No Build

Copenhagen JS - 06 Mar 2025

Rohit Sharma

rohit.sh | github.com/r0hitsharma

We're all (mostly) using frameworks in frontend development.

That’s useful, sometimes we just want to work on something quick, but we’re now used to just starting from `<framework> init`.

This also prevents us from considering approaches/architectures which are cumbersome in our frameworks but totally fine without it.

What do frameworks solve for us:

- Componentization
- State management
- DOM manipulation
- Building (transpiling, tree shaking, bundling, & more)
- Dependency management

What do frameworks solve for us:

- Componentization
- ~~State management~~
- DOM manipulation
- ~~Building (transpiling, tree shaking, bundling, & more)~~
- ~~Dependency management~~

index.html

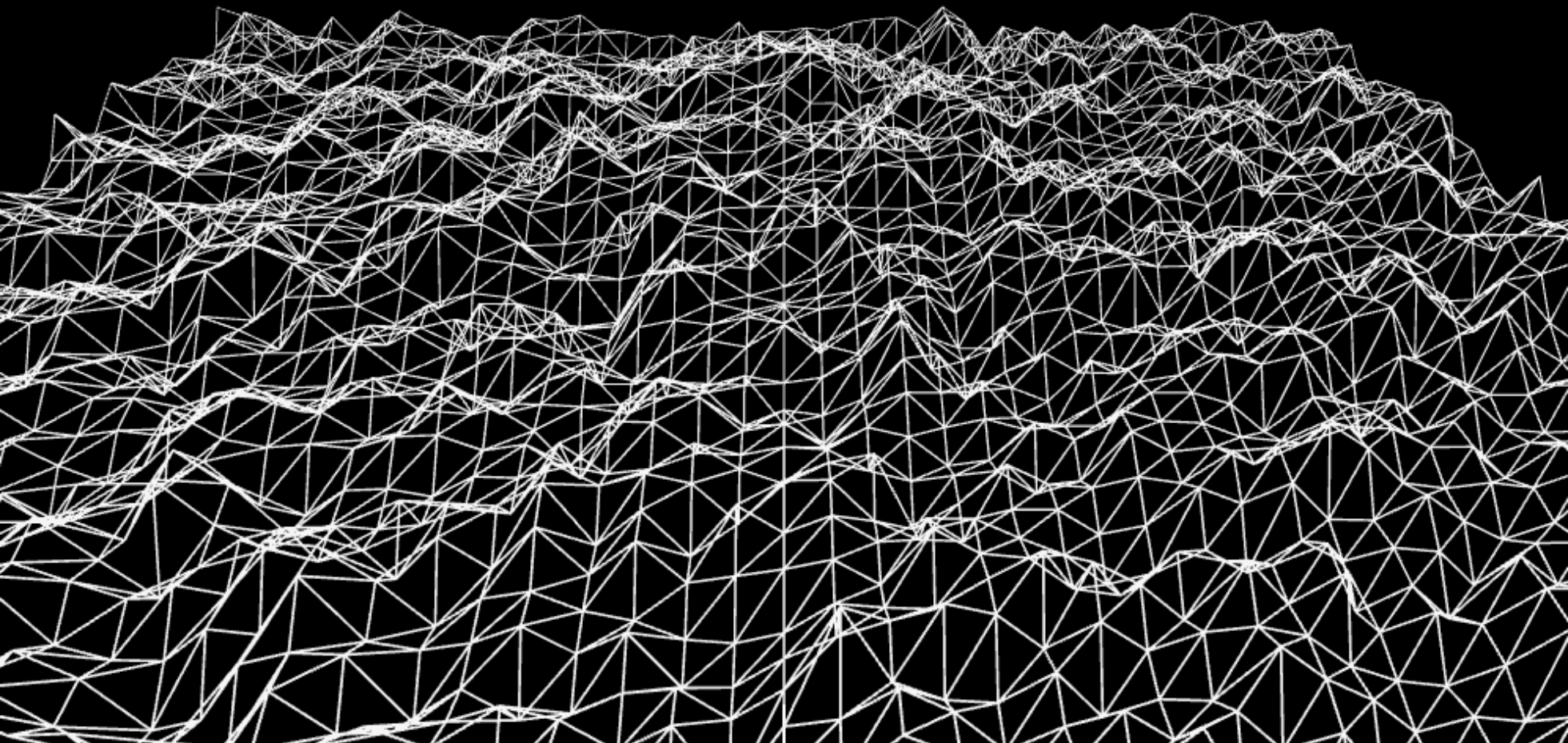
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>No Build</title>
  <link rel="stylesheet" href="https://unpkg.com/chota@latest">
</head>
<body class="container">
  <header class="is-center">
    <section>
      <h1>Perlin Terrain</h1>
    </section>
  </header>

  <main class="is-center"></main>

  <script src="https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js"></script>
  <script src="./perlin-terrain.js"></script>
</body>
</html>
```

perlin-terrain.js

sketch using p5.js

```
// width, height and scale collide with p5 globals
// so are shortened here
const w = 1000, h = 800;
const sc = 20;

const [rows, columns] = [h, w].map(x => x/sc);
const pOffset = 0.5;

function setup() {
  createCanvas(w, h, WEBGL);
  frameRate(24);
}

// this is what we change each frame to
// move perlin noise along y-axis
let flying = 0;
```

```
function draw() {
  background(0);

  stroke(255);
  noFill();

  // translate(width/2, height/2);
  rotateX(PI/3);

  translate(-w/2, -h/2);

  for(let y= 0; y < rows; y++){
    beginShape(TRIANGLE_STRIP);

    for(let x = 0; x < columns; x++){
      vertex(
        x * sc, y * sc,
        noise(x * pOffset, (y * pOffset) - flying) * 100)
      ;
      vertex(
        x * sc, (y+1) * sc,
        noise(x * pOffset, ((y +1) * pOffset) - flying) * 100
      );
    }
```



Add auto-reloading (via Makefile)

```
.PHONY: dev
dev:
  npx live-server --no-browser .
```

Move to ES modules

```
<main class="is-center"></main>

-  <script src="https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js"></script>
-  <script src="./perlin-terrain.js"></script>
+  <script type="module" src="./perlin-terrain.js"></script>
 </body>
 </html>
```

```
+import 'https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js';

-// width, height and scale collide with p5 globals so are shortened here
-const w = 1000, h = 800;
+// p5 attaches itself to window via a side-effect
+const { p5 } = window;
+
+const width = 1000, height = 800;
 const sc = 20;

-const [rows, columns] = [h, w].map(x => x/sc);
+const [rows, columns] = [height, width].map(x => x/sc);
 const pOffset = 0.5;
```

Importmap

```
<main class="is-center"></main>

+  <script type="importmap">
+    {
+      "imports": {
+        "p5": "https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js"
+      }
+    }
+  </script>
   <script type="module" src="./perlin-terrain.js"></script>
```

```
-import 'https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js';
+import 'p5';
```

References:

- https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script/type/importmap
- https://github.com/jsenv/importmap-node-module

package.json

```
{
  "name": "no-build-sketch",
  "scripts": {
    "dev": "live-server --no-browser ."
  },
  "devDependencies": {
    "live-server": "^1.2.2"
  },
  "dependencies": {
    "chota": "^0.9.2",
    "p5": "^1.11.3"
  }
}
```

```
<script type="importmap">
{
  "imports": {
-         "p5": "https://cdn.jsdelivr.net/npm/p5@1.11.3/lib/p5.min.js"
+         "p5": "./node_modules/p5/lib/p5.min.js"
  }
}
```

Type hinting

```
    },
    "devDependencies": {
+     "@types/p5": "^1.7.6",
      "live-server": "^1.2.2"
    }
  },
```

At this point we have:

- Live reload
- Import via importmap
- Dependency management via package.json
- Components and isolation using ES modules
- Scripts via package.json
- Type hinting, inference and correctness check

What about interactivity?

What about interactivity?

Note: most frameworks support a lot more rendering options (and other features). Most importantly rehydration/resumability of SSR chunks.

Server-side rendering in a HTTP framework (Hono)

```
import { serve } from '@hono/node-server'
import { Hono } from 'hono'
import { type FC, useState } from 'hono/jsx'

const app = new Hono()

function Counter() {
  const [count, setCount] = useState(0)

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  )
}

app.get('/', (c) => c.html(<Counter />))

const port = 3000
console.log(`Server is running on http://localhost:${port}`)

serve({ fetch: app.fetch, port })
```

Server-side rendering in a HTTP framework (Hono)

- Pretty straightforward
- Can respond/stream HTML to the client
- No rehydration/resumability so not interactive
- Though can render any component without changes

  (might need more config than this basic example)

References:

- https://hono.dev/docs/guides/jsx
- https://bun.sh/docs/runtime/jsx
- https://docs.deno.com/runtime/reference/jsx

Transpile JSX to JS via esm.sh

Reference: https://github.com/esm-dev/esm.sh/blob/main/README.md

Transpiling using esm.sh/run

Adding `react` and `react-dom` dependencies.

```
   </header>

-  <main class="is-center"></main>
+  <main class="is-center">
+    <div id="root"></div>
+  </main>

   <script type="importmap">
     {
       "imports": {
-        "p5": "./node_modules/p5/lib/p5.min.js"
+        "p5": "./node_modules/p5/lib/p5.min.js",
+        "react-dom": "https://esm.sh/react-dom/client",
+        "react": "https://esm.sh/react"
       }
     }
```

Transpiling using esm.sh/run

Adding react JSX as `text/babel` or `text/jsx` template.

```
-  <script type="module" src="./perlin-terrain.js"></script>
+
+  <script type="module" src="https://esm.sh/run"></script>
+
+  <script type="text/babel">
+    import ReactDOM from 'react-dom';
+    import { useState } from 'react';
+
+    function Counter() {
+      const [count, setCount] = useState(0)
+      return (
+        <div>
+          <p>Count: {count}</p>
+          <button onClick={() => setCount(count + 1)}>Increment</button>
+        </div>
+      )
+    }
+
+    const root = ReactDOM.createRoot(document.getElementById('root'));
+    root.render(<Counter />);
+  </script>
 </body>
```

Transpiling using esm.sh/run

- Powerful but limiting

```
+ Full interactivity
- Only one template per react-root
- Remote transpilation
```

- What if we could transpile client side?
  - Using request context like with SSR
  - Return fully client-side transpiled JS
  - Request would be made by a HTML page via a `<script />`

Hono server with transpilation

`server.tsx`

```tsx
import { serve } from '@hono/node-server'
import { Hono } from 'hono'

import { serveStatic } from '@hono/node-server/serve-static'
import { esbuildTranspiler } from '@hono/esbuild-transpiler/node'

const app = new Hono()

app.get(':scriptName{.+.tsx?}', esbuildTranspiler())
app.use('*', serveStatic({ root: './static' }))

const port = 3000
console.log(`Server is running on http://localhost:${port}`)

serve({
  fetch: app.fetch,
  port
})
```

Hono server with transpilation

`static/index.html`

```
<!DOCTYPE html>
<html lang="en">
<head> ... </head>
<body class="container">
  <main class="is-center">
    <div id="root"></div>
  </main>

  <script type="importmap">
    {
      "imports": {
        "p5": "./node_modules/p5/lib/p5.min.js",
        "react-dom/client": "https://esm.sh/react-dom/client",
        "react": "https://esm.sh/react"
      }
    }
  </script>

  <!-- server will respond with transpiled JS -->
  <script type="module" src='./index.tsx'></script>
</body>
</html>
```

Hono server with transpilation

`index.tsx` Response

```
// HTTP/1.1 200 OK
// content-type: text/javascript
// content-length: 503
// Date: Mon, 10 Mar 2025 23:03:13 GMT
// Connection: keep-alive
// Keep-Alive: timeout=5

import ReactDOM from "react-dom/client";
import React, { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return /* @__PURE__ */ React.createElement("div", null,
  /* @__PURE__ */ React.createElement("p", null, "Count: ", count),
  /* @__PURE__ */ React.createElement(
    "button",
    { onClick: () => setCount(count + 1) },
    "Increment"
    )
  );
}
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(/* @__PURE__ */ React.createElement(Counter, null));
```

Hono server with transpilation

`static/index.tsx`

```tsx
// Dependencies are resolved using importmap
import ReactDOM from 'react-dom/client';
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0)
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root')!);
root.render(<Counter />);
```

Hono server with transpilation

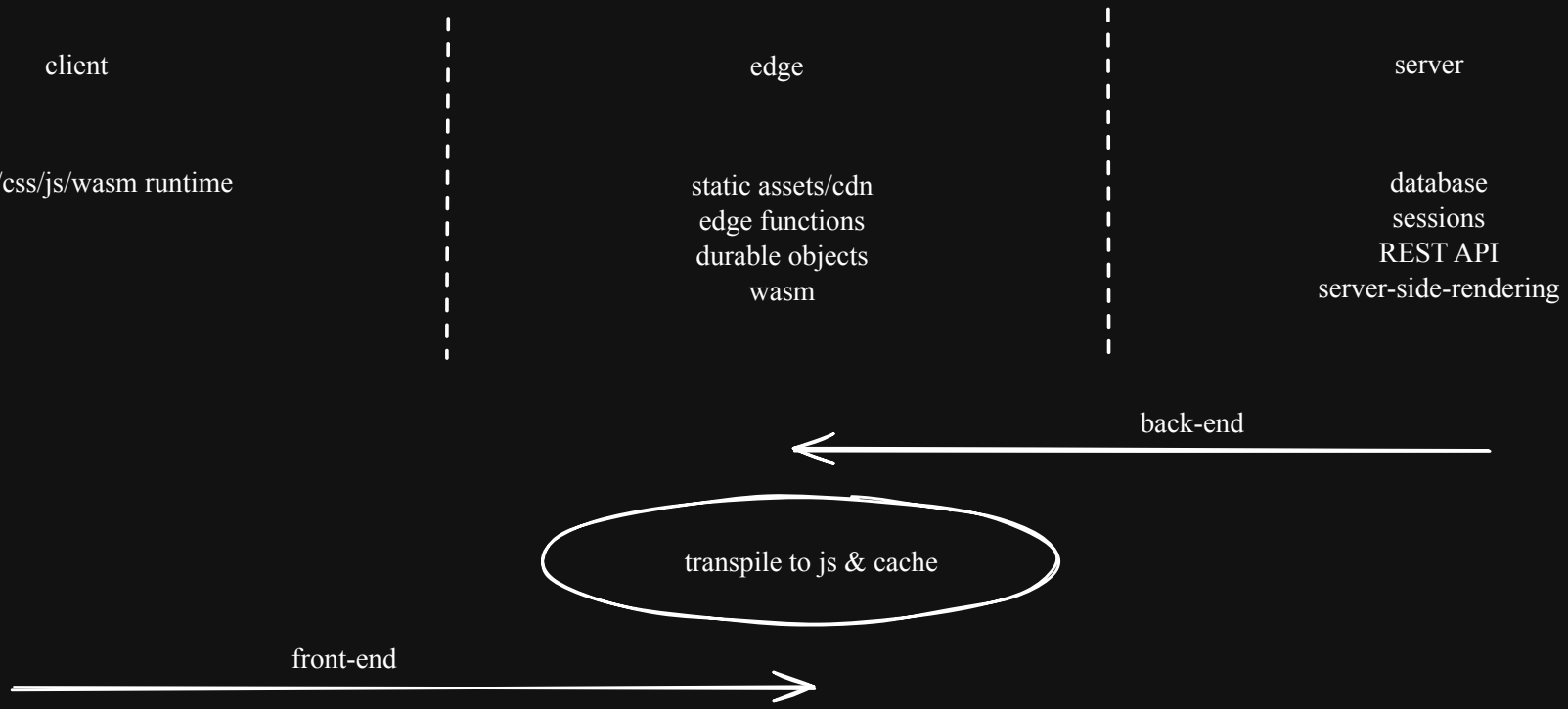
- Powerful but limiting

```
+ Full interactivity
+ Flexible
+ Easy opt-out to SPA or full-stack frameworks
+ Simple deployment
- Import-maps not as ergonomic as package.json
- Weird
```

References:

- https://www.npmjs.com/package/@hono/esbuild-transpiler
- https://bun.sh/docs/api/transpiler

Where could we possibly go with this?



Thank you!

Copenhagen JS - 06 Mar 2025

Rohit Sharma

rohit.sh | github.com/r0hitsharma