

BALANCING PRIVACY & PERFORMANCE

Front-end Forward, 31 August 2017



@declanrek

STARTMAIL



MEET STARTMAIL

Secure webmail

Privacy is a basic human right

Privacy by design

STARTMAIL = SECURE AND PRIVATE WEBMAIL



BENEFITS WHY STARTMAIL? PRICING PRIVACY SUPPORT BLOG

ENGLISH ▾

LOG IN



Privacy is a basic human right

[Meet StartMail](#)

The easiest way to protect yourself from unwanted intrusion and mass surveillance.

[FREE 7-DAY TRIAL](#)

[SIGN UP](#)



[Compose](#)[Check mail](#)[search web \(new window\)](#)**Inbox** (204)[reply](#) [reply all](#) [forward](#) [print](#) [delete](#) [empty trash](#)[search mail](#)[Drafts](#)[Sent](#)[Trash](#)[Spam](#)[Manage folders](#)[Manage filters](#)[Support](#)[From:](#)[Subject:](#)[Received:](#)**This folder is empty**

No message selected

STARTMAIL

Full PGP encryption

Mailbox encryption(user vault)

IP address not stored and minimal logging

No user profile

Deleted = deleted

STARTMAIL

No cloud services

No webpage test

No pagespeed insights

No ngrok

Latency because no CDN

No native apps



Progressive



Responsive



Network
Independent



App-Like



Fresh



Safe



Discoverable



Re-engageable



Installable



Linkable

A PWA is

WEB TECHNOLOGY

The Web platform: Browser technologies

Core platform

- HTML
- DOM
- ECMAScript
- URL
- Fetch (includes CORS)
- XMLHttpRequest
- Encoding

CSS features

- CSS roadmap
- Animations
- Background-image options
- Border images
- Border radius (rounded corners)
- Box shadows
- Box sizing
- Cascading and inheritance
- Colors
- Compositing and Blending
- Device Adaptation
- Downloadable fonts (@font-face)
- Exclusions
- Feature queries (@supports)
- Filter Effects
- Fixed positioning (position:fixed)
- Flexible box layout (Flexbox)
- Font-feature settings
- Font loading
- Gradients
- Grid layout
- Hyphenation
- Image Values and Replaced Content
- Logical properties
- Masking
- Media Queries
- Motion paths
- Multiple-column layout
- Multiple backgrounds
- Opacity
- Overflow
- Pointer events

Graphics and typography

- Canvas
- WebGL
- SVG
- WOFF
- MathML
- Web Animations

Media

- Web Audio API
- WebVTT
- Media Source Extensions
- Media Fragments

Platform interaction, events, messaging

- Notifications API
- Cross-document messaging
- Channel messaging
- Fullscreen API
- Geofencing
- Geolocation
- Device Orientation
- Screen Orientation

Pseudo-elements

- Scroll Snap Points
- Selectors
- Shapes
- Sticky positioning (position:sticky)
- Style attributes
- Syntax
- Text decoration
- Text overflow
- Text shadows
- Transforms (2D)
- Transforms (3D)
- Transitions
- Values and Units
- Will Change
- Writing modes
- CSSOM (CSS Object Model)
- CSSOM View Module

UI Events (formerly DCM Events)

- Pointer Events
- Touch Events
- Pointer Lock
- Gamepad
- getUserMedia
- Battery Status
- Vibration
- Beacon
- HTML Media Capture (the capture attribute)
- Clipboard API and events

Storage and Files

- Storage (NavigatorStorage+StorageManager)
- Web Storage (localStorage)
- Indexed Database
- File API
- Blob URLs
- File Reader

Real-time communication

- WebRTC
- Object RTC (ORTC) API for WebRTC
- WebSocket protocol
- WebSocket API
- Server-Sent Events
- Push API

Web Components

- Custom Elements
- Shadow DOM
- Templates

HTML bonus features

- classList (DOMTokenList)
- dataset (data-* attributes)
- async for scripts
- defer for scripts
- Session-history management
- hashchange
- Sandboxed Iframe
- Drag and drop
- contentEditable
- HTML Editing APIs

ARIA

Performance optimization and analysis

- Web Workers
- Shared Workers
- Timing control for script-based animations
- Navigation Timing
- Page Visibility
- User Timing
- Performance Timeline
- High Resolution Time
- SIMD (Single instruction, multiple data)

Security and privacy

- Content Security Policy (CSP)
- Upgrade Insecure Requests
- Web Cryptography API
- Referrer policy
- Tracking Preference Expression (DNT)

Other core-platform bonus features

- Structured cloning
- Transferable objects
- Mutation observers
- Streams
- DOM Parsing and Serialization
- DOM XPath
- Quirks Mode
- Internationalization API
- Promises
- JSON parsing
- Typed Array
- Service Workers
- querySelector() method
- matches() method
- matchMedia() method
- data URLs

Various other

- Web IDL
- Link header
- Content-Disposition header

A feature history of the modern Web Platform

As new web APIs become
available, front-enders get more
responsibilities

Security was never my
responsibility until...

LATENCY



IP address or hostname

www.startmail.com

Ping



RESPONSE TIMES

100ms = Instantaneous

1 second = Uninterrupted flow

FAKE IT

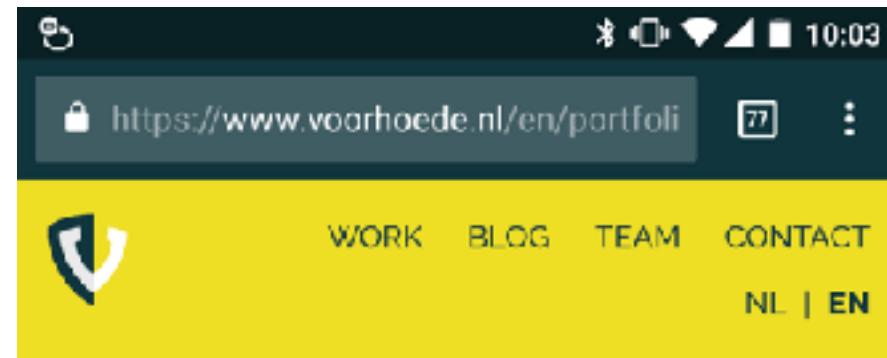


A network proxy in your
browser?

SERVICE WORKER

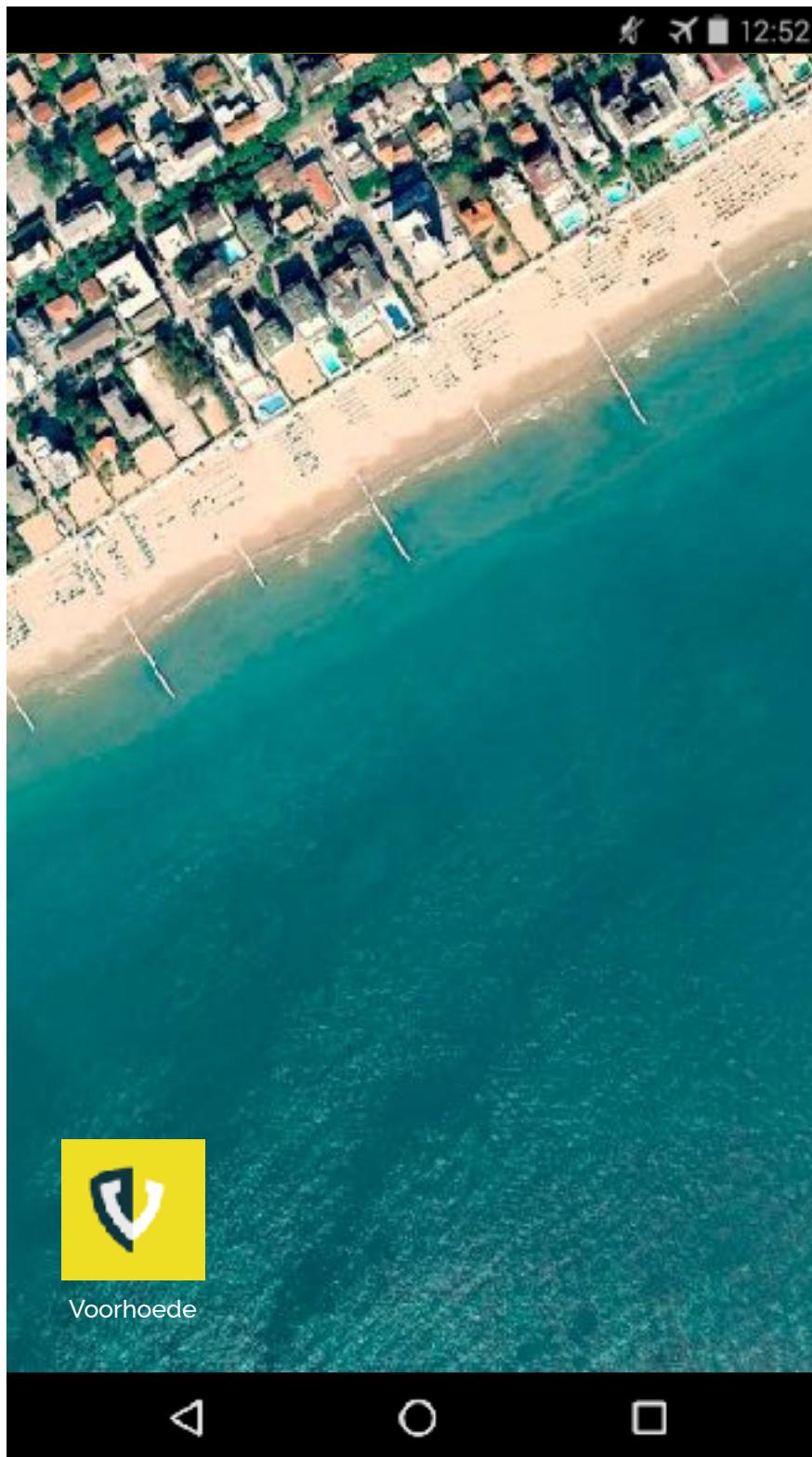
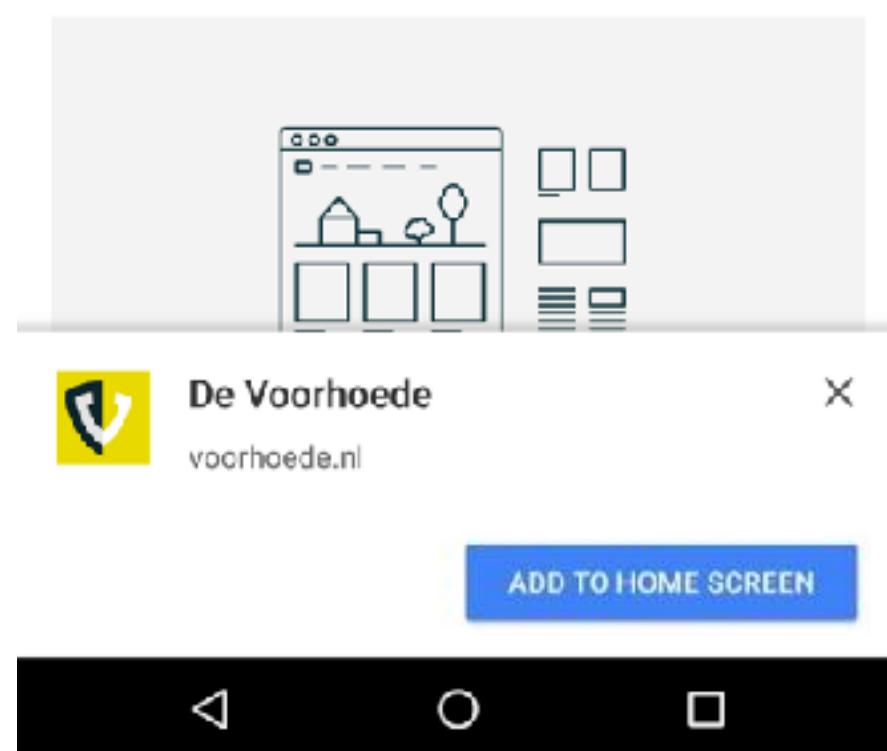


VOORHOEDE.NL

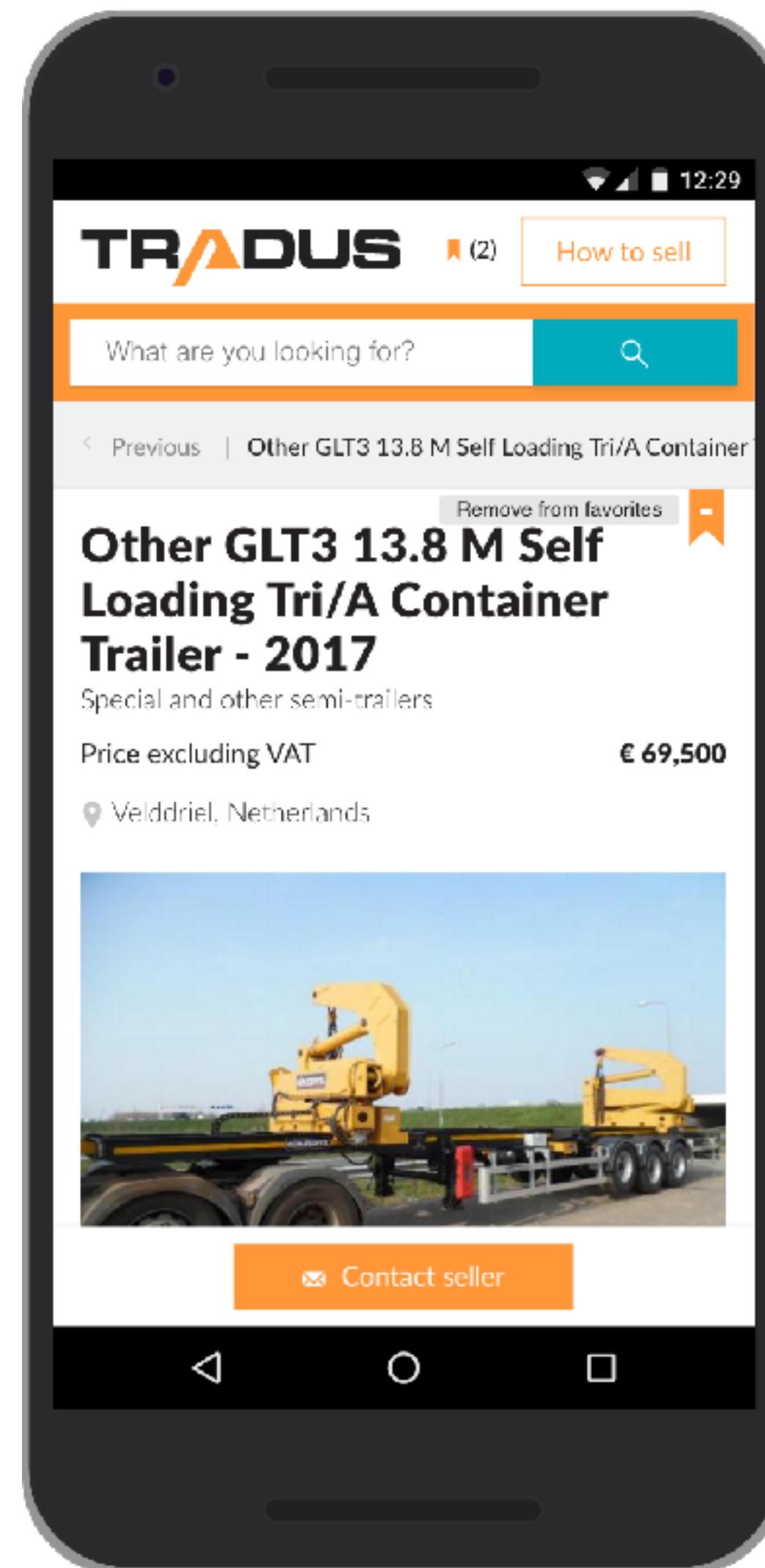
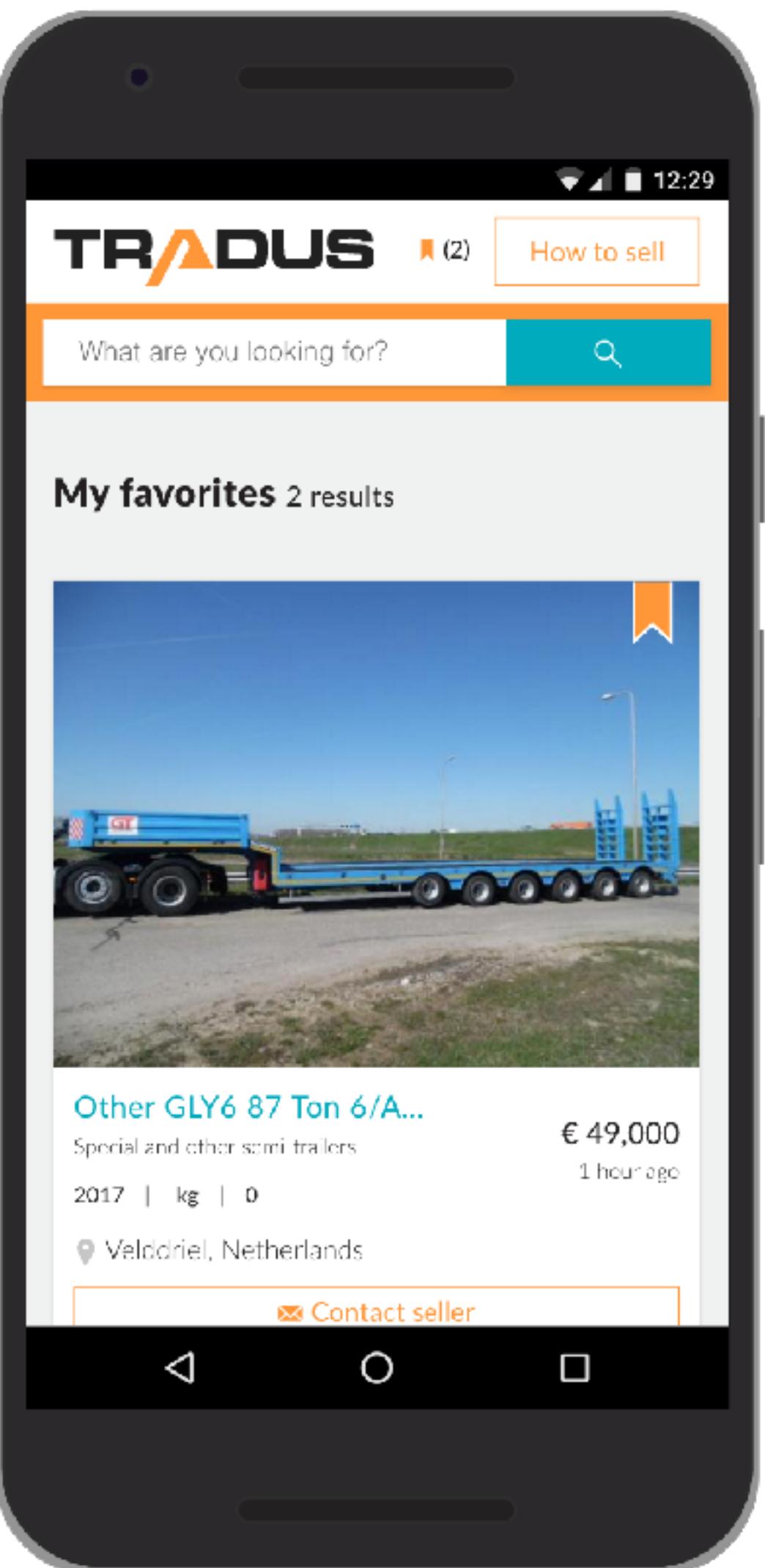
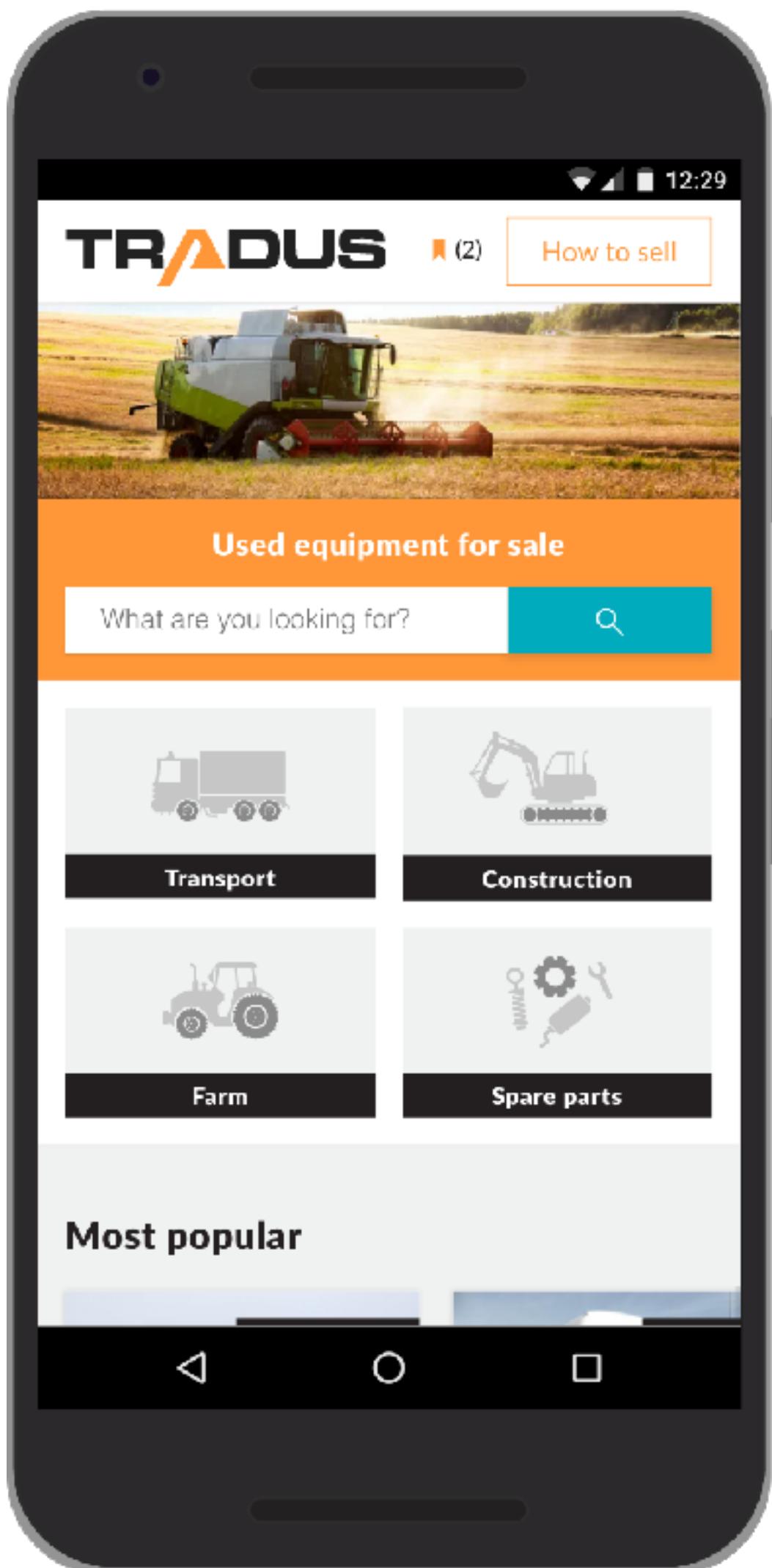


Work

At De Voorhoede we built web sites and apps for innovative start-ups, small businesses and large enterprises. These are the stories behind our projects.

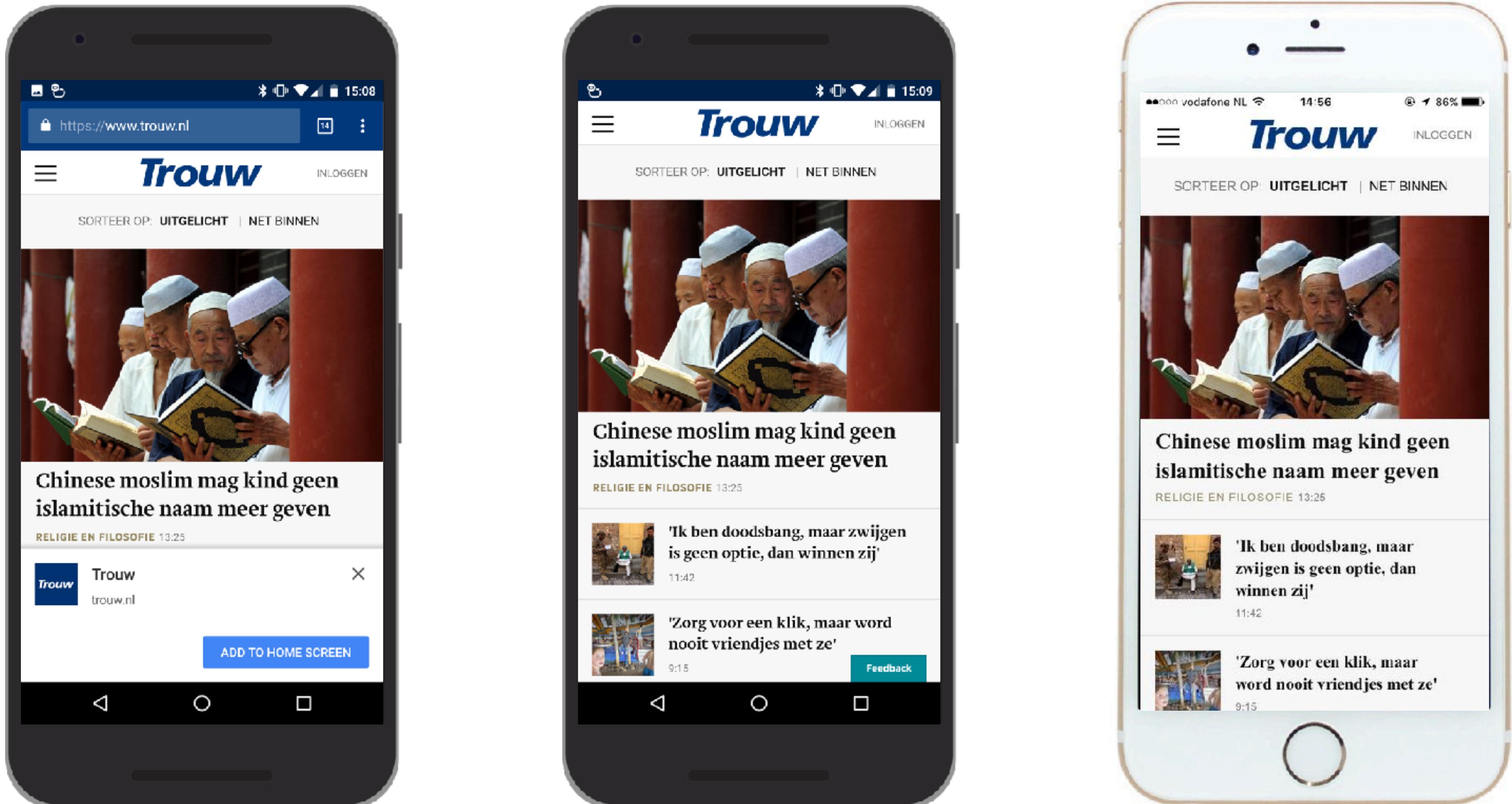


EXAMPLE: TRADUS.COM



tradus.com

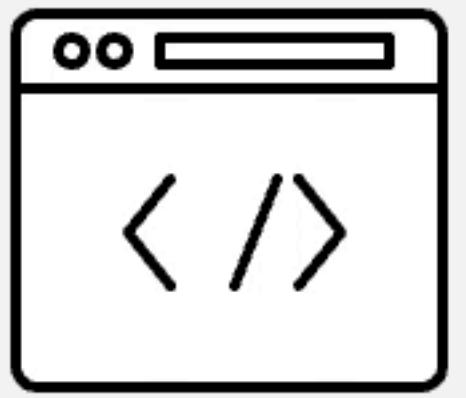
TROUW.NL



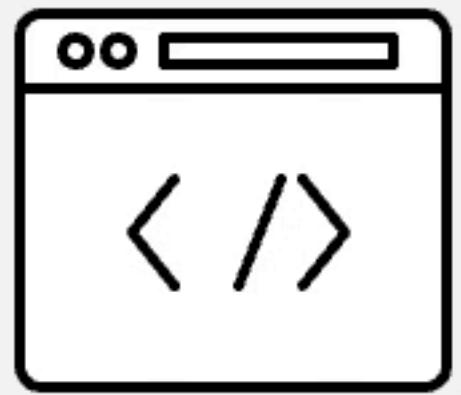
left-to-right: web, Android app, iOS app

[read voorhoede.nl/nl/portfolio/trouw-volkskrant-parool/](http://read.voorhoede.nl/nl/portfolio/trouw-volkskrant-parool/)

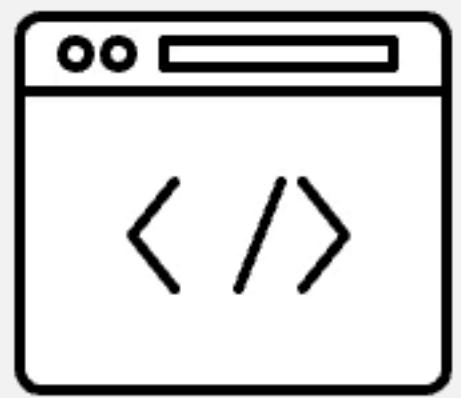
SERVICE WORKER AS PROXY



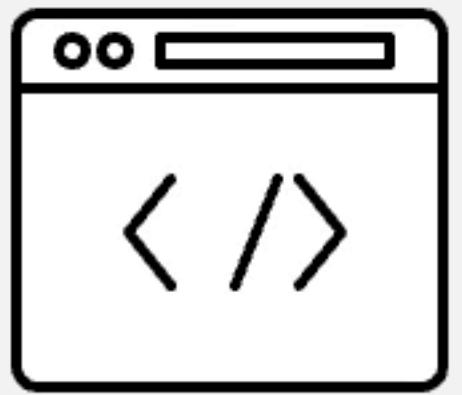
HIJACK FETCH



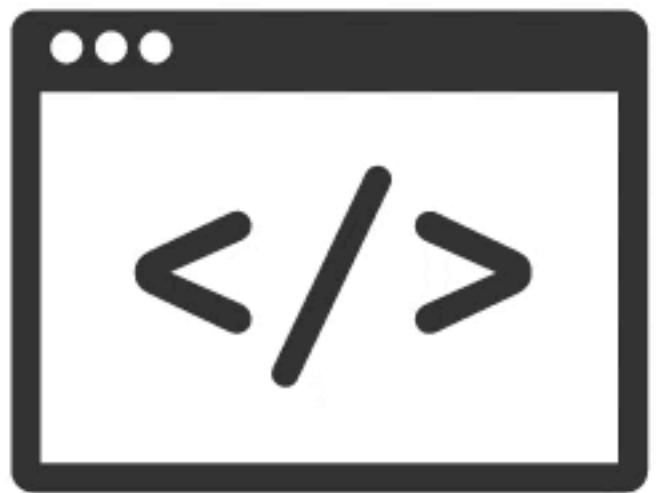
CACHE ON FETCH



SERVE FROM CACHE



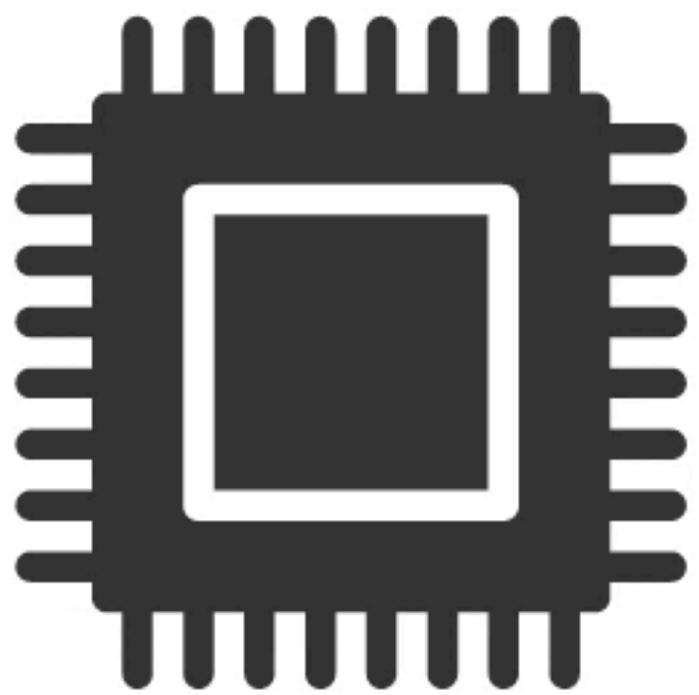
STALE WHILE REVALIDATE



Page



Cache

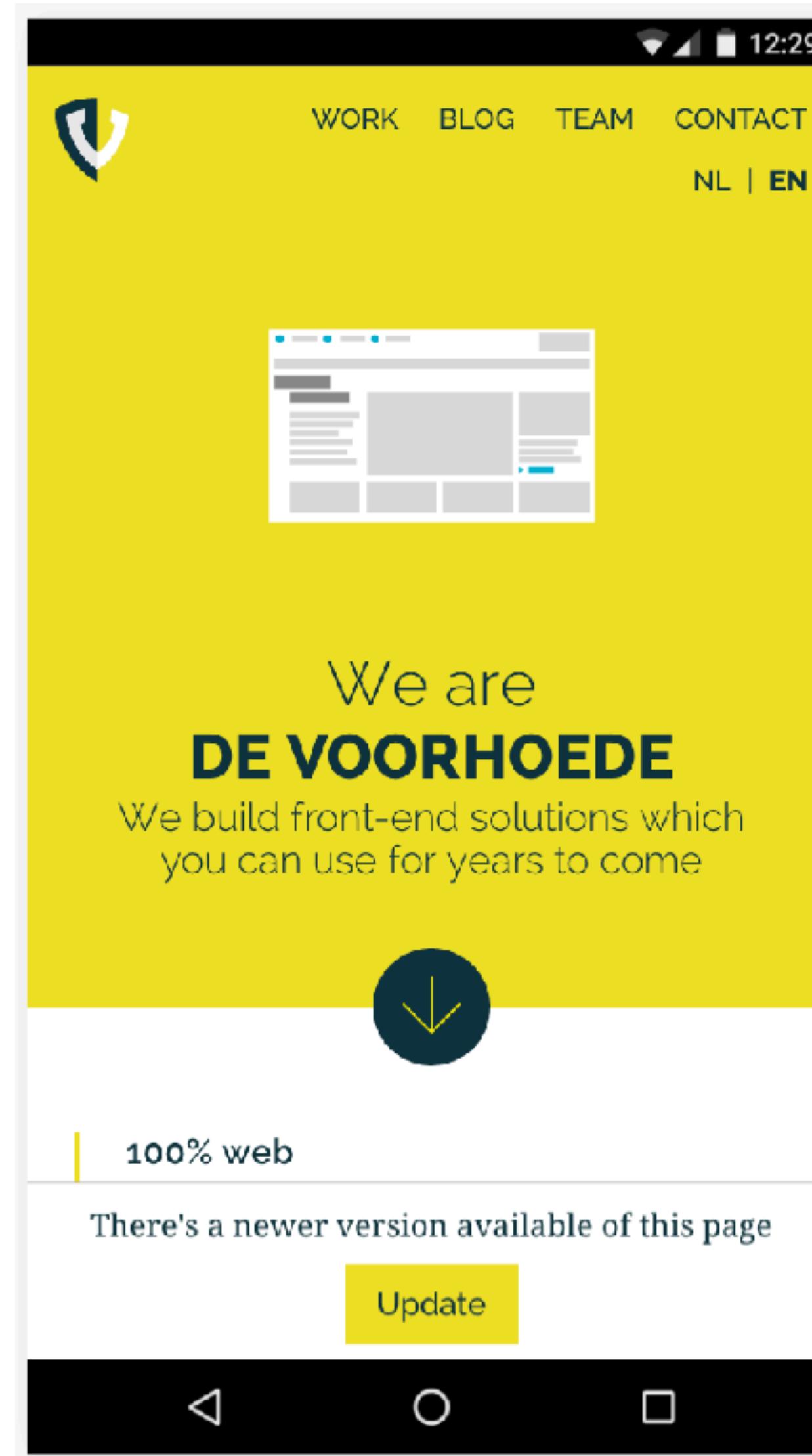


ServiceWorker



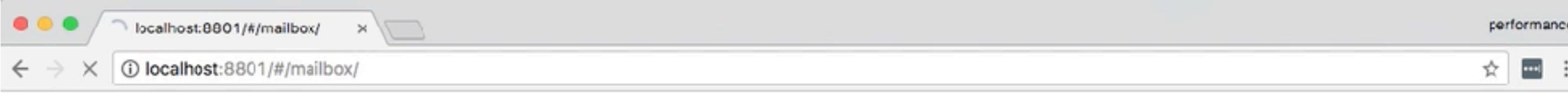
Network

STALE WHILE REVALIDATE

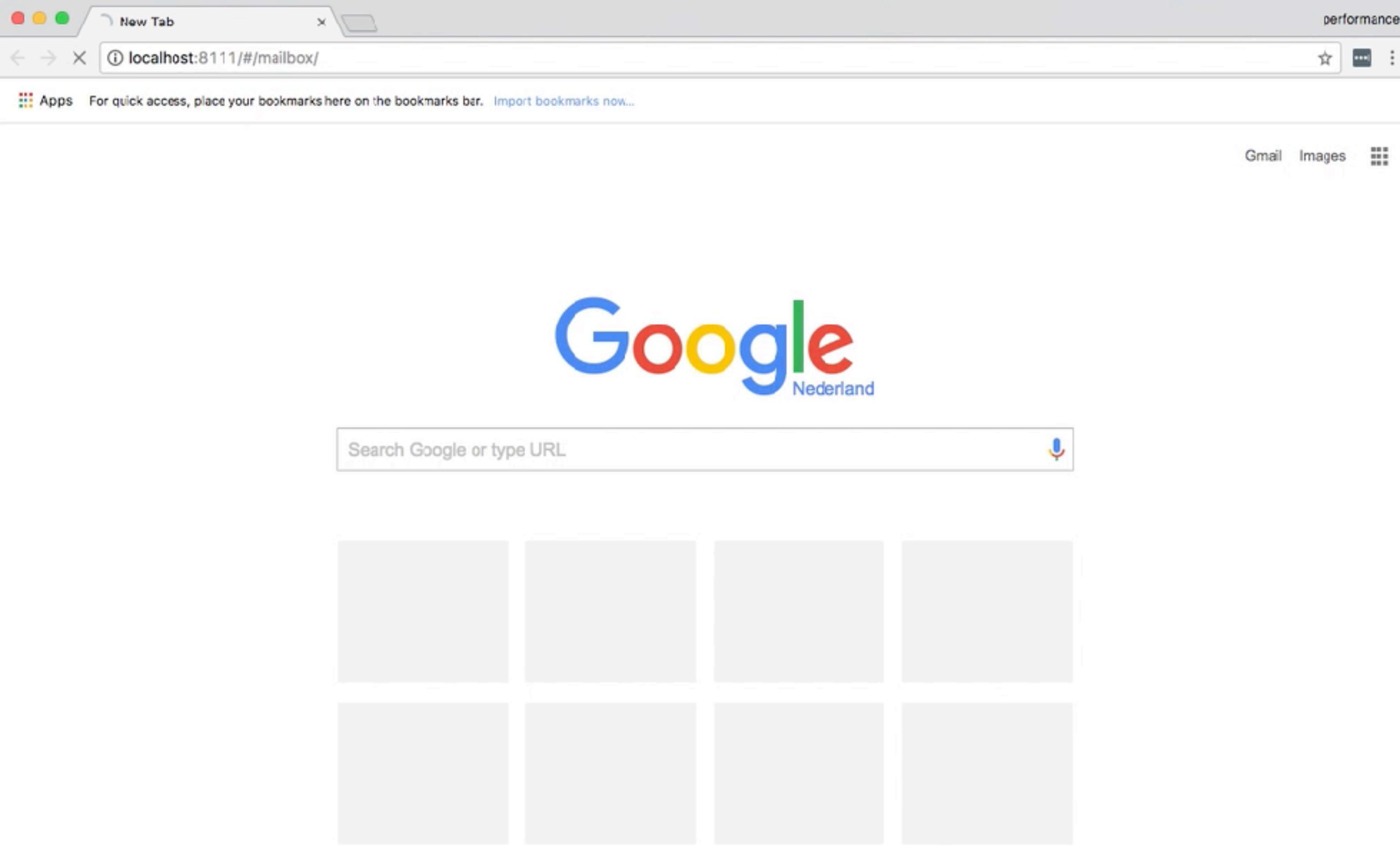


LET'S DO THIS!

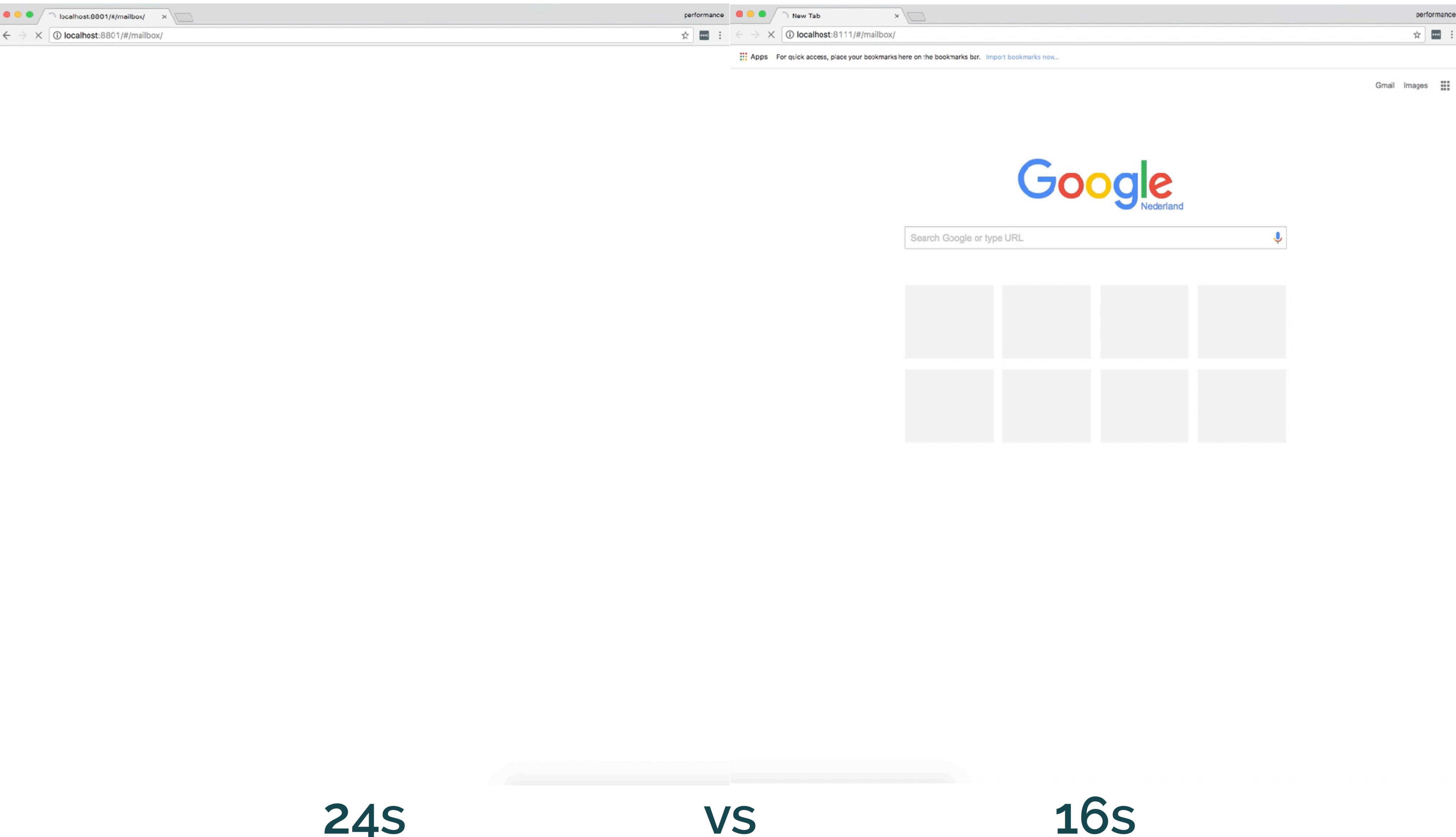




NEW STARTMAIL ON HIGHSPEED INTERNET



NEW STARTMAIL WITH SERVICE WORKER



24s

vs

16s

CACHED RESPONSES!!

The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, there's a sidebar with a list of messages from 'Tijn Schmits' and 'Aeryn Tonita'. The main area displays a table of cached requests under the 'Request' column and their status under the 'Response' column.

#	Request	Response
0	https://api.nylas.com/messages/6dapowlkmm39e2o63unwj1pe9	OK
1	https://api.nylas.com/messages/av3tjfm8xvmzzpaio9r2u0zsu	OK
2	https://api.nylas.com/messages?in=6tv1f6yil0pziavaddek7rsvs&limit=51&offset=0	OK
3	https://api.nylas.com/messages?in=ezt8jxw8j0hjdse3hxd75x2vu&limit=51&offs...	OK

The sidebar also lists other storage types: Local Storage, Session Storage, IndexedDB, Web SQL, and Cookies. Under Cache, 'Cache Storage' is expanded, showing 'workbox-precach' and 'messages - http://'. 'messages - http://' is highlighted with a blue selection bar. Other sections include Application Cache and Frames, with 'top' listed under Frames.

OFFLINE === UNAUTHENTICATED

The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, there's a sidebar with a list of messages from "StartMail WebMail". The main area displays network requests and storage information.

Application

Manifest

Service Workers

Clear storage

Storage

#	Request	Response
0	https://api.nylas.com/messages/6dapowlkmm39e2o63unwj1pe9	OK
1	https://api.nylas.com/messages/av3tjfm8xvmzzpaio9r2u0zsu	OK
2	https://api.nylas.com/messages?in=6tv1f6yil0pziavaddek7rsvs&limit=51&offset=0	OK
3	https://api.nylas.com/messages?in=ezt8jxw8j0hjdse3hxd75x2vu&limit=51&offs...	OK

Cookies

- <http://localhost:8111/#/mailbox/ezt8jxw8j0hjdse3hxd75x2vu>

Cache

Cache Storage

- [workbox-precache](#)
- [messages - http://localhost:8111/#/mailbox/ezt8jxw8j0hjdse3hxd75x2vu](#)
- [Application Cache](#)

Frames

- [top](#)

CACHED RESPONSES?!

The screenshot shows the Chrome DevTools interface with the Application tab selected. In the Cache section, there are three entries:

#	Request	Response
0	https://api.nylas.com/messages/6dapowlkmm39e2o63unwj1pe9	OK
1	https://api.nylas.com/messages/av3tjfm8xvmzzpaio9r2u0zsu	OK
2	https://api.nylas.com/messages?in=6tv1f6yil0pziavaddek7rsvs&limit=51&offs...	OK

In the Console tab, a promise is being resolved:

```
caches.match('https://api.nylas.com/messages/av3tjfm8xvmzzpaio9r2u0zsu')
  .then(response => response.json())
  .then(responseBody => console.log(responseBody))
```

The promise status is pending, and the response body is displayed:

```
VM1818:3
{account_id: "dw8ulo7jflxelumie5wt8dalt", bcc: Array(0), body: "<p>Hi,</p><p>Good luck with the demo again today!</p><p>Kind regards,<br>Tijn</p>", cc: Array(0), date: 1502970512, ...}
```

CACHED RESPONSES?!??!

The screenshot shows a browser developer tools window with the "Application" tab selected. The "Cache" section is open, displaying a list of items. One item is expanded, showing a complex object structure representing an email message. The object includes properties like account_id, bcc, body, cc, date, events, files, folder, from, id, object, reply_to, snippet, starred, subject, thread_id, to, unread, and __proto__. The body property contains HTML content: "Hi, Good luck with the demo again today! Kind regards, Tijn". The object is identified by the identifier VM1818:3.

```
.then(responseBody => console.log(responseBody))
< ► Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}

  VM1818:3
  ▼ {account_id: "dw8ulo7jflxelumie5wt8dalt", bcc: Array(0), body: "<p>Hi,</p><p>Good luck with the demo again today!
  </p><p>Kind regards,<br>Tijn</p>", cc: Array(0), date: 1502970512, ...} ⓘ
    account_id: "dw8ulo7jflxelumie5wt8dalt"
    ▶ bcc: []
      body: "<p>Hi,</p><p>Good luck with the demo again today!</p><p>Kind regards,<br>Tijn</p>"
    ▶ cc: []
      date: 1502970512
    ▶ events: []
    ▶ files: []
    ▶ folder: {display_name: "INBOX", id: "ezt8jxw8j0hjdse3hxd75x2vu", name: "inbox"}
    ▶ from: [...]
      id: "av3tjfm8xvmzzpaio9r2u0zsu"
      object: "message"
    ▶ reply_to: []
      snippet: "Hi, Good luck with the demo again today! Kind regards, Tijn"
      starred: false
      subject: "Demo today"
      thread_id: "2h9a05axn6zpc84cx9byu2jjk"
    ▶ to: [...]
      unread: false
    ▶ __proto__: Object
```

I'm caching **sensitive data!**

ALL EXAMPLES ARE PUBLIC



Delete onbeforeunload?

Authentication on
cache storage?

Let's try encryption?

ENCRYPTION



ON FETCH

```
self.addEventListener('fetch', event => {
  const request = event.request;
  if (isApiGetRequest(request)) {
    event.respondWith(tryCacheThenNetwork(request, 'messages-cache'));
  } else {
    event.respondWith(fetch(event.request));
  }
});
```

TRY CACHE THEN NETWORK

```
function tryCacheThenNetwork(request, cacheName) {
  return caches.open(cacheName)
    .then(cache => cache.match(request.url))
    .then(response => {
      if (response) {
        return decryptResponse(response);
      } else {
        return fetchAndCache(request, cacheName);
      }
    });
}
```

FETCH AND CACHE

```
async function fetchAndCache(request, cacheName) {
  const response = await fetch(request.url, {
    headers: new Headers(),
    credentials: 'include'
  });

  const encryptedResponse = await encryptResponse(response.clone());

  return caches.open(cacheName)
    .then(cache => cache.put(request, encryptedResponse))
    .then(() => response)
}
```

CHANGE THE RESPONSE

```
function encryptResponse(response) {
  return response.text()
    .then(text => encrypt(text, 'supersecretpassword'))
    .then(encryptedBody => {
      const options = {
        status: response.status,
        statusText: response.statusText,
        headers: {'content-type': 'application/json'}
      };
      return new Response(JSON.stringify(encryptedBody), options);
    });
}
```

ENCRYPTING A STRING WITH FORGE

```
function encrypt(message, password) {
  const salt = forge.random.getBytesSync(128);
  const key = forge.pkcs5.pbkdf2(password, salt, 4, 16);
  const iv = forge.random.getBytesSync(16);

  const cipher = forge.cipher.createCipher('AES-CBC', key);
  cipher.start({iv: iv});
  cipher.update(forge.util.createBuffer(message));
  cipher.finish();

  const cipherText = forge.util.encode64(cipher.output.getBytes());

  return {
    cipher_text: cipherText,
    salt: forge.util.encode64(salt),
    iv: forge.util.encode64(iv)
  };
}
```

In cryptography, a **salt** is random data that is used as an additional input to a one-way function that "hashes" a password or passphrase.

ENCRYPTING A STRING WITH FORGE

```
function encrypt(message, password) {
  const salt = forge.random.getBytesSync(128);
  const key = forge.pkcs5.pbkdf2(password, salt, 4, 16);
  const iv = forge.random.getBytesSync(16);

  const cipher = forge.cipher.createCipher('AES-CBC', key);
  cipher.start({iv: iv});
  cipher.update(forge.util.createBuffer(message));
  cipher.finish();

  const cipherText = forge.util.encode64(cipher.output.getBytes());

  return {
    cipher_text: cipherText,
    salt: forge.util.encode64(salt),
    iv: forge.util.encode64(iv)
  };
}
```

An **Initialization Vector** is an unpredictable random number used to “initialize” an encryption function, resulting in different ciphertexts everytime you encrypt the same message

ENCRYPTING A STRING WITH FORGE

```
function encrypt(message, password) {
  const salt = forge.random.getBytesSync(128);
  const key = forge.pkcs5.pbkdf2(password, salt, 4, 16);
  const iv = forge.random.getBytesSync(16);

  const cipher = forge.cipher.createCipher('AES-CBC', key);
  cipher.start({iv: iv});
  cipher.update(forge.util.createBuffer(message));
  cipher.finish();

  const cipherText = forge.util.encode64(cipher.output.getBytes());

  return {
    cipher_text: cipherText,
    salt: forge.util.encode64(salt),
    iv: forge.util.encode64(iv)
  };
}
```

We can encrypt a string

THE MESSAGE AS OBJECT

```
{  
  account_id: "dw8ulo7jflxelumie5wt8dalt",  
  bcc: [],  
  body: "<p>Hi,</p>\n<p>Good luck with the demo again today!</p>\n<p>Kind regards,<br>Tijn</p>",  
  cc: [],  
  date: 1502970512,  
  events: [],  
  files: [],  
  folder: {  
    display_name: "INBOX",  
    id: "ezt8jxw8j0hjdse3hxd75x2vu",  
    name: "inbox"  
  },  
  from: [  
    {  
      email: "tijn@startmail.com",  
      name: "Tijn Schmits"  
    }  
  ],  
  id: "av3tjfm8xvmzzpaio9r2u0zzu",  
  object: "message",  
  reply_to: [],  
  snippet: "Hi, Good luck with the demo again today! Kind regards, Tijn",  
  starred: false,  
  subject: "Demo today",  
  thread_id: "2kbj6ob1pke0g8f04j62z7e9g",  
  to: [  
    {  
      email: "blubbleepblub1@startmail.com",  
      name: ""  
    }  
  ],  
  unread: false  
}
```

THE MESSAGE AS STRING

```
' {"account_id":"dw8ulo7jflxelumie5wt8dalt","bcc":[],"body":"<p>Hi,</p>\n<p>Good luck with the demo again today!</p>\n<p>Kind regards,<br>Tijn</p>","cc":[],"date":1502970512,"events":[],"files":[],"folder":  
{"display_name":"INBOX","id":"ezt8jxw8j0hjdse3hxd75x2vu","name":"inbox"}, "from": [ {"email":"tijn@startmail.com","name":"Tijn Schmits"} ], "id": "av3tjfm8xvmzzpaio9r2u0zzu", "object": "message", "reply_to": [], "snippet": "Hi, Good luck with the demo again today! Kind regards, Tijn", "starred": false, "subject": "Demo today", "thread_id": "2kbj6ob1pke0g8f04j62z7e9g", "to": [{"email": "blubbleepblub1@startmail.com", "name": ""}], "unread": false} '
```

ENCRYPTED MESSAGE

```
{  
    cipher_text: '2W9UmQxx2FUHG..',  
    iv: 'vomfUHS2AZrBr6XyTmM2mw..',  
    salt: 'YDX/0pAtvbXpe3WAVALX..'  
}
```

DECRYPTION



SERVE FROM CACHE

```
function tryCacheThenNetwork(request, cacheName) {
  return caches.open(cacheName)
    .then(cache => cache.match(request.url))
    .then(response => {
      if (response) {
        return decryptResponse(response);
      } else {
        return fetchAndCache(request, cacheName);
      }
    });
}
```

DECRYPTING

```
function decryptResponse(response) {
  return response.json()
    .then(responseBody => decrypt(responseBody, 'supersecretpassword'))
    .then(decryptedBody => {
      const options = {
        status: response.status,
        statusText: response.statusText,
        headers: {'content-type': 'application/json'}
      };
      return new Response(JSON.stringify(decryptedBody), options);
    });
}
```

DECRYPTING

```
function decrypt({ cipherText, salt, iv }, password) {
  const key = forge.pkcs5.pbkdf2(password, forge.util.decode64(salt), 4, 16);

  const decipher = forge.cipher.createDecipher('AES-CBC', key);
  decipher.start({iv: forge.util.decode64(iv)});
  decipher.update(forge.util.createBuffer(forge.util.decode64(cipherText)));
  decipher.finish();

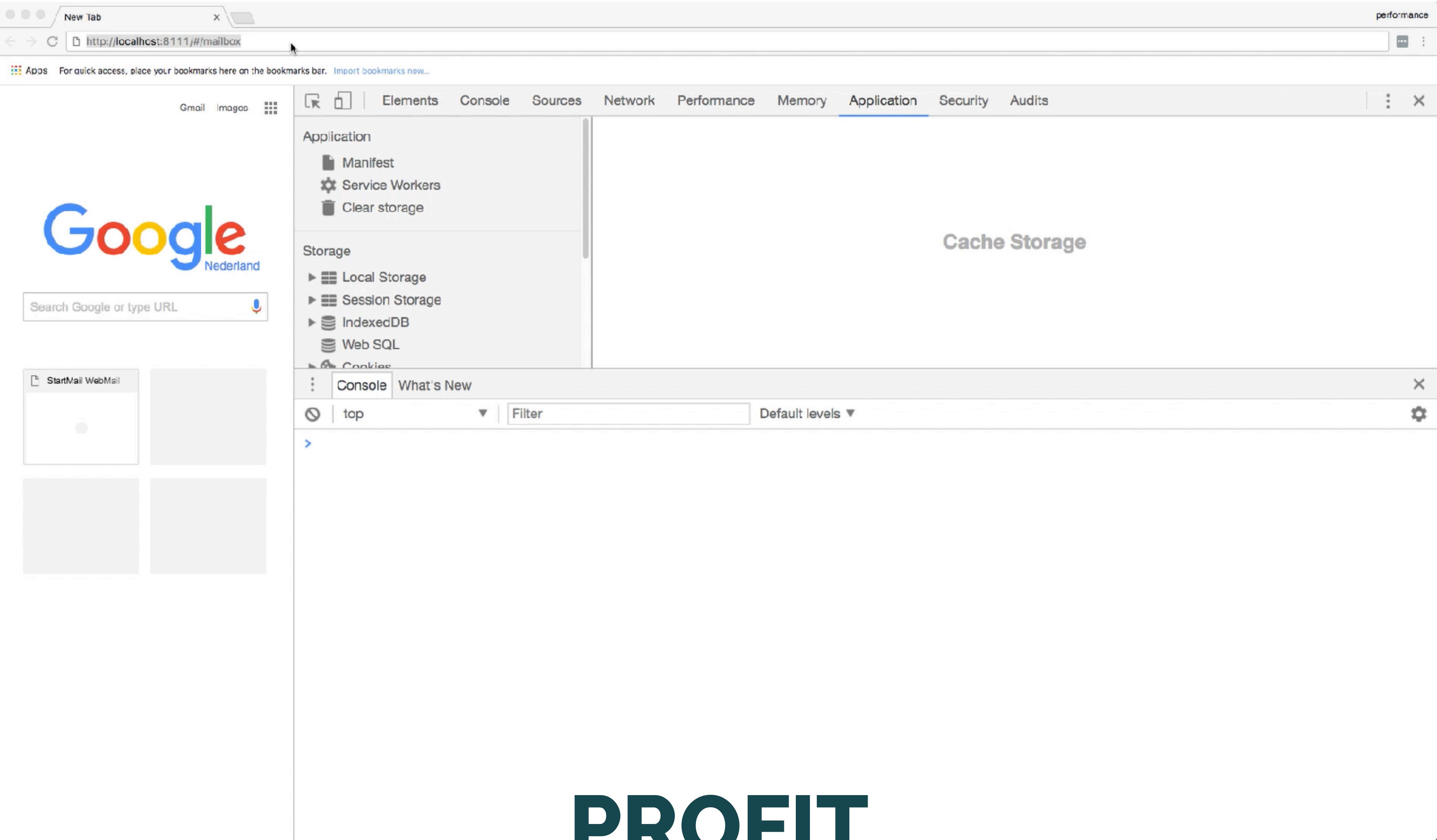
  return decipher.output.toString();
}
```

DECRYPTING

```
function decryptResponse(response) {
  return response.json()
    .then(responseBody => decrypt(responseBody, 'supersecretpassword'))
    .then(decryptedBody => {
      const options = {
        status: response.status,
        statusText: response.statusText,
        headers: {'content-type': 'application/json'}
      };
      return new Response(decryptedBody, options);
    });
}
```

PUTTING IT TOGETHER





One more thing...

Where to store the
supersecretpassword?

Making an authenticated request
results in a response with the **crypto
key as response header**

AS A RESPONSE HEADER WHEN ENCRYPTING

```
async function fetchAndCache(request, cacheName) {
  const response = await fetch(request.url, {
    headers: new Headers(),
    credentials: 'include'
  });

  const encryptionKey = response.headers.get('crypto-key');
  const encryptedResponse = await encryptResponse(response.clone(), encryptionKey);

  return caches.open(cacheName)
    .then(cache => cache.put(request, encryptedResponse.clone()))
    .then(() => response)
}
```

FOR DECRYPTING

From the page when logging in:

```
fetch('/api/auth', {  
  method: 'POST',  
  headers: {  
    'content-type': 'application/json'  
  },  
  body: { username, password }  
})
```

Resulting in response with crypto key for decrypting

FOR DECRYPTING

From the page when checking authentication status:

```
fetch('/api/auth', {  
  method: 'GET',  
  credentials: 'include'  
})
```

Resulting in response with crypto key for decrypting

AS A REQUEST HEADER WHEN DECRYPTING

From the page:

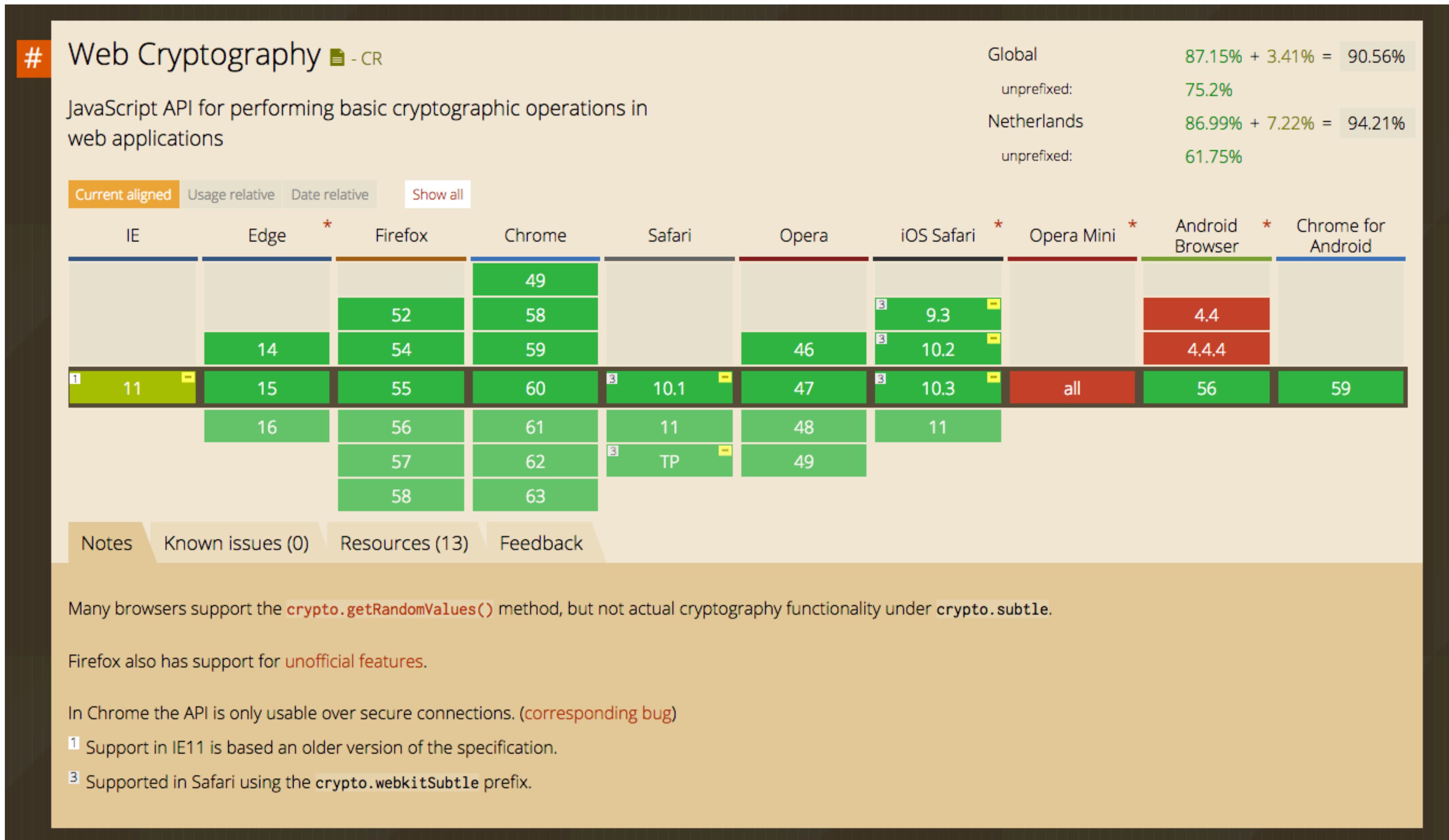
```
fetch('/api/messages/12345', {  
  method: 'GET',  
  headers: {  
    'content-type': 'application/json',  
    'crypto-key': cryptoKeyFromAuthResponse  
  },  
  credentials: 'include'  
})
```

AS A REQUEST HEADER WHEN DECRYPTING

```
function tryCacheThenNetwork(request, cacheName) {
  return caches.open(cacheName)
    .then(cache => cache.match(request.url))
    .then(response => {
      if (response) {
        return decryptResponse(response, request.headers.get('crypto-key'));
      } else {
        // we could even strip out the crypto-key header,
        // so it will never hit the network
        return fetchAndCache(request, cacheName);
      }
    });
}
```

We give up offline, but have
control over the network

WEB CRYPTOGRAPHY API



[Can I use Web Cryptography](#)

Thanks!

Startmail is hiring!

What do you think?



DE VOORHOEDE

front-end developers