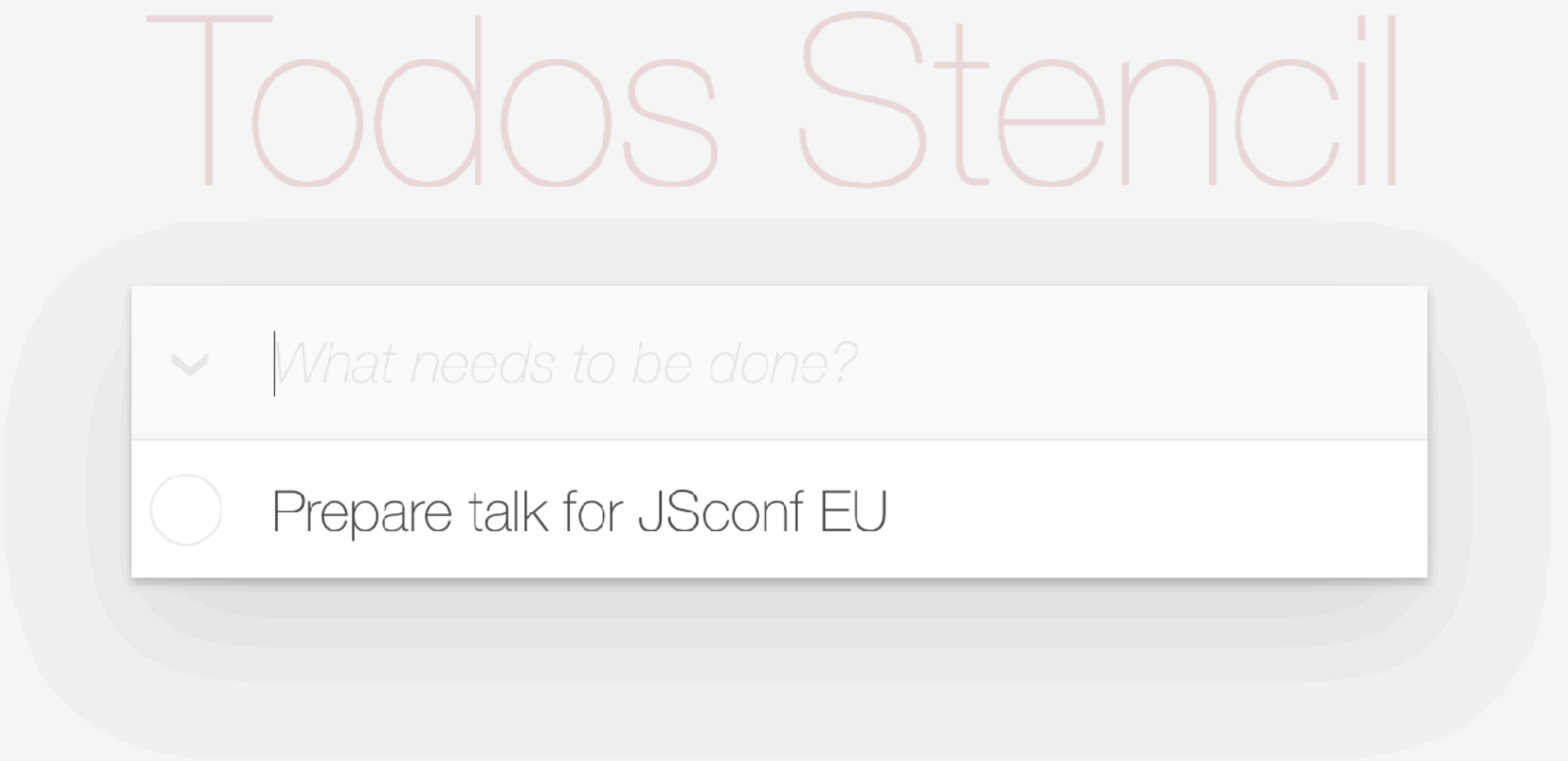


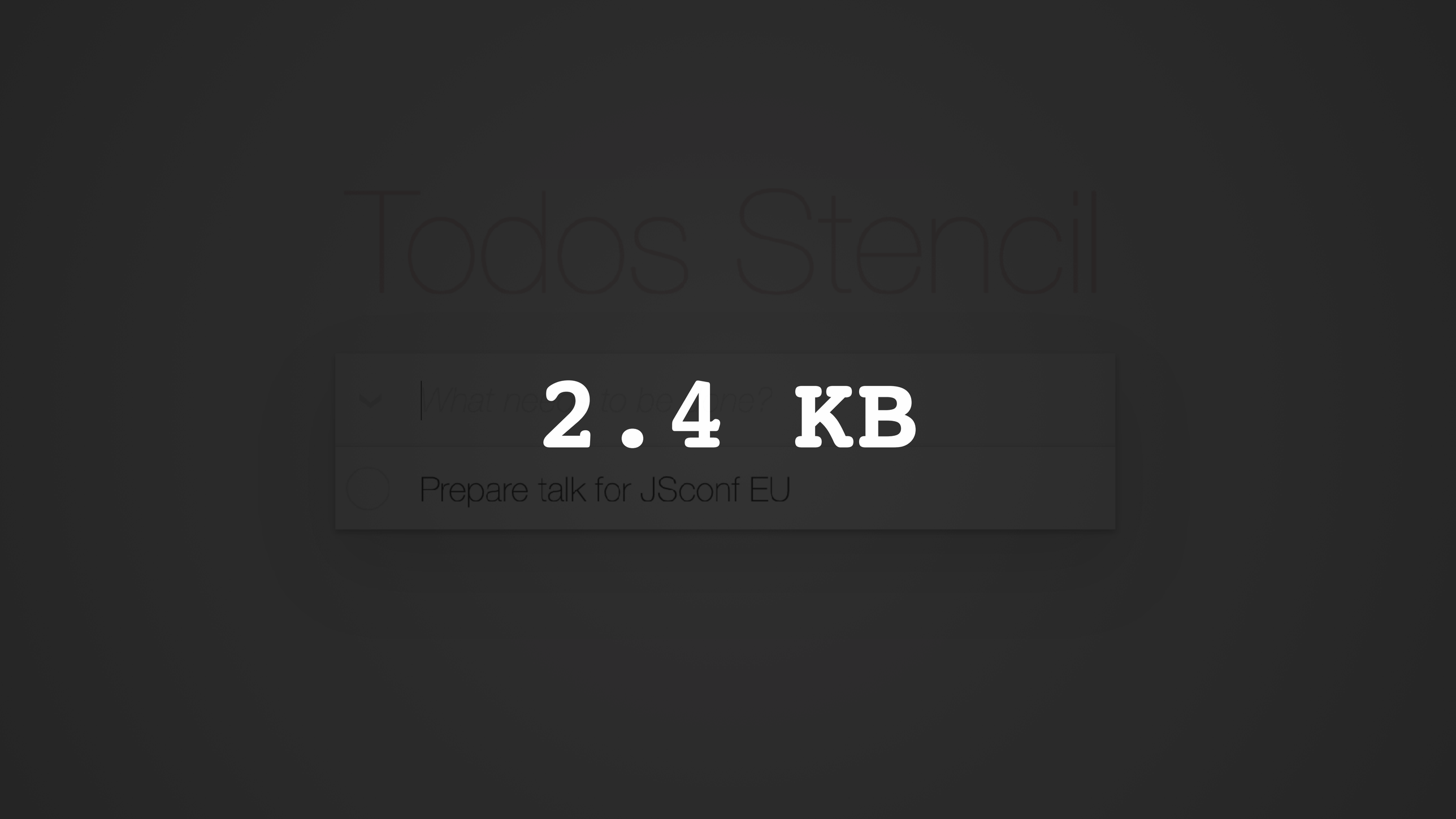
Components generated by
Stencil are **dependency free**

87 bytes

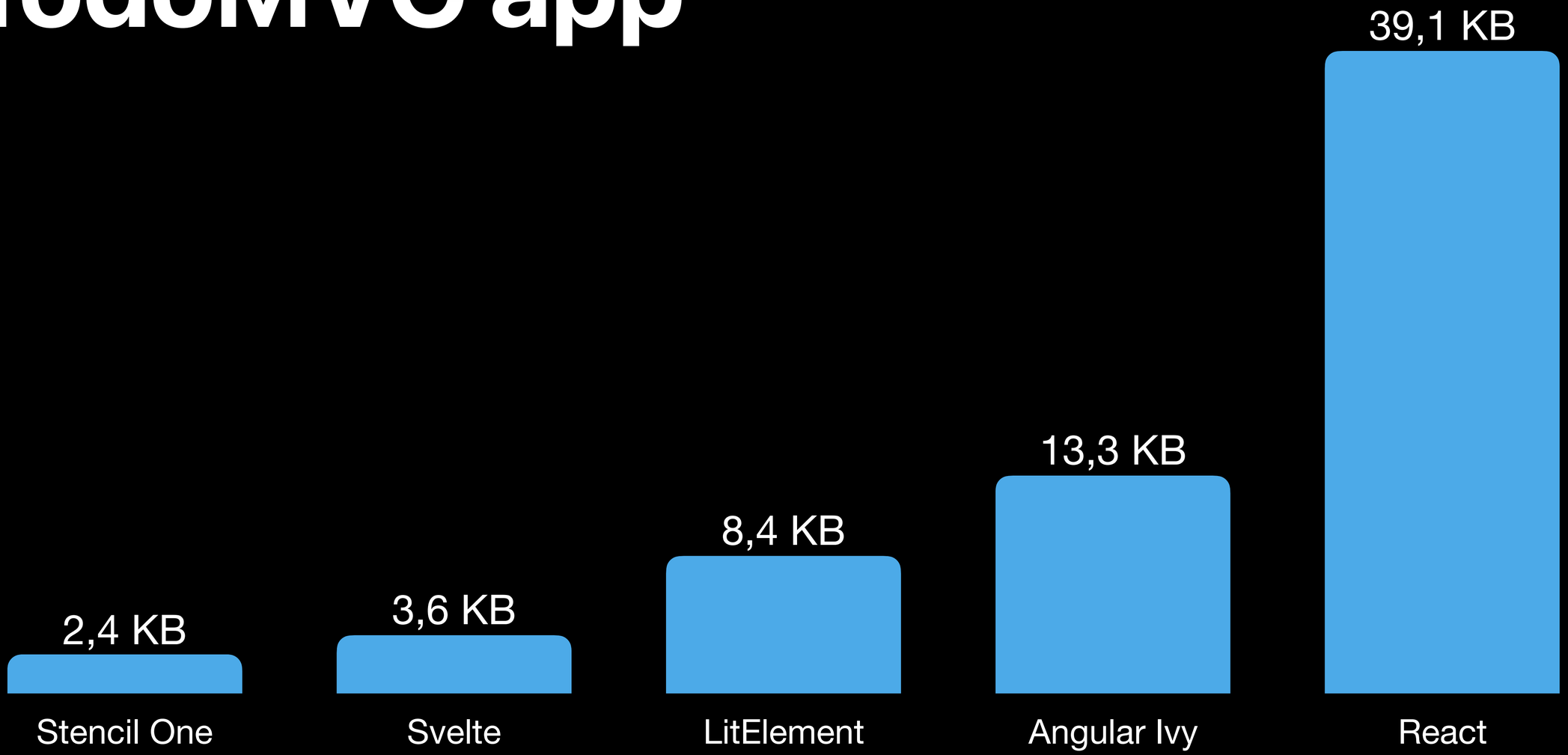
hello-world

(compressed)


Todos Stencil
What needs to be done?
Prepare talk for JSconf EU

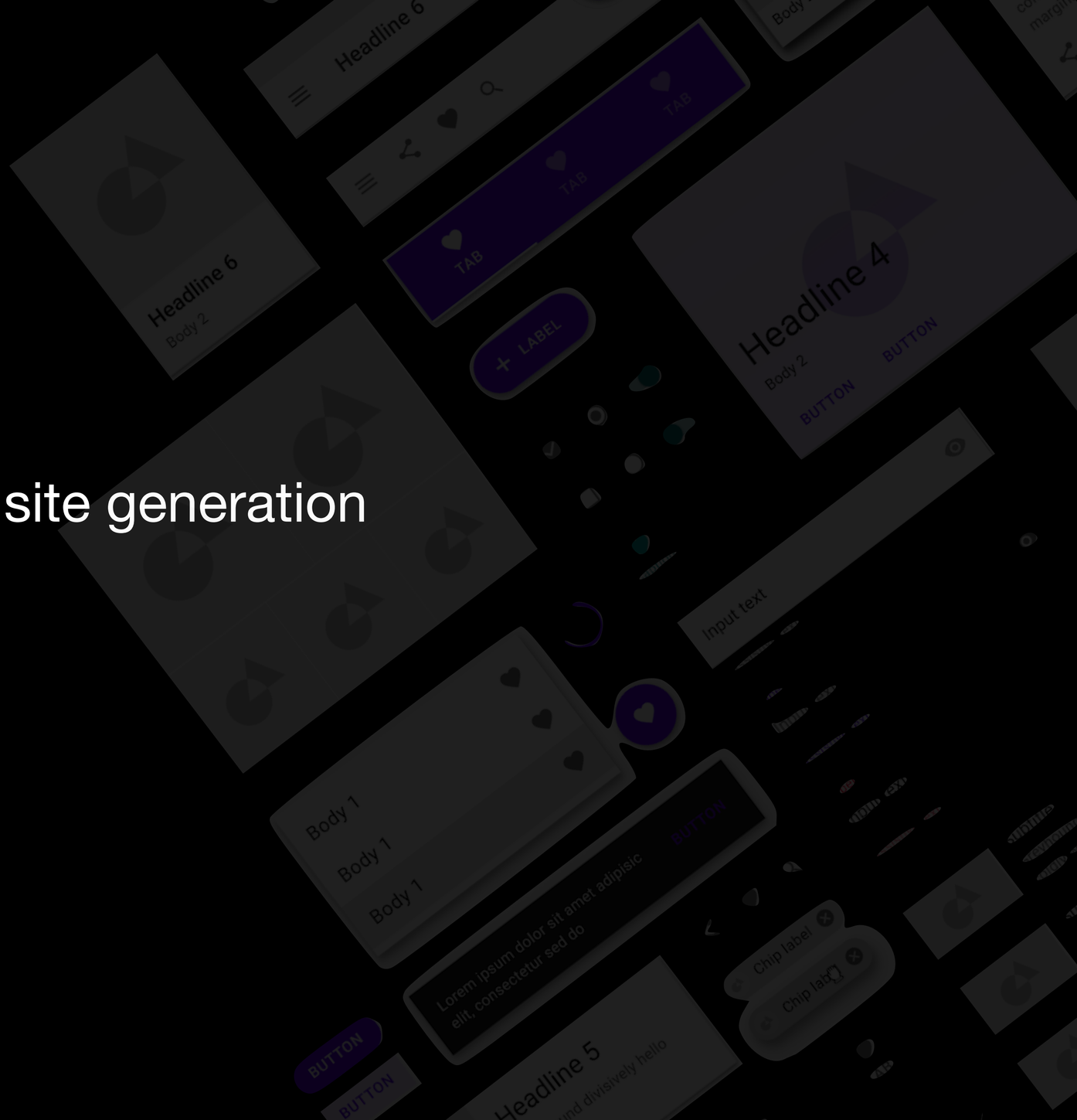

Todos Stencil
What needs to be done?
Prepare talk for JSconf EU
2.4 KB

TodoMVC app


2,4 KB
3,6 KB
8,4 KB
13,3 KB
39,1 KB
Stencil One
Svelte
LitElement
Angular Ivy
React


DS

- Support and Advisory
- Framework bindings
- Playground
- Automatic documentation site generation
- Usage tracking
- Visual Regression testing
- Per component versioning
- Training



Docs for your DS

System XYZ 0.0.1

OVERVIEW

- Getting Started
- Design Tokens
- Changelog

COMPONENTS

- Badge
- Button
- Card
- Checkbox
- Code
- Container
- Dialog
- Grid
- Heading
- Input

Getting Started

This is the description for the page, pulled from the data. It can be edited in the markdown files.

Installing lorem ipsum

System XYZ can be installed using the following command:

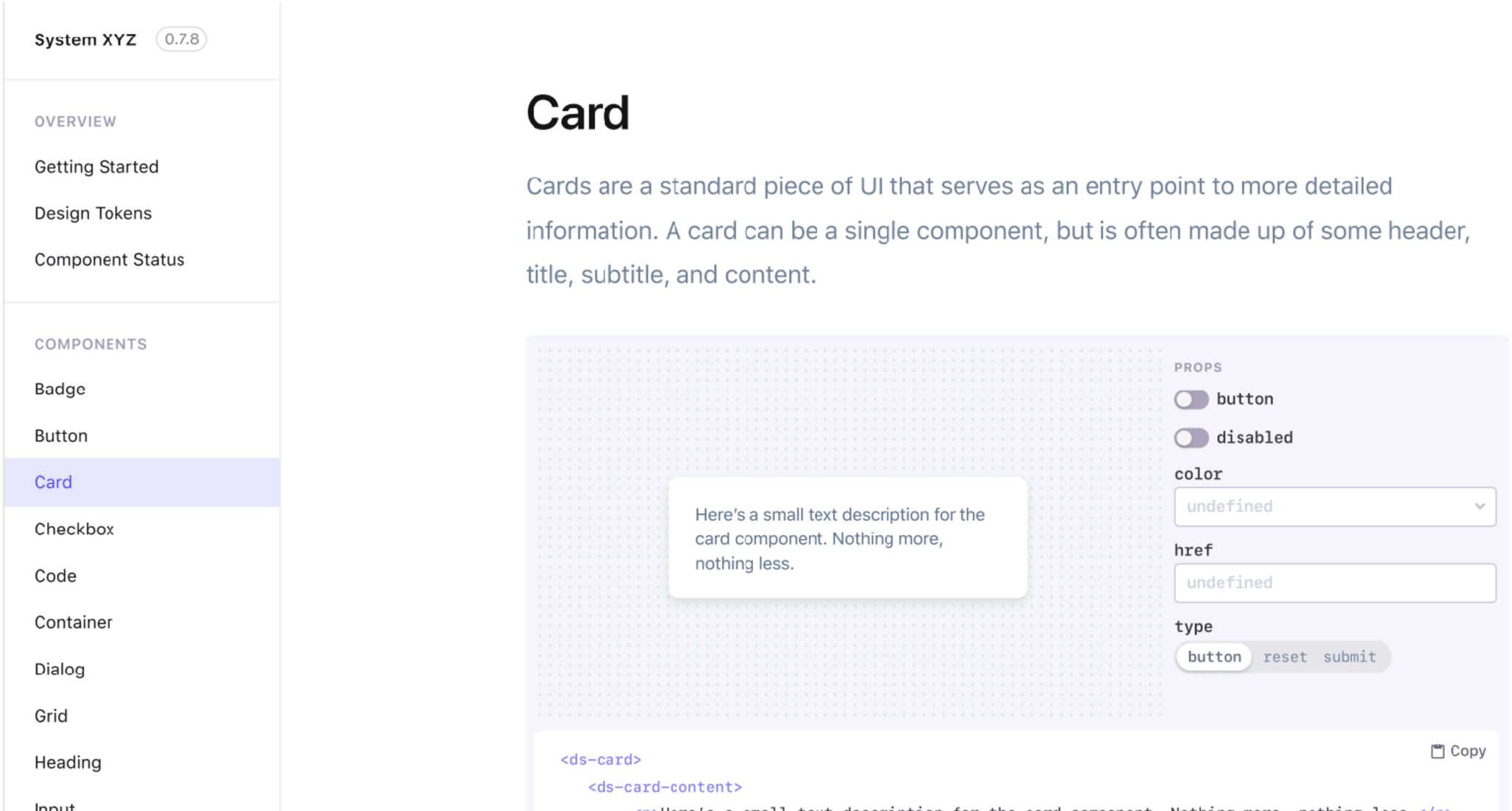
```
npm install @evilcorp/systemxyz
```

Installing dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris:

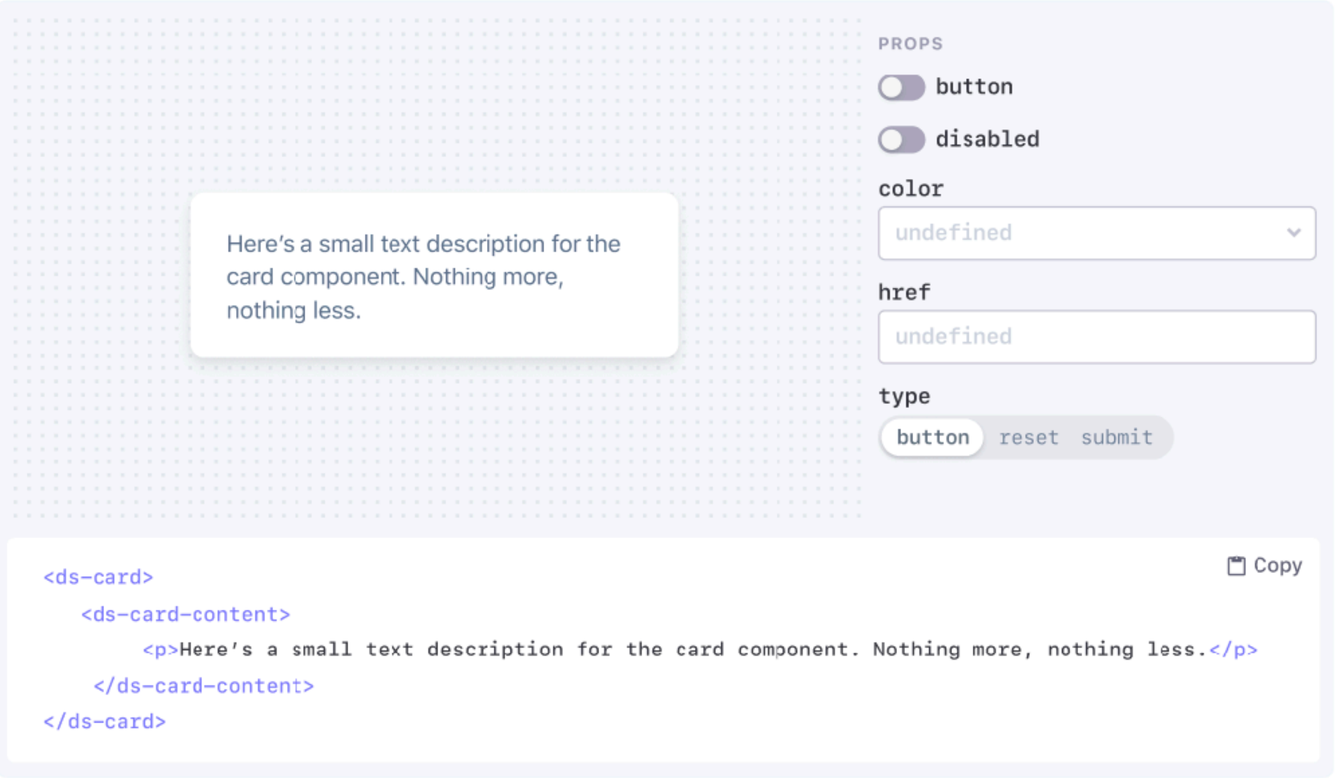
```
npm install @evilcorp/component
```

Docs for your DS



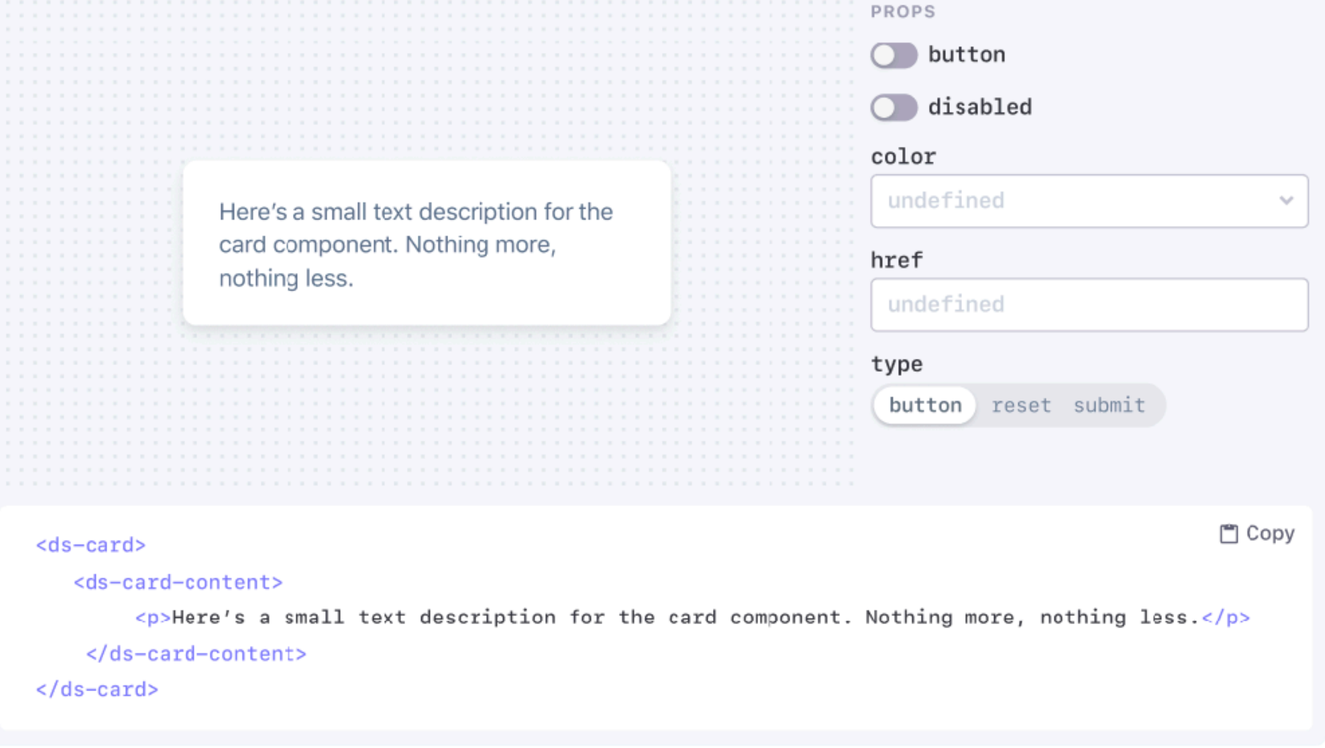
Card

Cards are a standard piece of UI that serves as an entry point to more detailed information. A card can be a single component, but is often made up of some header, title, subtitle, and content.



Usage

CONTENTS



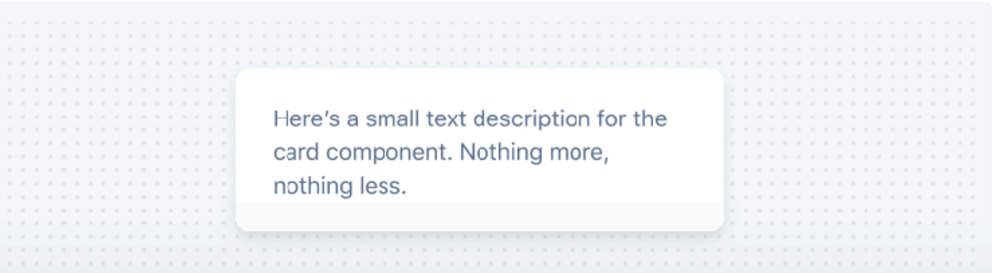
Copy

```
<ds-card>
   <ds-card-content>
        <p>Here's a small text description for the card component. Nothing more, nothing less.</p>
    </ds-card-content>
</ds-card>
```

Usage

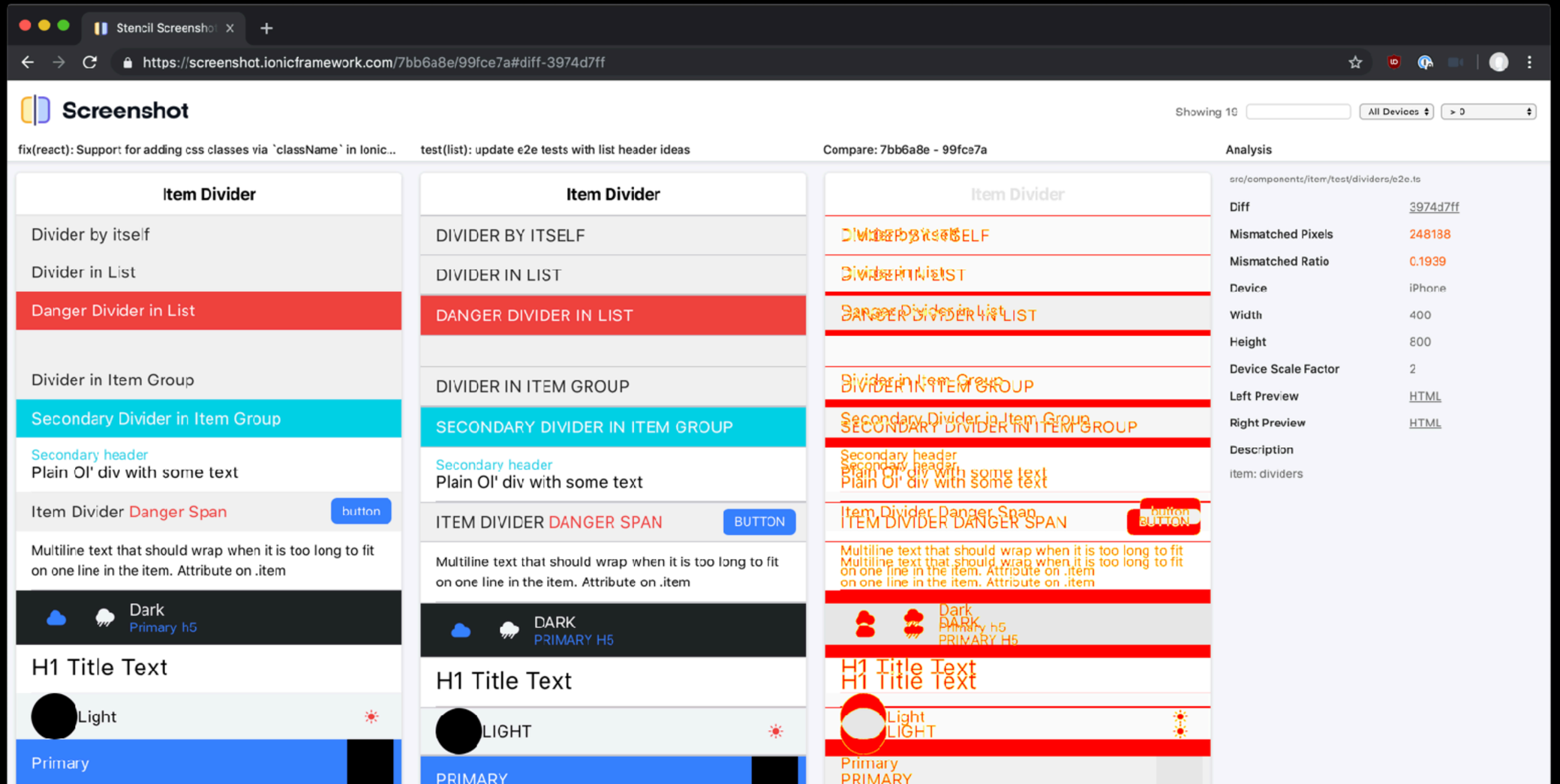
Basic Example

Basic usage for the card component is quite simple. To use a card, follow this structure:



CONTENTS

Visual Diff testing

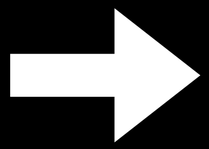


Roadmap

- CSS modules
- Web Workers
- Compiler API
- Faster Incremental Build
- New “Custom Elements” output
- Source maps
- Forms

CSS modules

```
@Component({
  tag: 'my-cmp',
  stylesUrl: './my-component.css'
})
export MyComponent {}
```



```
import styles from './my-component.css';

@Component({
  tag: 'my-cmp',
  styles
})
export MyComponent {}
```

CSS modules

NO BREAKING CHANGES

CSS modules

```
import md from './my-component.md.css';
import ios from './my-component.ios.css';

@Component({
  tag: 'my-cmp',
  styles: {
    md,
    ios
  }
})
export MyComponent {}
```

CSS modules

```
import globalStyles from '../global.css';
import styles from './my-component.css';

@Component({
  tag: 'my-cmp',
  styles: [
    globalStyles,
    styles
  ]
})
export MyComponent {}
```

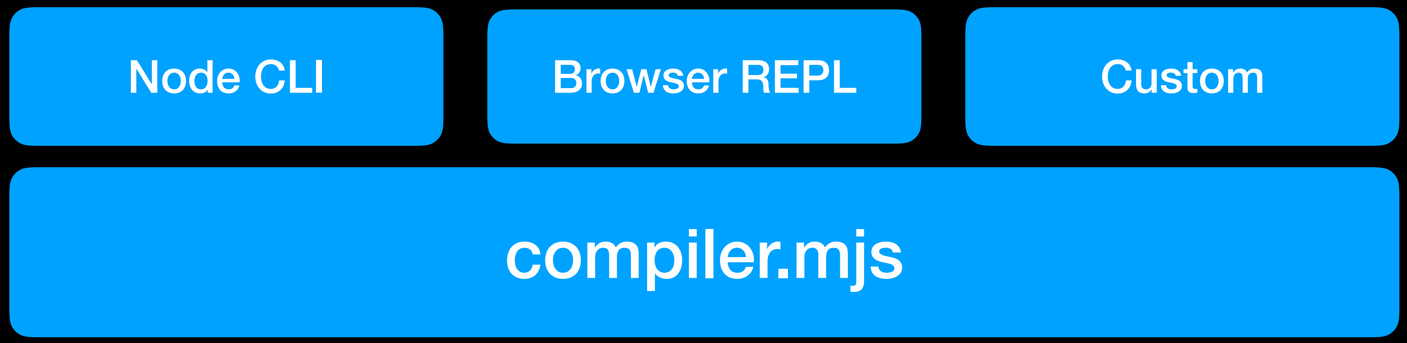
CSS modules

```
import styles from './my-component.css';

// Dynamically modify styles
styles.insertRule('#blanc { color: white }', 0);

@Component({
  tag: 'my-cmp',
  styles
})
export MyComponent {}
```

CSS modules

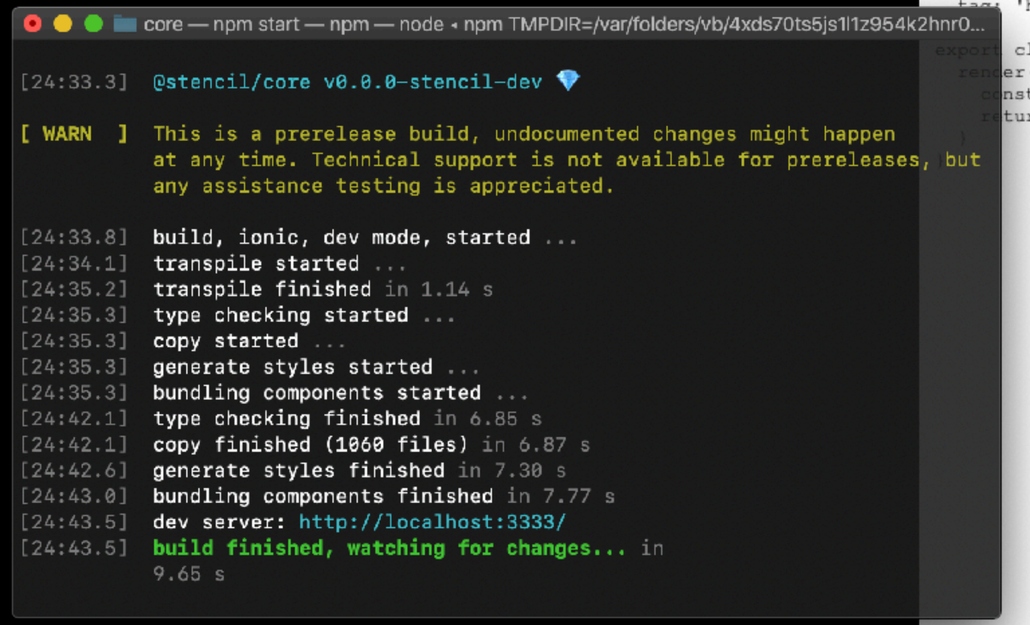
- Single API for all CSS
- CSS unduplication out of the box
- Works better outside stencil
  - Build stencil outside stencil
- Extendable
- It will be eventually standarized
  - https://github.com/w3c/webcomponents/issues/759

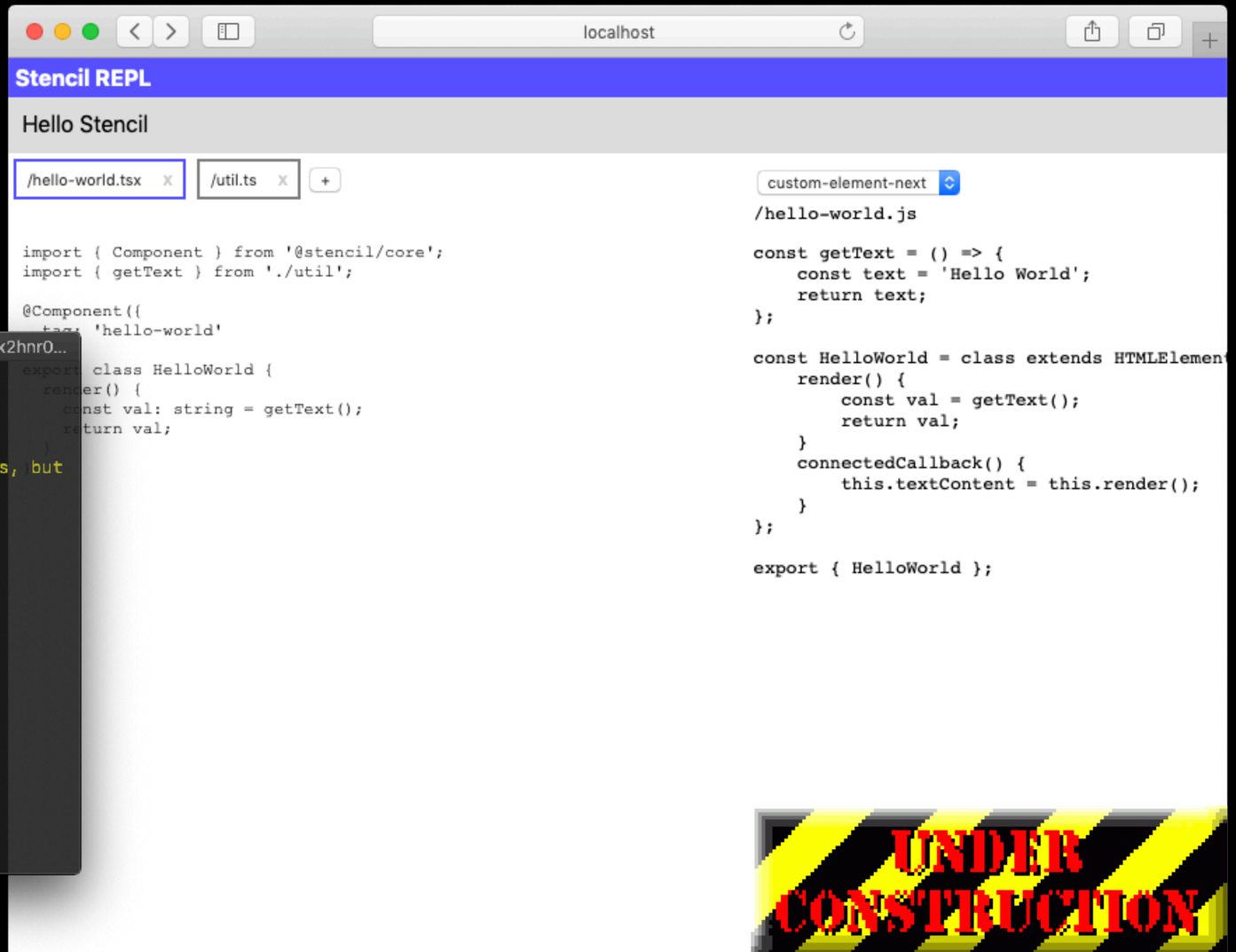
Compiler API



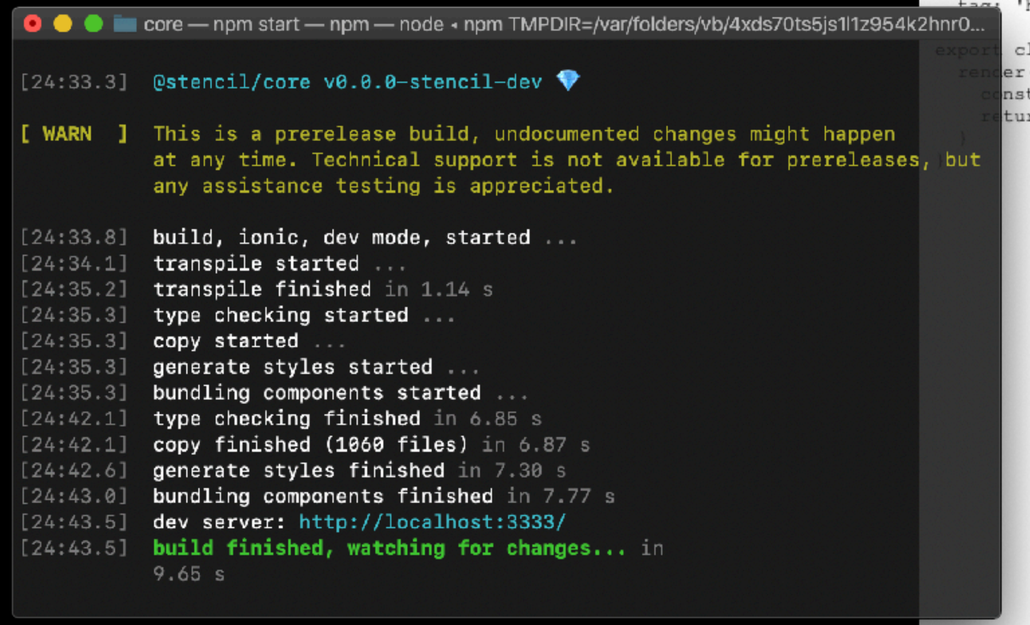
Same “script” works in Node and Browser

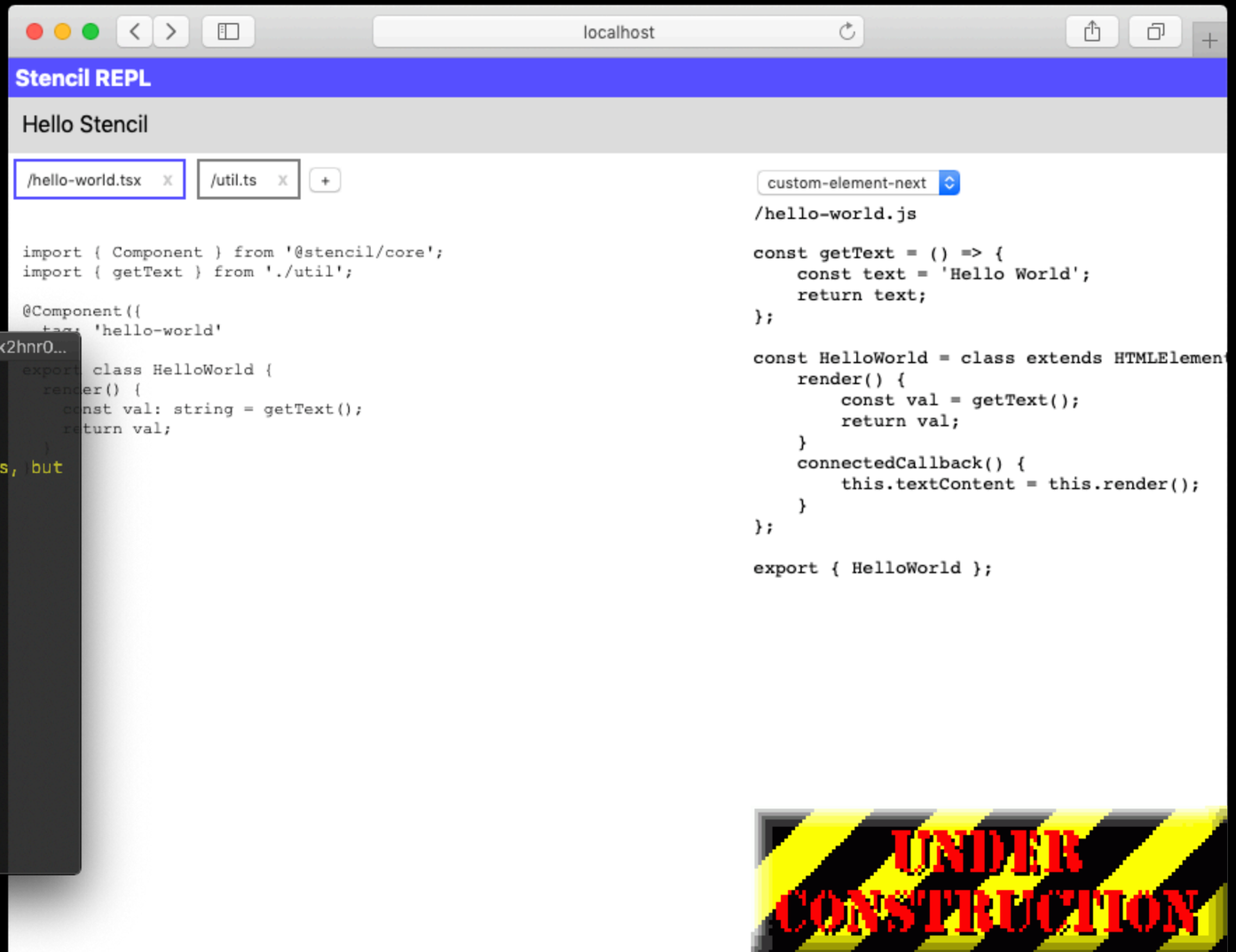
Compiler API





Compiler API





Compiler API

```
import { createCompiler } from './compiler.mjs';

// Create a compiler with some user config.
const compiler = await createCompiler(CONFIG);

// Single build
const buildResults = await compiler.build();

// Watch build
const watcher = await compiler.createWatcher();
watcher.on('buildFinish', results => {
  console.log(results);
});
watcher.start();
```

Fast Incremental Builds

- Typescript 3.6 incremental build
- All builds are incremental (even the first one)
- Typechecking is incremental
- Transpilation + type checking
  - From 10seconds to ~100ms (in ionic/core)

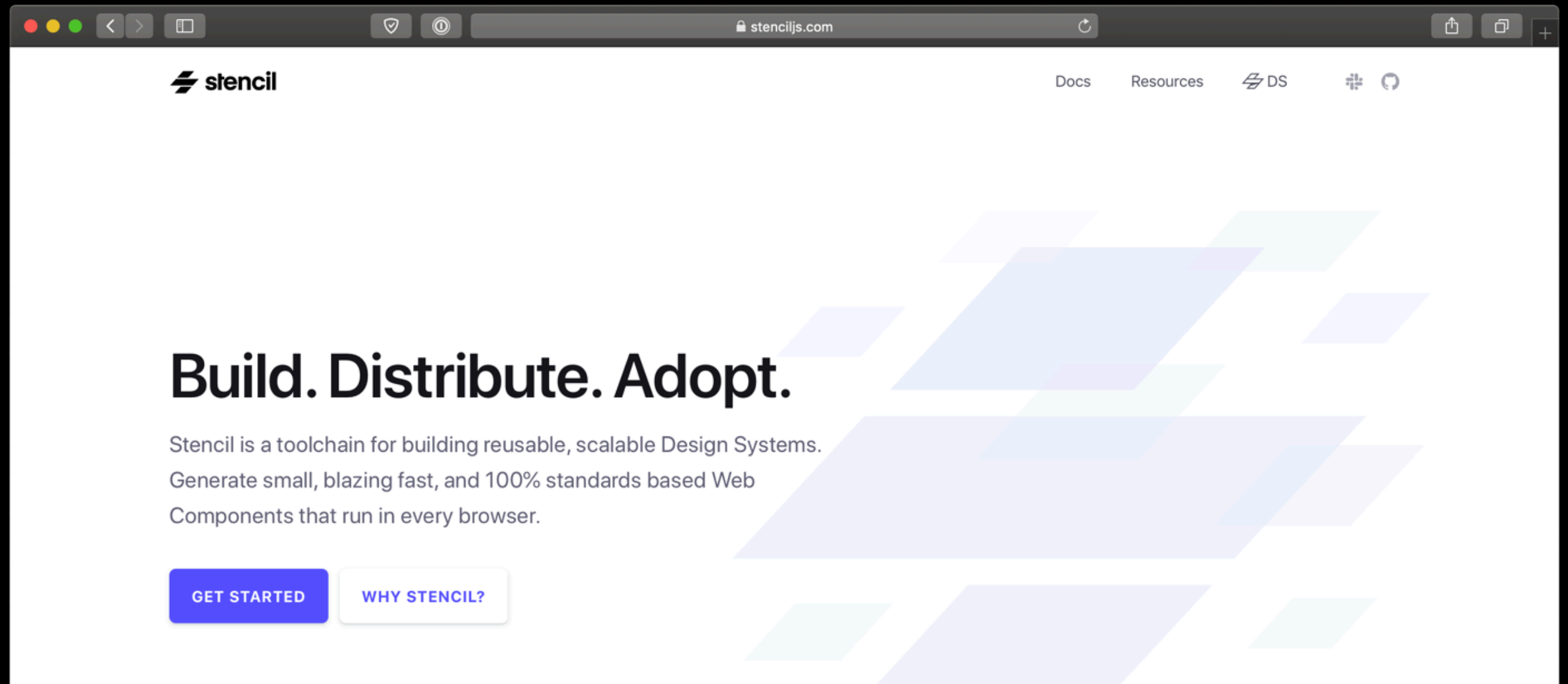
New “Custom Elements”

- Web Components extending from HTMLElement
- No lazy loading
- No self-register
- Works out of the box with WebPack / Rollup

New “Custom Elements”

stenciljs.com

```
$ npm init stencil
```



Thank You

@manucorporat