AKRAINO EDGE STACK

NextArch FOUNDATION

Realtime Collaboration on Web

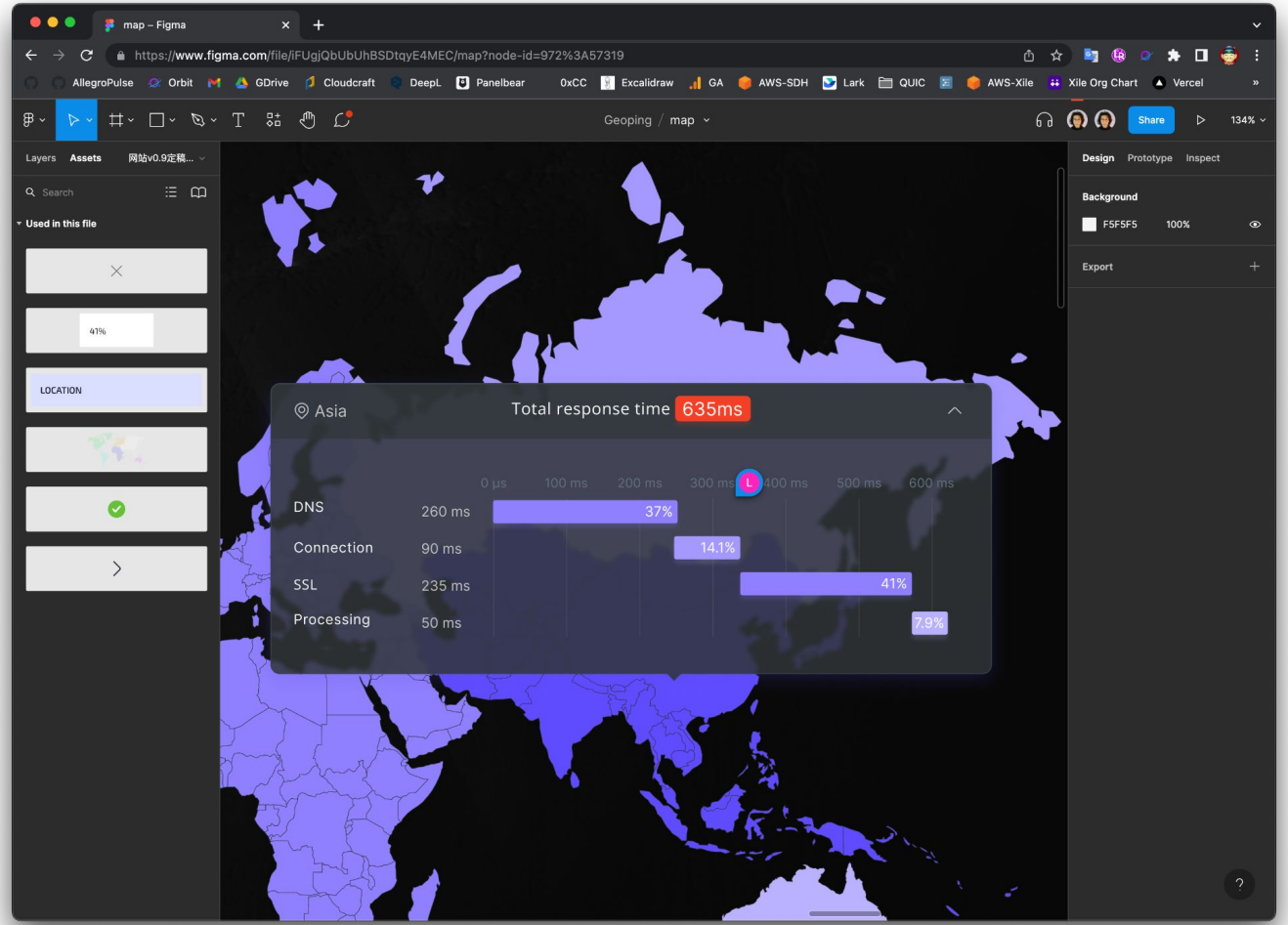
C.C. @ YoMo

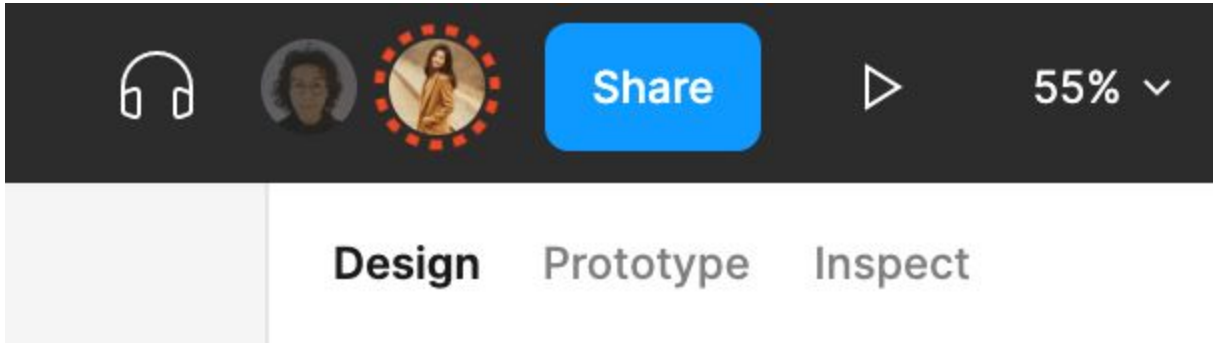
https://github.com/yomorun/yomo

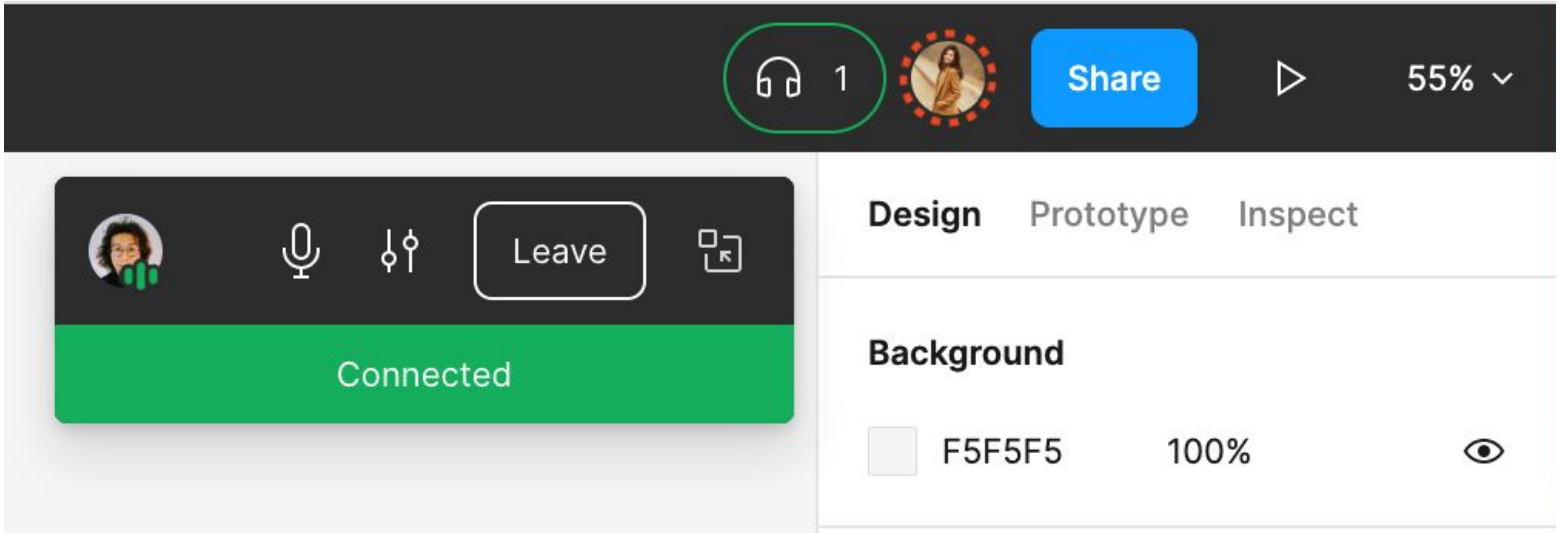
Nov 25 2021

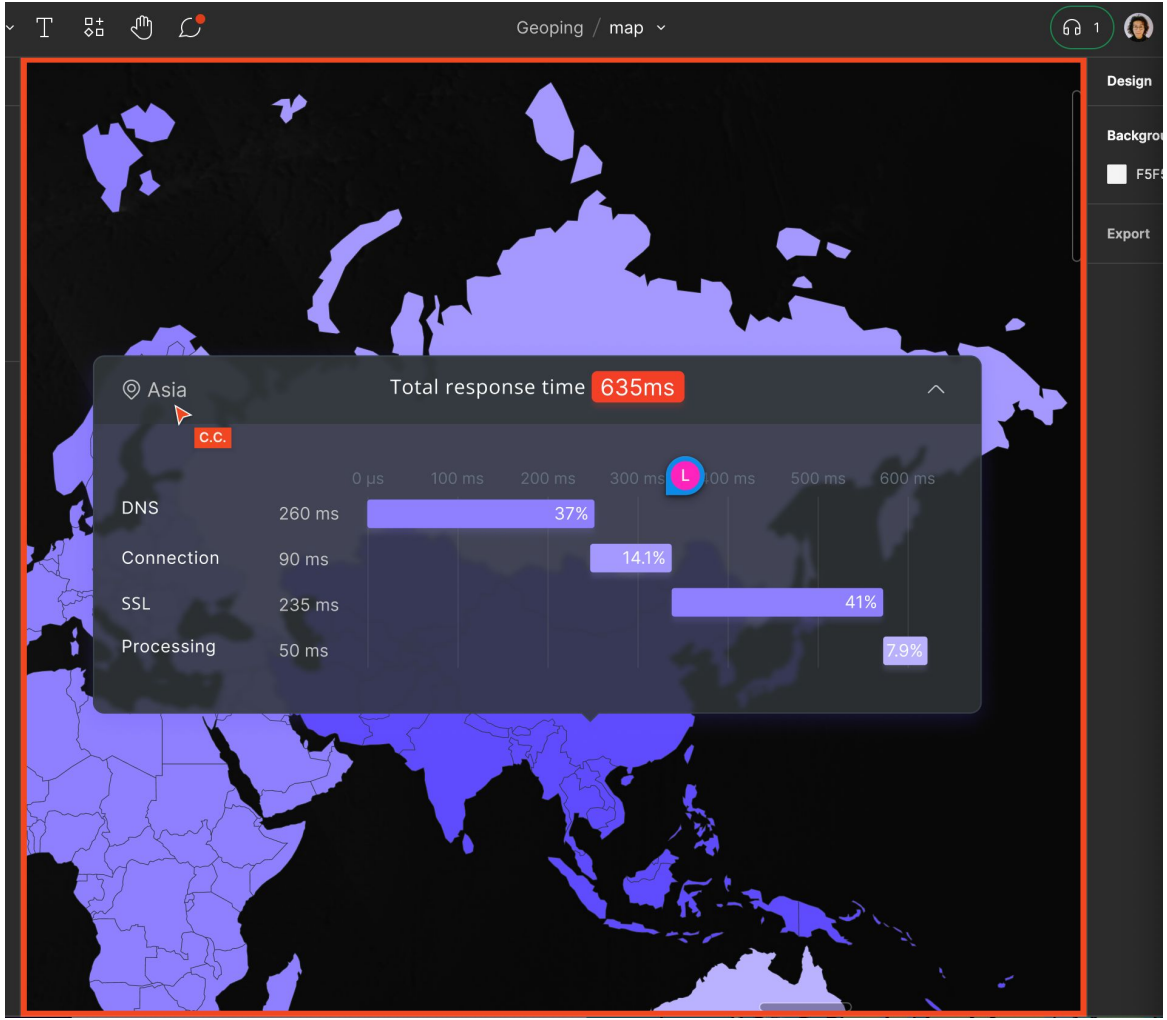
YoMo

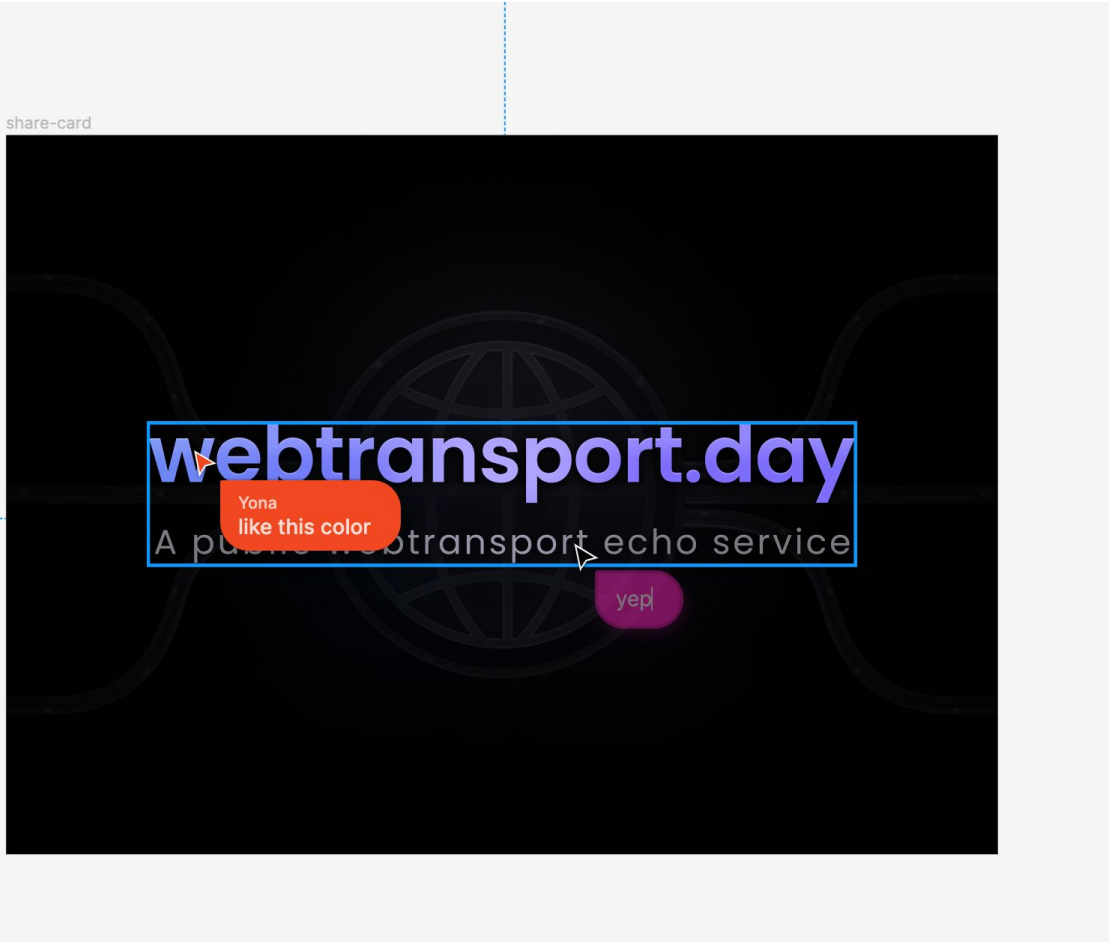
https://github.com/yomorun/yomo

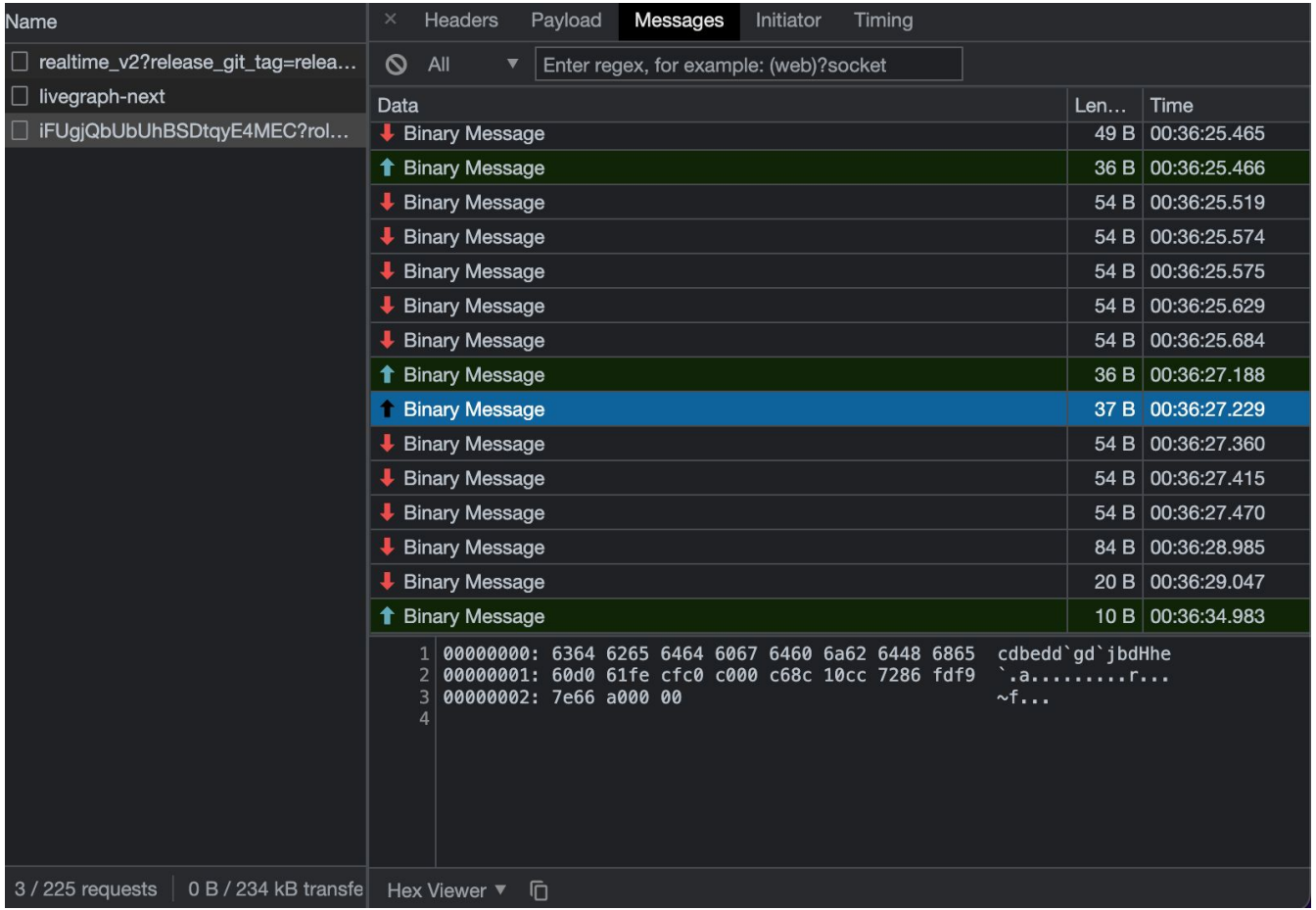


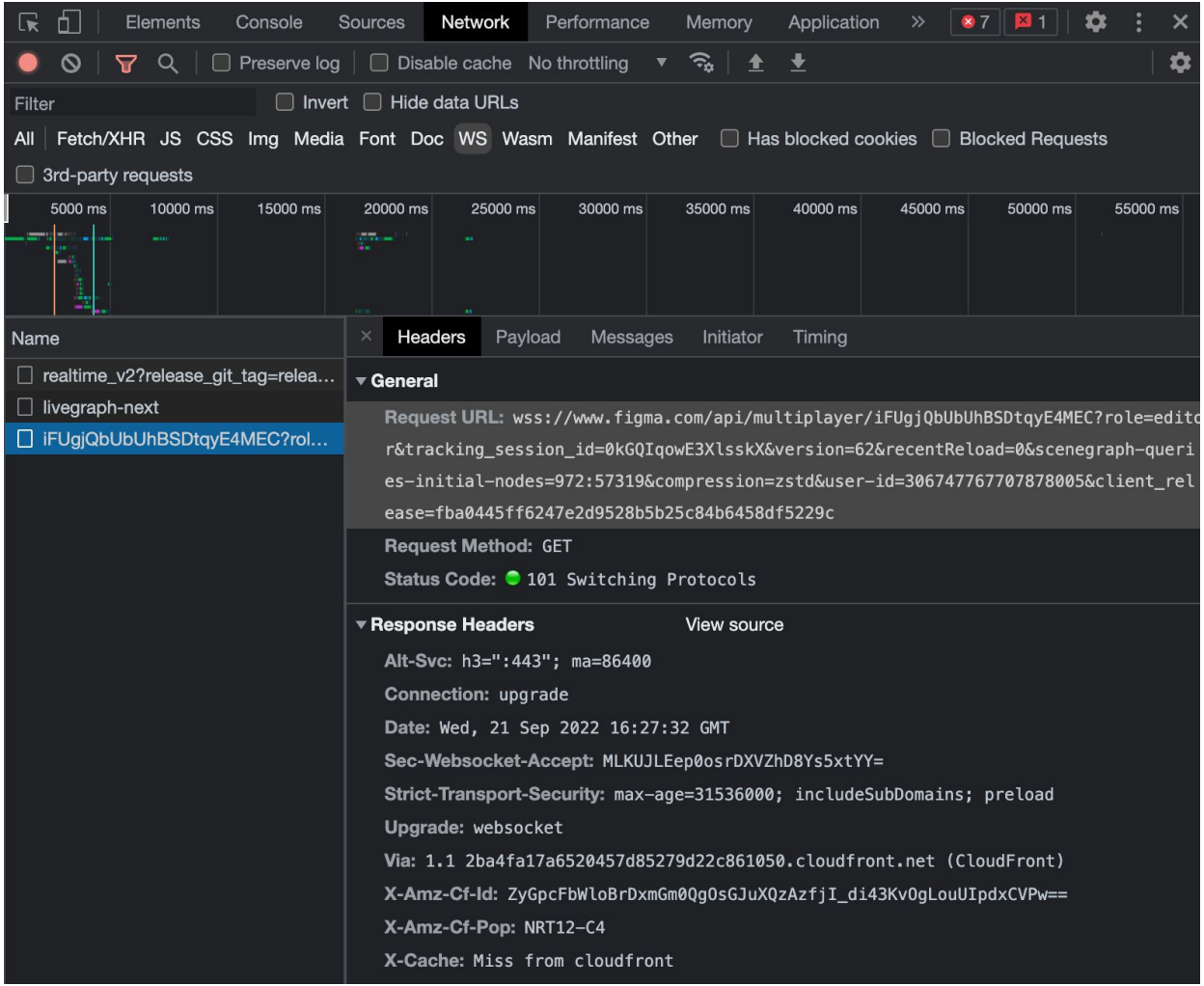

Share
55%
Design
Prototype
Inspect


1
Share
55%
Leave
Connected
Design
Prototype
Inspect
Background
F5F5F5
100%

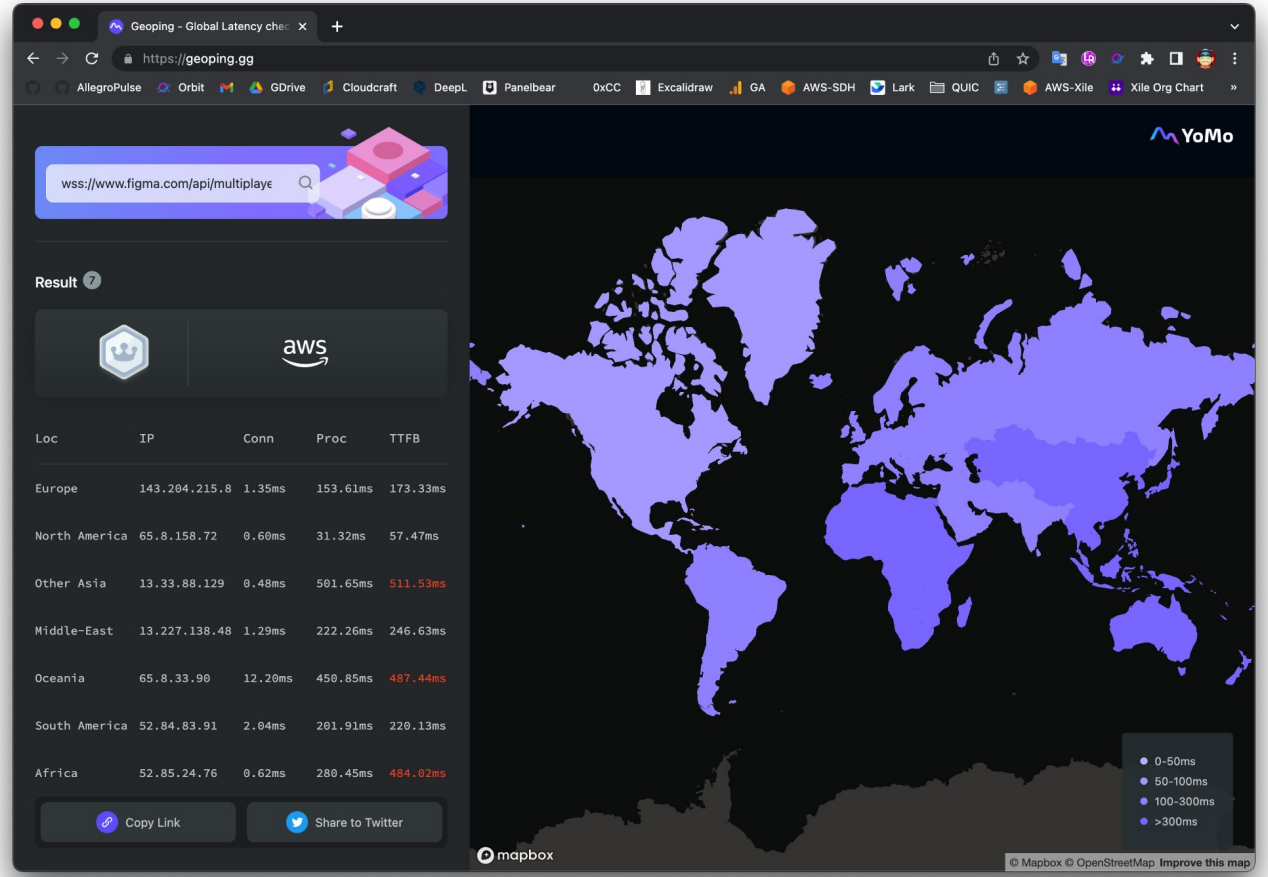



share-card
webtransport.day
Yona
like this color
A public webtransport echo service
yep

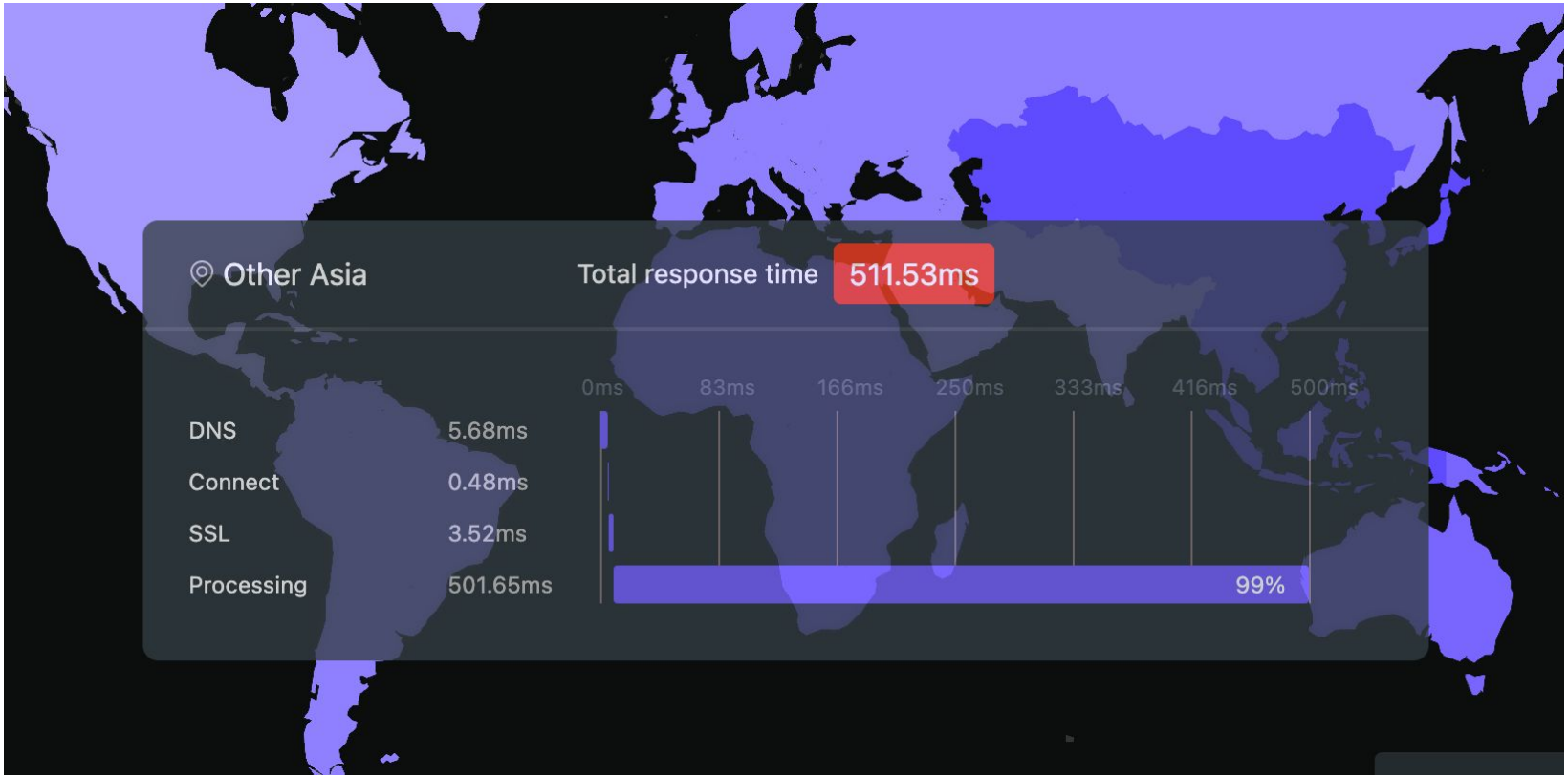



Elements
Console
Sources
Network
Performance
Memory
Application
7
1
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
5000 ms
10000 ms
15000 ms
20000 ms
25000 ms
30000 ms
35000 ms
40000 ms
45000 ms
50000 ms
55000 ms
Name
realtime_v2?release_git_tag=relea…
livegraph-next
iFUgjQbUbUhBSDtqyE4MEC?rol…
Headers
Payload
Messages
Initiator
Timing
General
Request URL: wss://www.figma.com/api/multiplayer/iFUgjQbUbUhBSDtqyE4MEC?role=editor&tracking_session_id=0kGQIqowE3XlsskX&version=62&recentReload=0&scenegraph-queries-initial-nodes=972:57319&compression=zstd&user-id=306747767707878005&client_release=fba0445ff6247e2d9528b5b25c84b6458df5229c
Request Method: GET
Status Code: 101 Switching Protocols
Response Headers
View source
Alt-Svc: h3=":443"; ma=86400
Connection: upgrade
Date: Wed, 21 Sep 2022 16:27:32 GMT
Sec-Websocket-Accept: MLKUJLEep0osrDXVZhD8Ys5xtYY=
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Upgrade: websocket
Via: 1.1 2ba4fa17a6520457d85279d22c861050.cloudfront.net (CloudFront)
X-Amz-Cf-Id: ZyGpcFbWloBrDxmGm0QgOsGJuXQzAzfjI_di43KvOgLouUIpdxCVPw==
X-Amz-Cf-Pop: NRT12-C4
X-Cache: Miss from cloudfront

https://geoping.gg



https://yomo.run


Other Asia
Total response time
511.53ms
0ms
83ms
166ms
250ms
333ms
416ms
500ms
DNS
5.68ms
Connect
0.48ms
SSL
3.52ms
Processing
501.65ms
99%

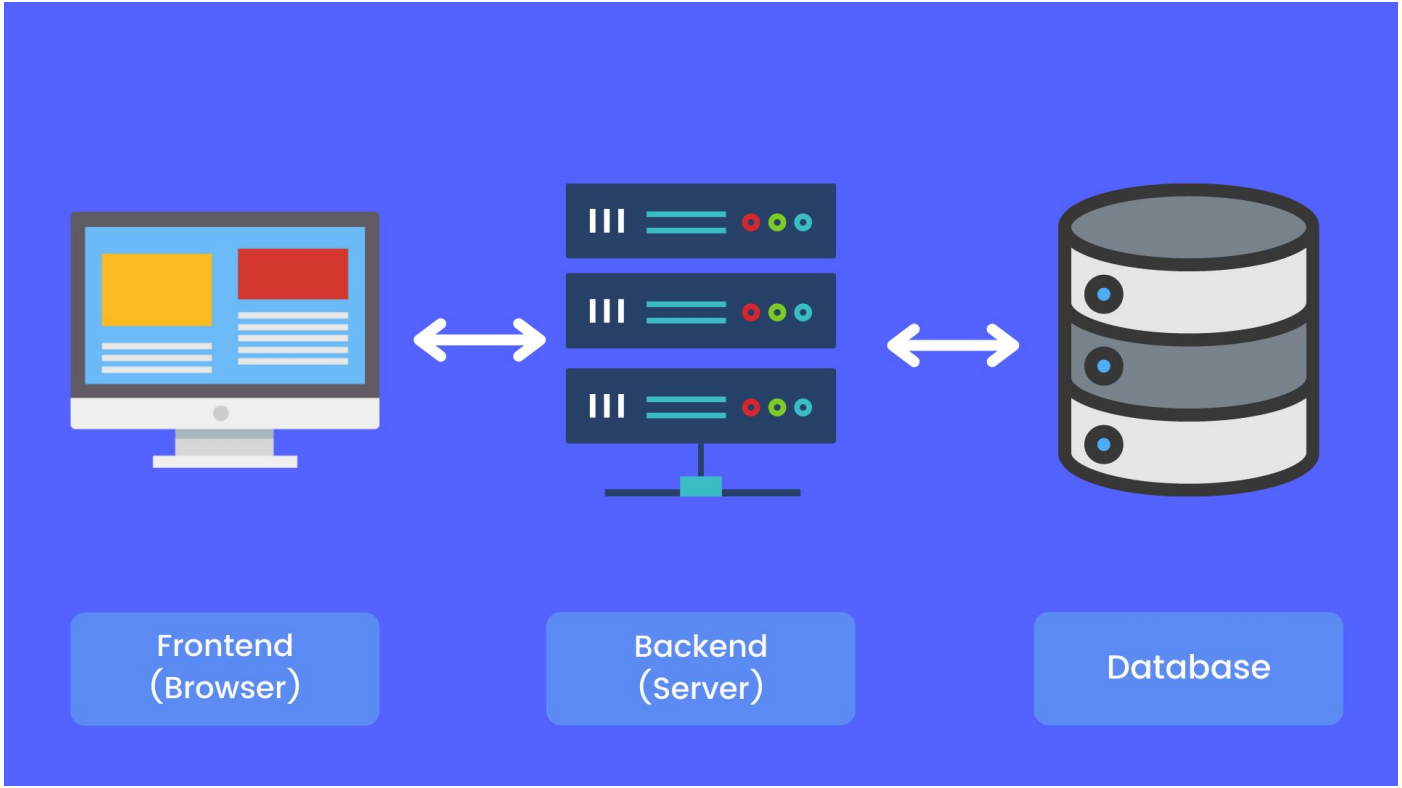
Loc	IP	Conn	Proc	TTFB
Europe	143.204.215.8	1.35ms	153.61ms	173.33ms
North America	65.8.158.72	0.60ms	31.32ms	57.47ms
Other Asia	13.33.88.129	0.48ms	501.65ms	511.53ms
Middle-East	13.227.138.48	1.29ms	222.26ms	246.63ms
Oceania	65.8.33.90	12.20ms	450.85ms	487.44ms
South America	52.84.83.91	2.04ms	201.91ms	220.13ms
Africa	52.85.24.76	0.62ms	280.45ms	484.02ms

https://nextjs.org/conf

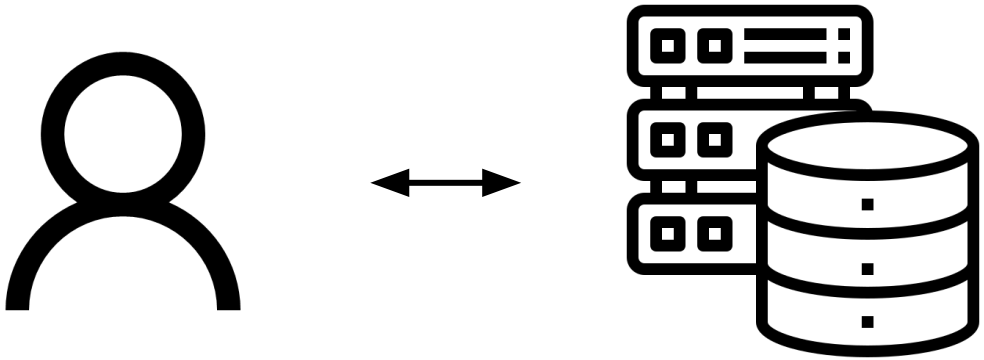
https://yomo.run

https://nextjs.org/conf

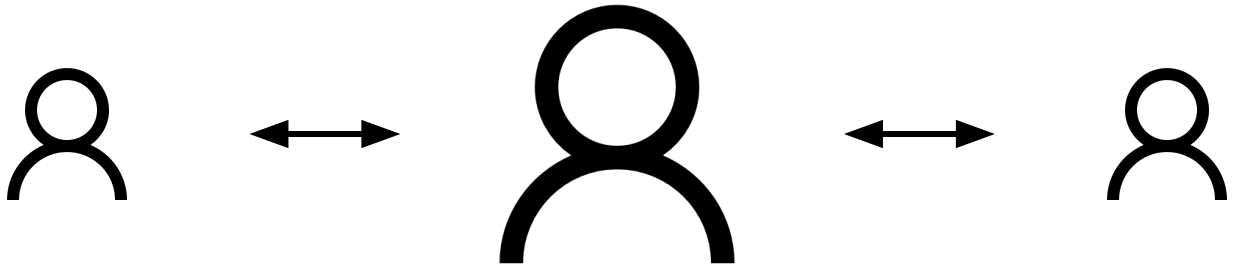
In the past decades



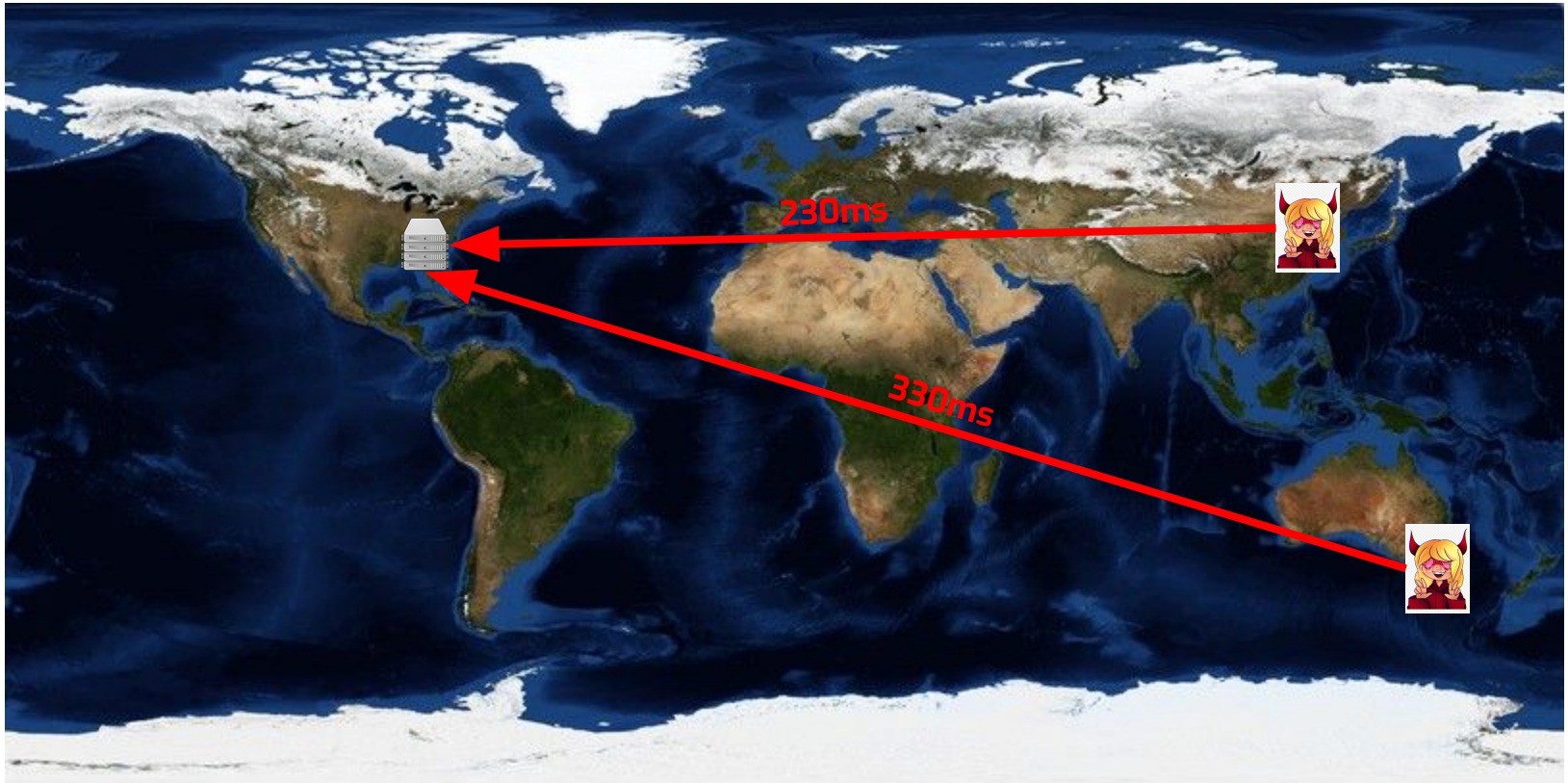
User - Database

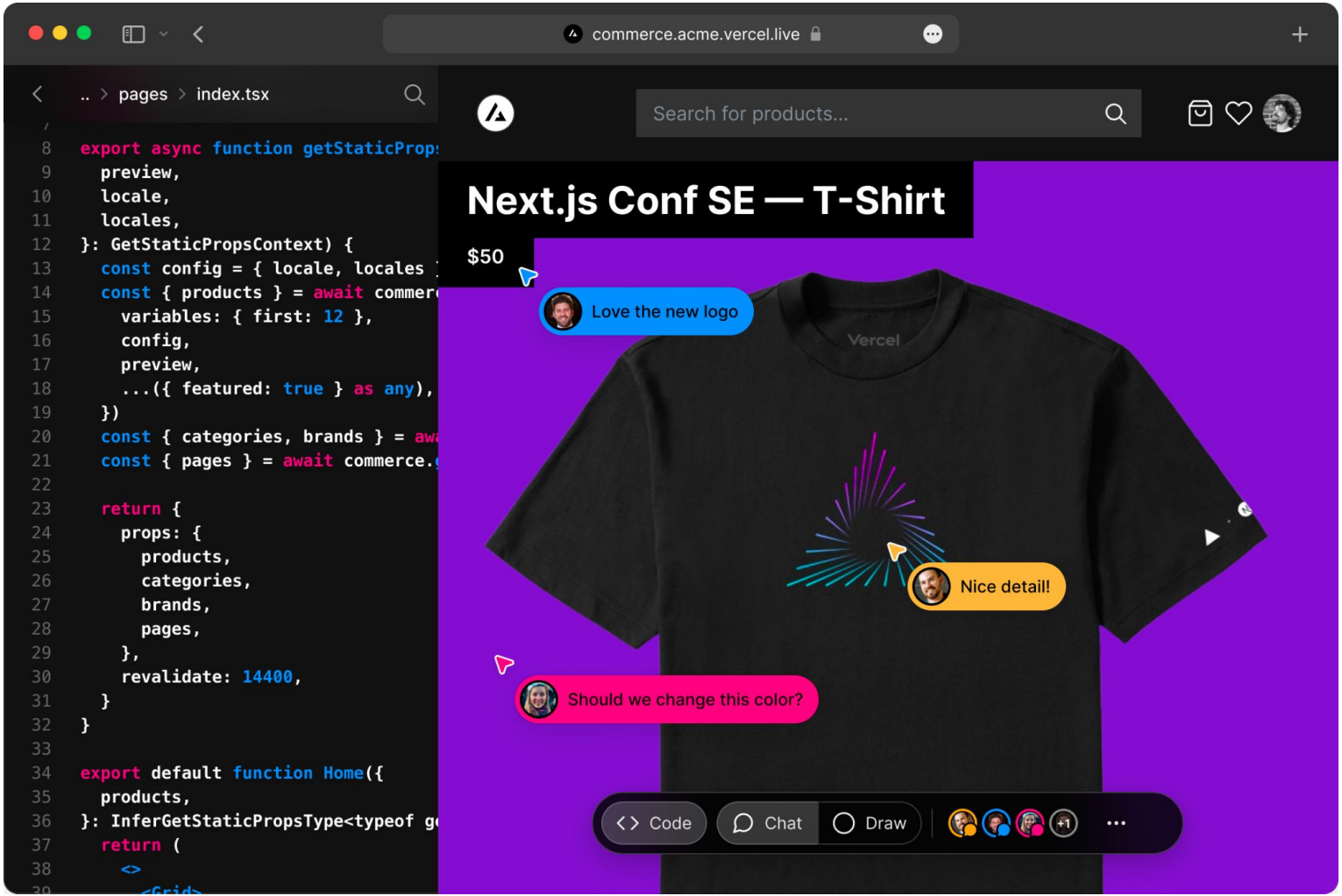


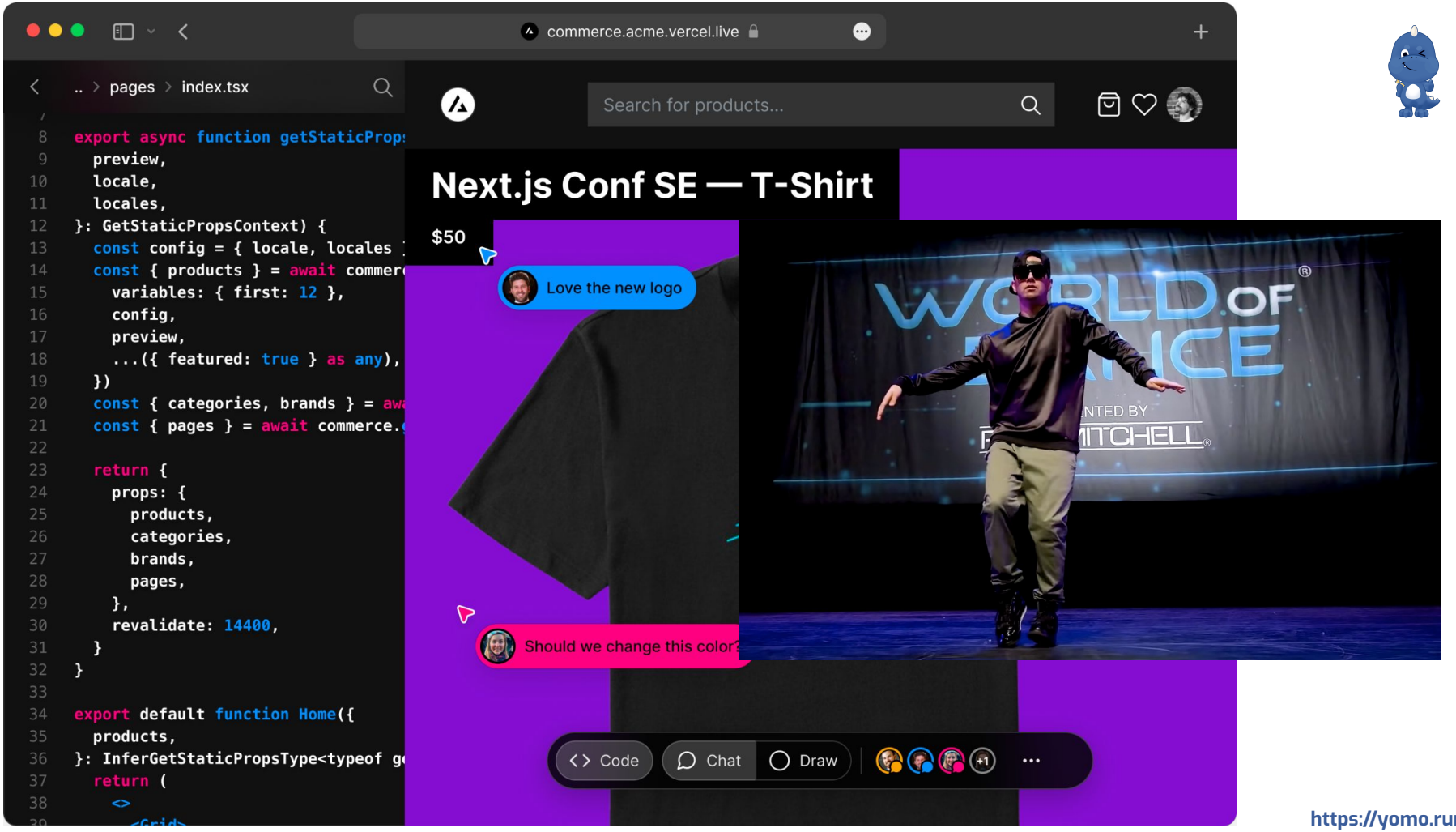
User - User



API & Database, Distributed ?







https://yomo.run

Web is migrating to the edge

Hydration: React 18, SSR, Streaming

reactwg / react-18 Public

Code | Pull requests | Discussions | Actions | Security | Insights

New Suspense SSR Architecture in React 18 #37

gaearon announced in **Deep Dive**

gaearon on Jun 5 Maintainer — edited

Overview

React 18 will include architectural improvements to React server-side rendering (SSR) performance. These improvements are substantial and are the culmination of several years of work. Most of these improvements are behind-the-scenes, but there are some opt-in mechanisms you'll want to be aware of, especially if you *don't* use a framework.

The primary new API is `renderToPipeableStream`, which you can read about in Upgrading to React 18 on the Server. We plan to write more about it in detail as it's not final and there are things to work out.

The primary existing API is `<Suspense>`.

This page is a high-level overview of the new architecture, its design, and the problems it solves.

SSR Streaming

Server-Side Streaming

Concurrent features in React 18 include built-in support for server-side Suspense and SSR streaming support. This allows you to server-render pages using HTTP streaming. This is an experimental feature in Next.js 12, but once enabled, SSR will use the same strict runtime as Middleware.

To enable, use the experimental flag `concurrentFeatures: true`:

```
// next.config.js
module.exports = {
  experimental: {
    concurrentFeatures: true
  }
}
```

Vercel

The future of edge computing is here

Now you can have the benefits of dynamic at the speed of static with Vercel Edge Functions.

Learn about Edge Functions

Say goodbye to cold boots

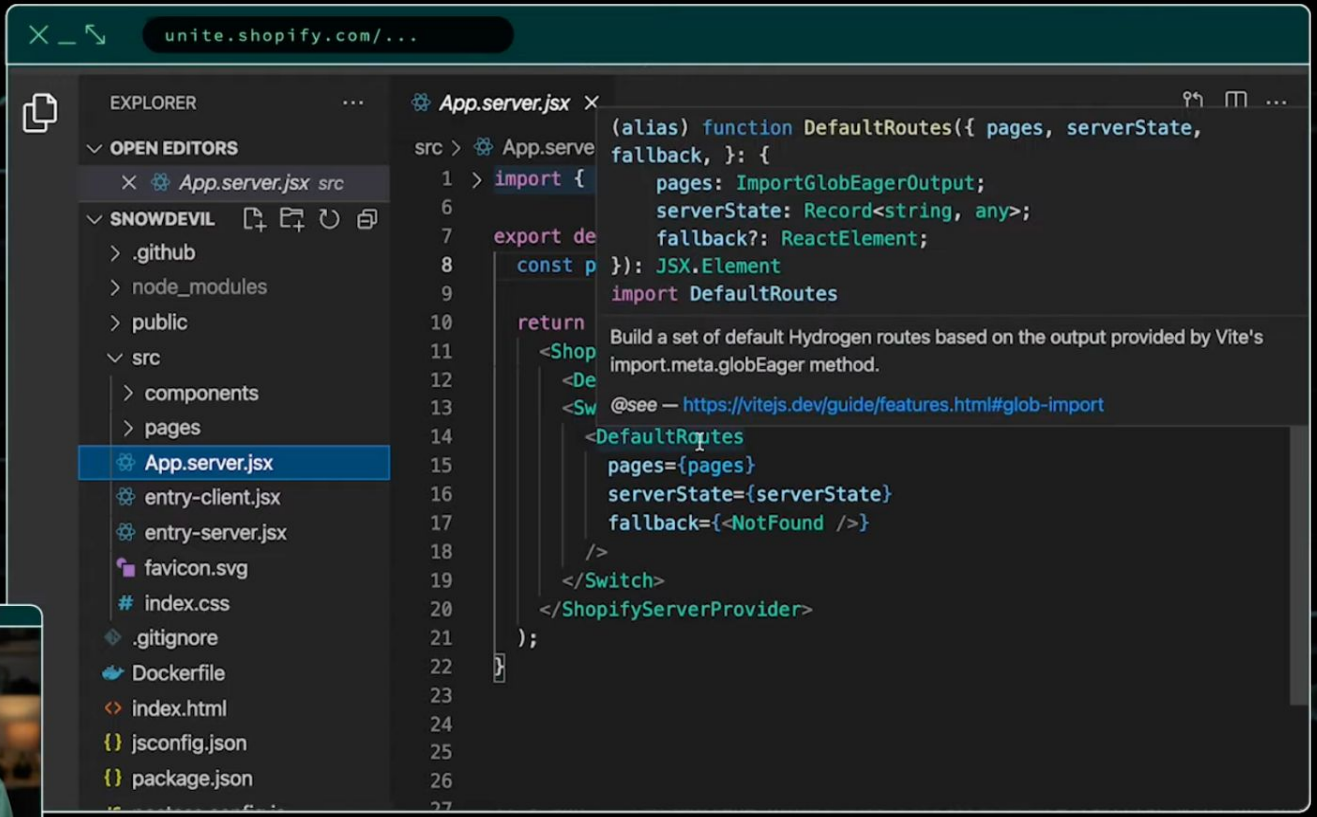
With Middleware deployed to Vercel, run globally-deployed functions, without any configuration. Even better? Your Next.js application will be optimized for the best performance every time.

De-risk experimentation

From feature flags to A/B testing, Middleware allows your developers to dynamically reconfigure routing at runtime to remove performance impacts and de-risk experimentation.

Make it personalized and fast

Reduce reliance on client-side JavaScript by deploying on the server without any performance tradeoffs. Your team can now serve users the right experience the first time with just code!



implements react router
from hydrogen we get a

Shopify Oxygen

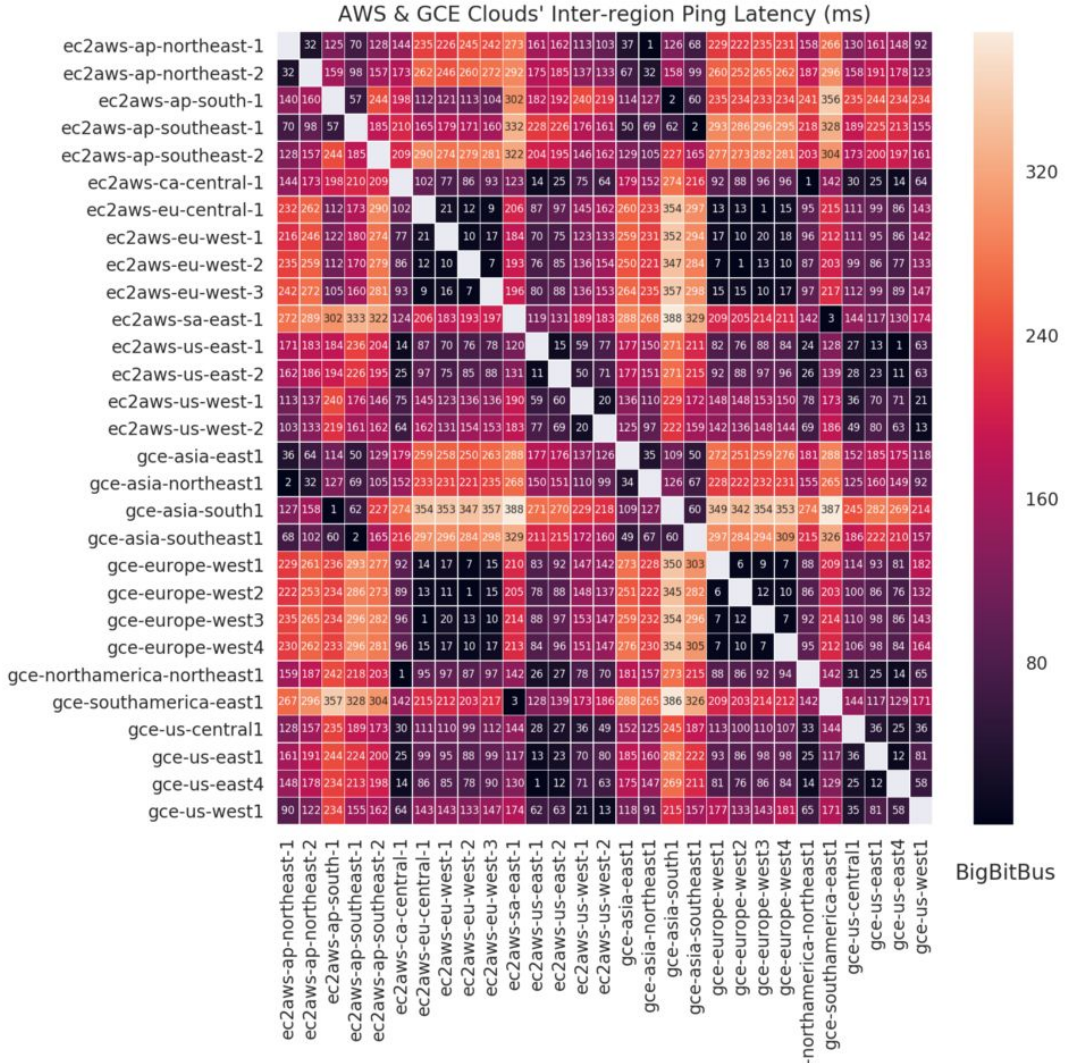
Web is migrating to the edge, but realtime still not

Region-to-Region

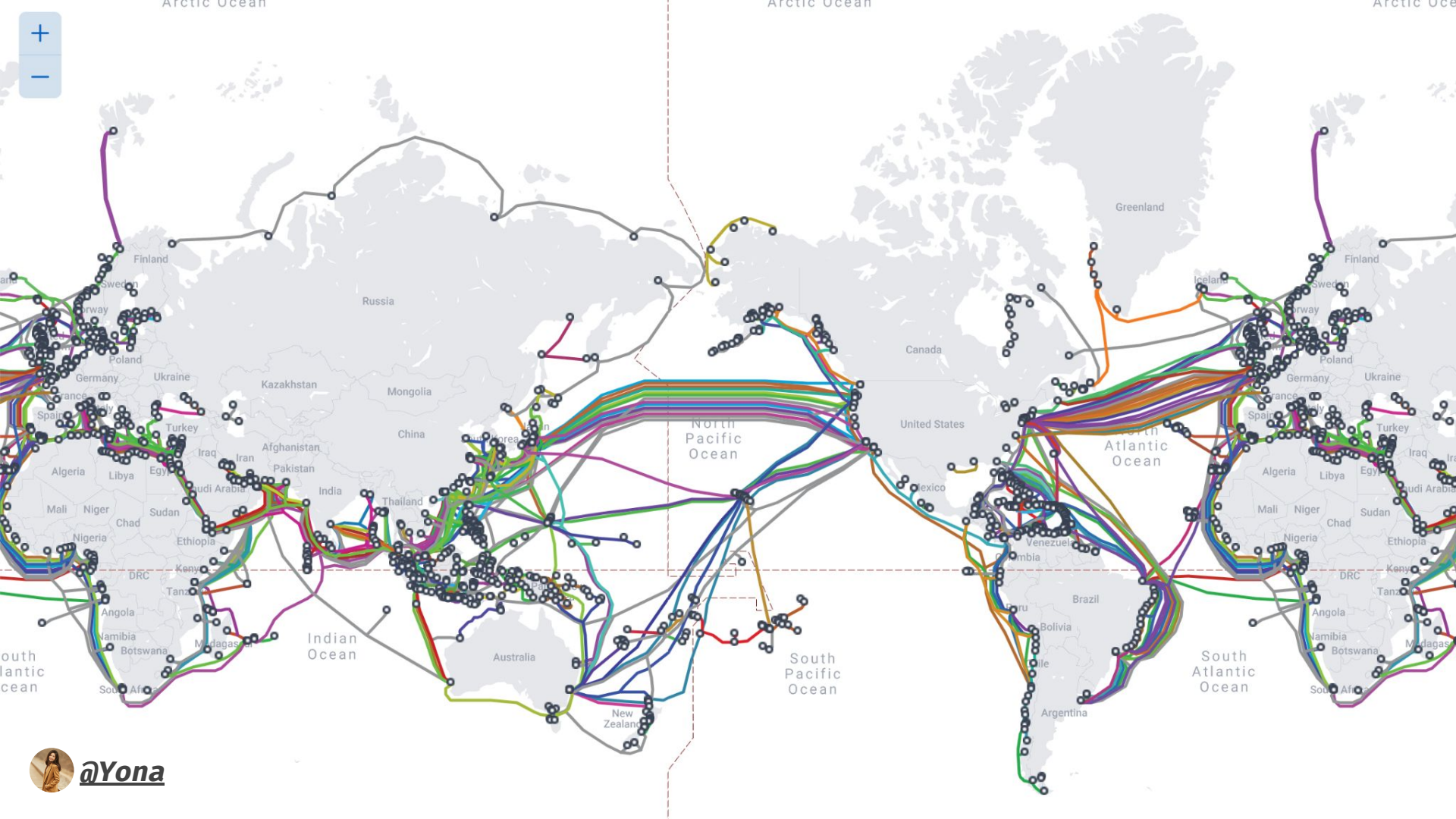
+ Add Source City

+ Add Destination City

	Barcelona	Hangzhou	Paris	Shanghai	Tokyo	Toronto	Washington
Amsterdam	31.509ms	209.917ms	11.352ms	197.382ms	231.31ms	91.629ms	82.043ms
Auckland	268.951ms	505.796ms	303.431ms	407.537ms	190.376ms	259.366ms	250.843ms
Copenhagen	39.303ms	272.211ms	23.563ms	248.208ms	231.074ms	111.46ms	103.301ms
Dallas	127.31ms	212.052ms	120.314ms	200.64ms	133.205ms	43.383ms	36.068ms
Frankfurt	35.154ms	257.19ms	11.317ms	247.415ms	218.633ms	101.32ms	94.13ms
Hangzhou	N/A	N/A	234.481ms	N/A	447.397ms	N/A	N/A
London	36.727ms	275.612ms	8.449ms	239.426ms	237.192ms	92.216ms	78.112ms
Los Angeles	154.058ms	154.344ms	159.333ms	162.815ms	116.154ms	70.676ms	67.959ms
Moscow	83.38ms	225.522ms	50.12ms	203.097ms	264.424ms	131.561ms	126.384ms
New York	96.042ms	314.683ms	73.104ms	276.22ms	176.782ms	20.472ms	8.942ms
Paris	27.241ms	284.159ms	—	269.094ms	229.661ms	93.264ms	85.749ms
Shanghai	N/A	5.886ms	313.506ms	—	N/A	240.598ms	N/A
Stockholm	51.576ms	215.577ms	29.546ms	246.817ms	238.672ms	98.411ms	100.142ms
Tokyo	212.995ms	80.695ms	228.448ms	81.588ms	—	163.561ms	172.569ms

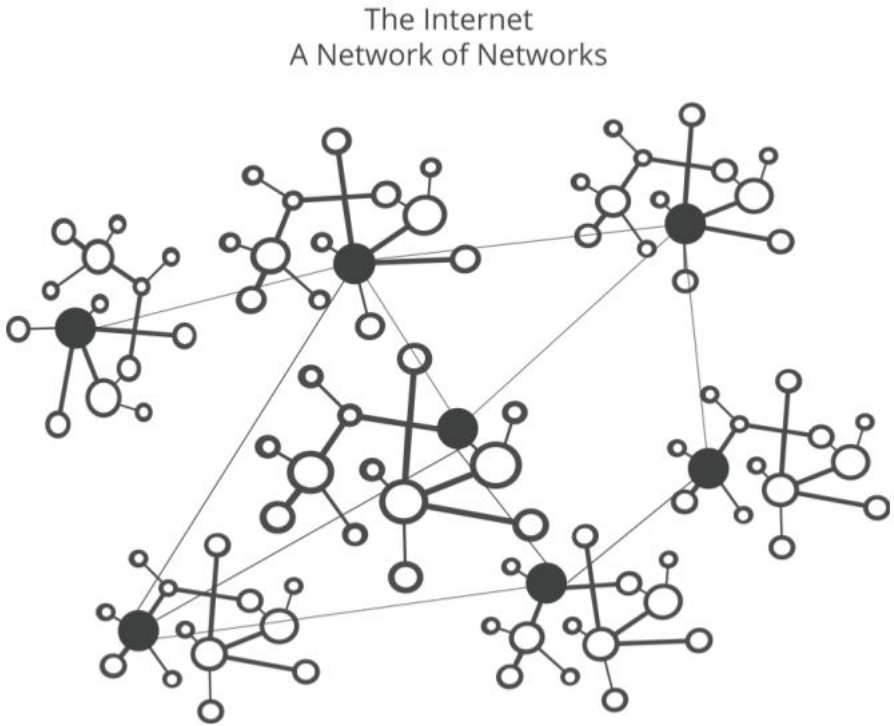

AWS & GCE Clouds' Inter-region Ping Latency (ms)
ec2aws-ap-northeast-1
ec2aws-ap-northeast-2
ec2aws-ap-south-1
ec2aws-ap-southeast-1
ec2aws-ap-southeast-2
ec2aws-ca-central-1
ec2aws-eu-central-1
ec2aws-eu-west-1
ec2aws-eu-west-2
ec2aws-eu-west-3
ec2aws-sa-east-1
ec2aws-us-east-1
ec2aws-us-east-2
ec2aws-us-west-1
ec2aws-us-west-2
gce-asia-east1
gce-asia-northeast1
gce-asia-south1
gce-asia-southeast1
gce-europe-west1
gce-europe-west2
gce-europe-west3
gce-europe-west4
gce-northamerica-northeast1
gce-southamerica-east1
gce-us-central1
gce-us-east1
gce-us-east4
gce-us-west1
320
240
160
80
BigBitBus

@Yona


Arctic Ocean
Arctic Ocean
Arctic Ocean
Greenland
Finland
Russia
Canada
Poland
Germany
Ukraine
Kazakhstan
Mongolia
China
United States
Turkey
Iraq
Iran
Afghanistan
Pakistan
India
Thailand
Algeria
Libya
Egypt
Saudi Arabia
Mali
Niger
Sudan
Chad
Nigeria
Ethiopia
DRC
Kenya
Tanzania
Angola
Namibia
Botswana
Madagascar
South Africa
Indian Ocean
Australia
New Zealand
North Pacific Ocean
South Pacific Ocean
Mexico
Venezuela
Colombia
Brazil
Bolivia
Peru
Chile
Argentina
North Atlantic Ocean
South Atlantic Ocean
Iceland
Sweden
Norway
France
Spain
Italy

@Yona

YoMo helps organize global workload



New realtime communication protocol

WebTransport

web.dev Learn Measure Blog Case studies About

Using WebTransport

WebTransport is an API offering low-latency, bidirectional, client-server messaging. Learn more about its use cases, and how to give feedback about the future of the implementation.

Jun 8, 2020 — Updated Jul 18, 2022

Available in: English, Русский, 中文, 日本語, and 한국어

Jeff Posnick

Twitter GitHub Glitch Homepage

What's WebTransport?

WebTransport is a web API that uses the HTTP/3 protocol as a bidirectional transport. It's intended for two-way communications between a web client and an HTTP/3 server. It supports sending data both unreliably via its datagram APIs, and reliably via its streams APIs.

Unreliable Support

Datagrams are ideal for sending and receiving data that do not need strong delivery guarantees. Individual packets of data are limited in size by the maximum transmission unit (MTU) of the underlying connection, and may or may not be transmitted successfully, and if they are transferred, they may arrive in an arbitrary order. These characteristics make the datagram APIs ideal for low-latency, best-effort data transmission. You can think of datagrams as user datagram protocol (UDP) messages, but encrypted and congestion-controlled.

Streams API

The streams APIs, in contrast, provide reliable, ordered data transfer. They're well-suited to scenarios where you need to send or receive one or more streams of ordered data. Using multiple WebTransport streams is analogous to establishing multiple TCP connections, but since HTTP/3 uses the lighter-weight QUIC protocol under the hood, they can be opened and closed without as much overhead.

Streams API - SendStream

```
// Send two Uint8Arrays to the server.
const stream = await transport.createSendStream();
const writer = stream.writable.getWriter();
const data1 = new Uint8Array([65, 66, 67]);
const data2 = new Uint8Array([68, 69, 70]);
writer.write(data1);
writer.write(data2);
try {
  await writer.close();
  console.log('All data has been sent.');
} catch (error) {
  console.error(`An error occurred: ${error}`);
}
```

Streams API - ReceiveStream

```
async function readFrom(receiveStream) {
  const reader = receiveStream.readable.getReader();
  while (true) {
    const {done, value} = await reader.read();
    if (done) {
      break;
    }
    // value is a Uint8Array
    console.log(value);
  }
}
```

```
const rs = transport.receiveStreams();
const reader = rs.getReader();
while (true) {
  const {done, value} = await reader.read();
  if (done) {
    break;
  }
  // value is an instance of ReceiveStream
  await readFrom(value);
}
```

Streams API - BidirectionalStream

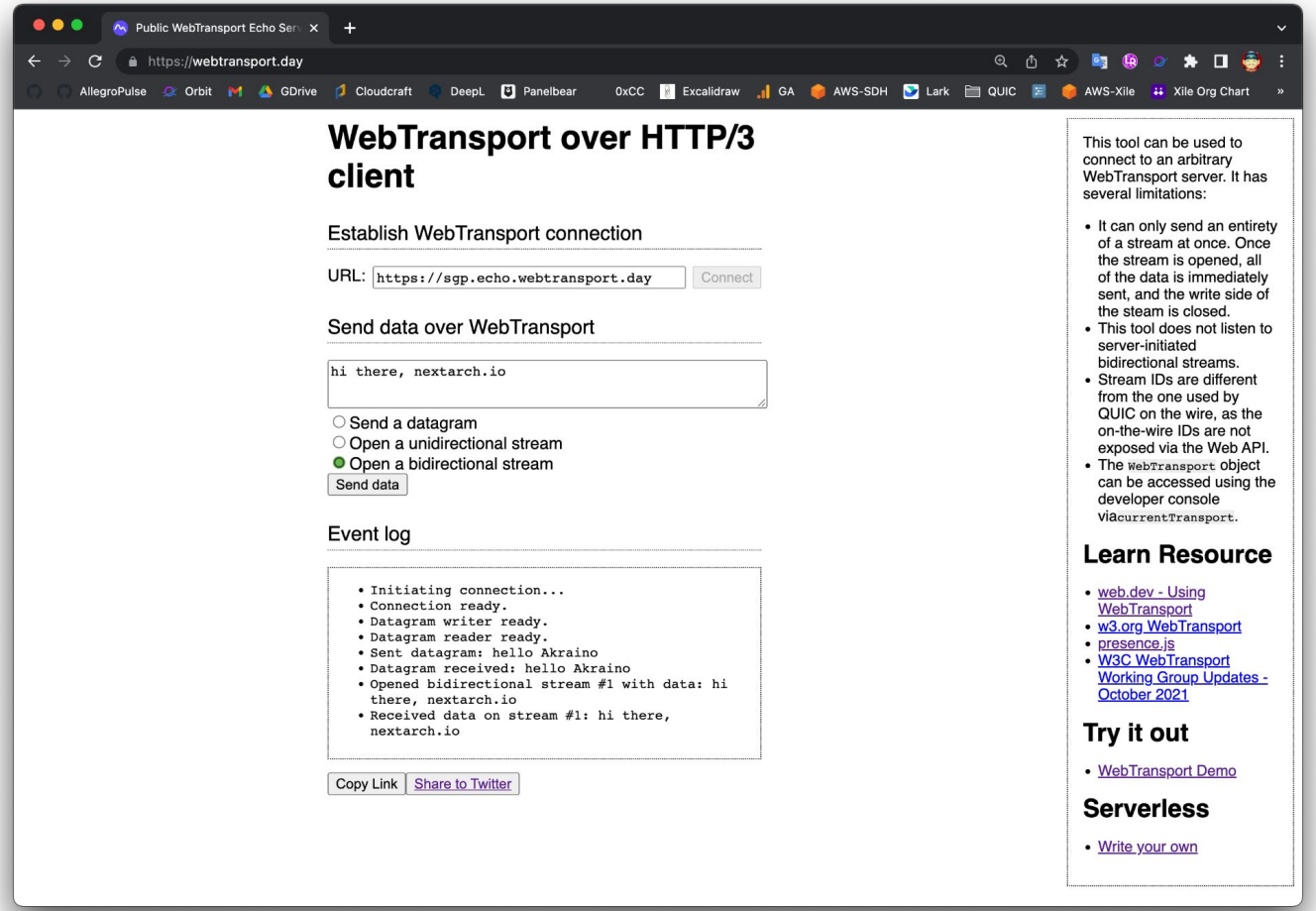
```
const rs = transport.receiveBidrectionalStreams();
const reader = rs.getReader();
while (true) {
  const {done, value} = await reader.read();
  if (done) {
    break;
  }
  // value is a BidirectionalStream
  // value.readable is a ReadableStream
  // value.writable is a WritableStream
}
```

Try it out

The best way to experiment with WebTransport is to start up a compatible HTTP/3 server. You can then use this page with a basic JavaScript client to try out client/server communications.

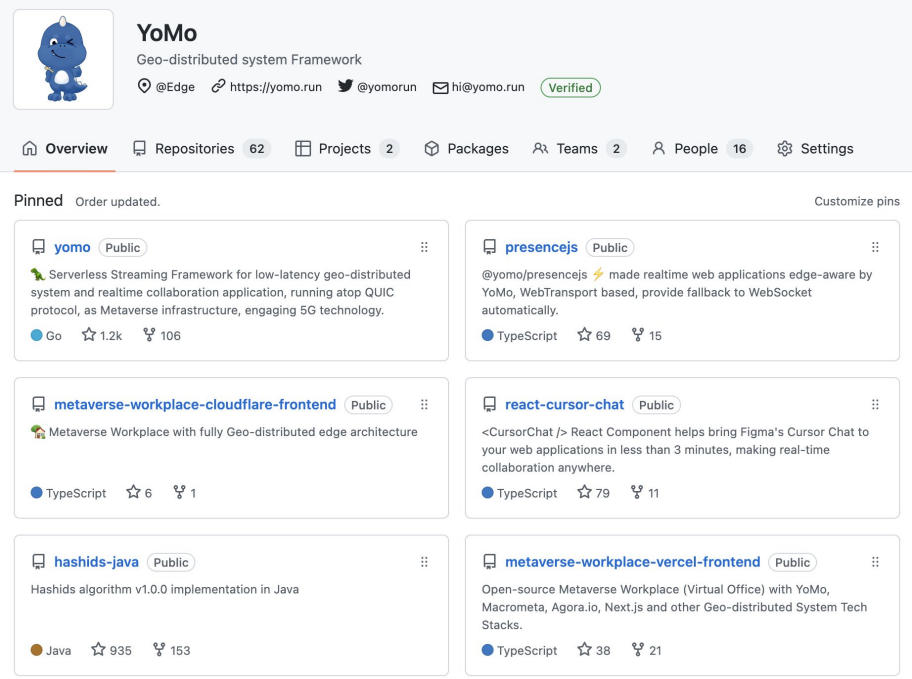
Additionally, a community-maintained echo server is available at webtransport.day.

https://webtransport.day



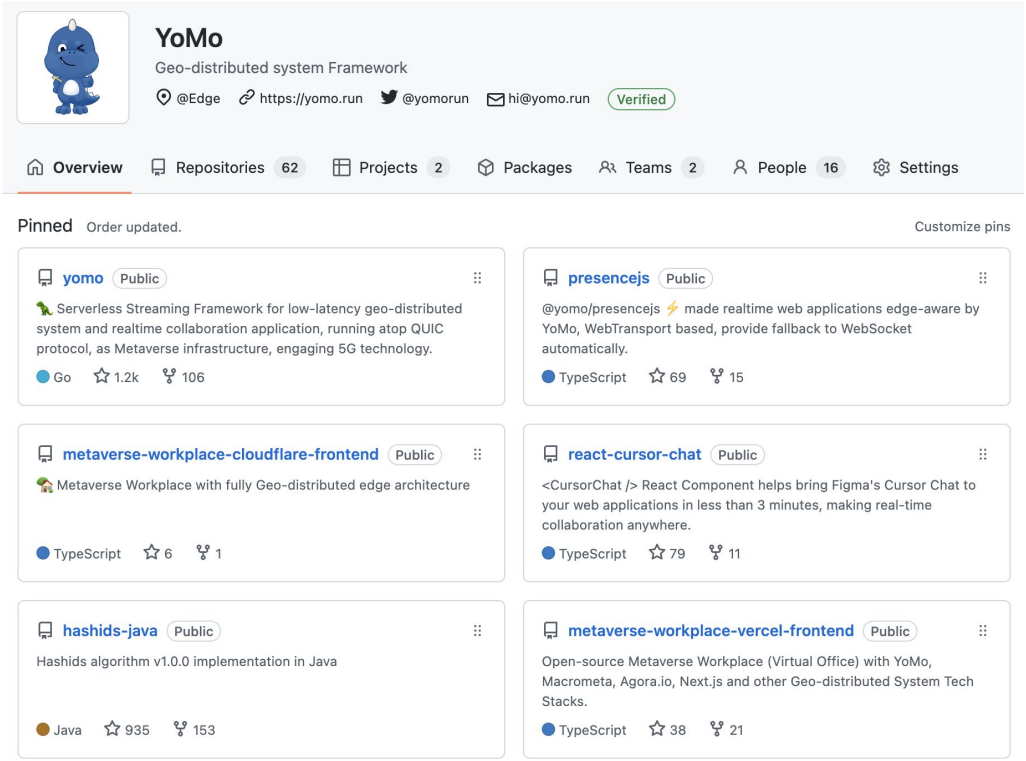
https://yomo.run

What we are crafting



- **WebTransport-polyfill** works with web browsers other than Chrome
- Provides developers with an easy-to-use **programming interface** rather than standard Streaming API
- Introduces a **wire protocol**, so every front-end developer doesn't need to build their communication specification.

What we are crafting



- **Unified backend** supports both WebTransport and WebSocket.
- Functions can be written in **WebAssembly** and deployed anywhere - browser, edge or cloud.

AKRAINO EDGE STACK

NextArch FOUNDATION

Thanks

https://github.com/yomorun/yomo

2022 Sep