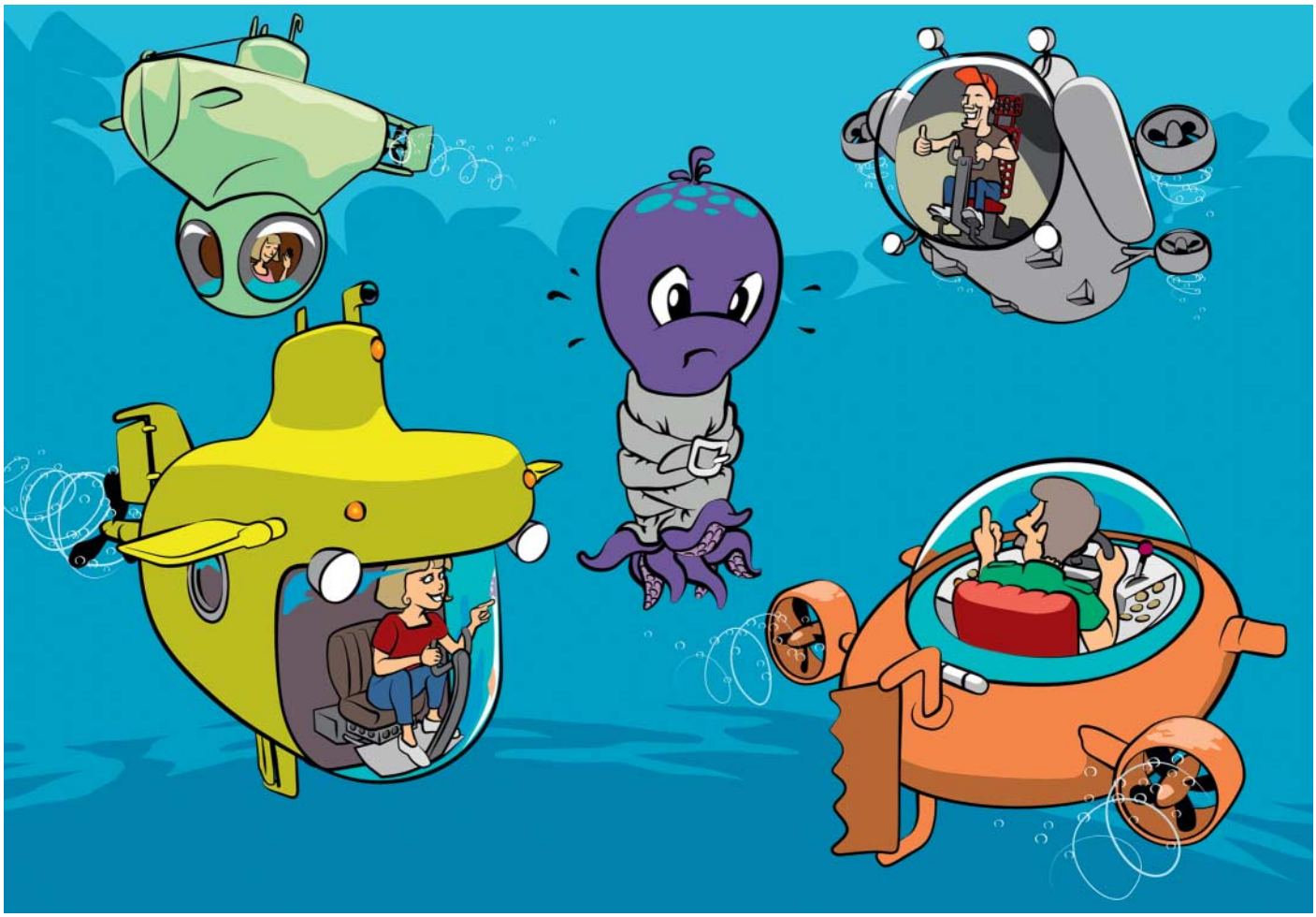


Bild: Rudolf A. Blaha

Gut eingebettet

Social-Media-Inhalte datenschutzfreundlich in die Website einbinden

Videos und Tweets bereichern Websites, aber externe Inhalte verraten Nutzerdaten an ihre Anbieter – normalerweise. Das c't-Projekt Embetty agiert als Vermittler zwischen externen Angeboten und der eigenen Website und ermöglicht, Social-Media-Content datenschutzfreundlich einzubinden.

Von Herbert Braun

Das Web ist nicht auf Grundlage des Datenschutzes gebaut. Ein häufig übersehenes Problem ist, dass der Aufruf einer einzigen Webseite in der Regel nicht nur von einem Webserver beantwortet wird: Oft verbindet sich ein Browser mit Dutzenden verschiedenen Hosts, ohne dass der Nutzer viel davon mitbekommt.

Manches lässt sich nun mal nicht so einfach auf den eigenen Server holen: Um Share- und Like-Buttons, Tweets, YouTube-Videos oder Facebook-Inhalte datenschutzfreundlich darzustellen, ist beträchtlicher Aufwand vonnöten, während es mit einem Daten sammelnden JavaScript vom Hersteller kinderleicht geht.

Für Share-Buttons bietet die c't bereits seit 2014 eine datenschutzfreundliche Lösung an: Der „Shariff“ fordert Anbieter-Code erst an, wenn der Anwender Buttons tatsächlich benutzt (siehe ct.de/y9rj). Mit „Embetty“ stellen wir in diesem Artikel eine Lösung für eingebettete Social-Media-Inhalte vor.

Embetty kann derzeit mit folgenden Inhaltstypen umgehen: YouTube-Videos, Vimeo-Videos, Facebook-Videos und Tweets. Die Videos selbst kommen zwar vom jeweiligen Webdienst – eine Umleitung der Streams über den Server wäre technisch und rechtlich sehr problematisch –, nicht aber das Vorschaubild. Der Browser spricht den externen Dienst also erst an, wenn der Nutzer das Video angeklickt hat. Dieser Artikel beschreibt die Grundzüge der Installation von Embetty.

Wie Shariff hat auch Embetty eine Server-Komponente – ansonsten ließen sich beispielsweise Tweets nur als „Click to Play“ darstellen, was kaum zumutbar wäre. Stattdessen fordert Embetty die In-

halte über einen Proxy an, der vom Betreiber der einbettenden Website unterhalten wird. Mit diesem Server kommuniziert ein JavaScript im Browser des Besuchers. Für die beiden Komponenten gibt es zwei GitHub-Repositorys: https://github.com/heiseonline/embetty für das Client-Skript, https://github.com/heiseonline/embetty-server für den Server.

Anders als bei Shariff ist das Backend nicht in PHP geschrieben, sondern fußt auf Node.js, genauer: auf dem wohl verbreitetsten Node.js-Webserver Express. Das natürliche Habitat dieses Werkzeugs ist ein Linux-Webserver, doch bekommt man es auch auf einem Windows-Arbeitsrechner zum Laufen.

Für die Installation gibt es verschiedene Wege. Wer bereits Docker nutzt [1], kann sofort loslegen – ein Image steht im Docker-Hub bereit.

```
docker run --name embetty --rm
 -p 8080:8080 -e
 VALID_ORIGINS=http://localhost
 heiseonline/embetty-server:latest
```

Dieser Befehl lädt das Docker-Image herunter und startet den Container unter dem Namen „embetty“; eventuelle frühere Instanzen dieses Containers löscht die Anweisung. Nun lauscht das Image auf Port 8080. Die Umgebungsvariable `VALID_ORIGINS` setzt den Localhost mit dem Standardport als gültige Quelle von Embetty-Anfragen.

Ob das funktioniert hat, können Sie im Browser ausprobieren: Wenn Sie http://localhost:8080 aufrufen, sollte Embetty mit einem kleinlauten „Not Found“ antworten. Etwas informativer ist http://localhost:8080/version, das die Versionsnummer ausgibt.

Außer per Docker lässt sich Embetty auch mit dem klassischen `git clone` und `npm install` herunterladen und installieren:

```
git clone https://github.com/↵
 ↳heiseonline/embetty-server.git
cd embetty-server
npm i
PORT=8080
VALID_ORIGINS=http://localhost
npm start
```

Dieser Aufruf funktioniert unter Windows nur, wenn Sie ihn in einer Cygwin-Konsole starten. In der normalen Windows-Shell lautet die Syntax geringfügig anders:

```
set PORT=8080 &&
VALID_ORIGINS=http://localhost &&
npm start
```

Bleibt noch ein dritter Weg, mit dem sich Embetty unter Linux ausprobieren lässt: Der von Facebook entwickelte Paketmanager Yarn nutzt die gleichen Pakete wie npm, soll aber schneller und sicherer als dieser arbeiten. Im Test startete der Server nach der Yarn-Installation nicht unter Windows.

```
yarn global
add @heise/embetty-server PORT=8080
VALID_ORIGINS=http://localhost
embetty start
```

Die ersten zwei Zeilen laden Embetty herunter, installieren es und setzen die Umgebungsvariable für den Port (voreingestellt ist im Container Port 3000), die dritte die Umgebungsvariable für gültige Zugriffsquellen, die vierte ruft das soeben installierte Konsolenwerkzeug `embetty` auf. Falls das System dieses nicht findet, müssen Sie die Pfad-Variable aktualisieren. Unter Linux geht das zum Beispiel so:

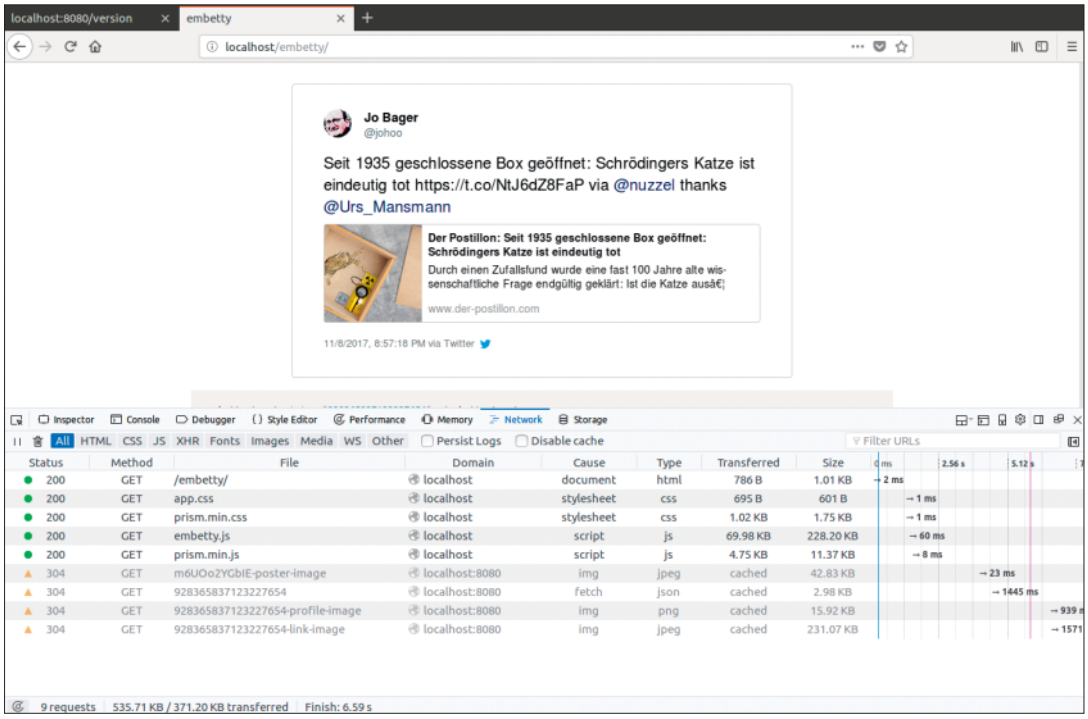
```
PATH="$PATH:`yarn global bin`"
```

Wie bei den beiden Varianten zuvor sollten Sie nun einen Server haben, der auf Port 8080 lauscht.

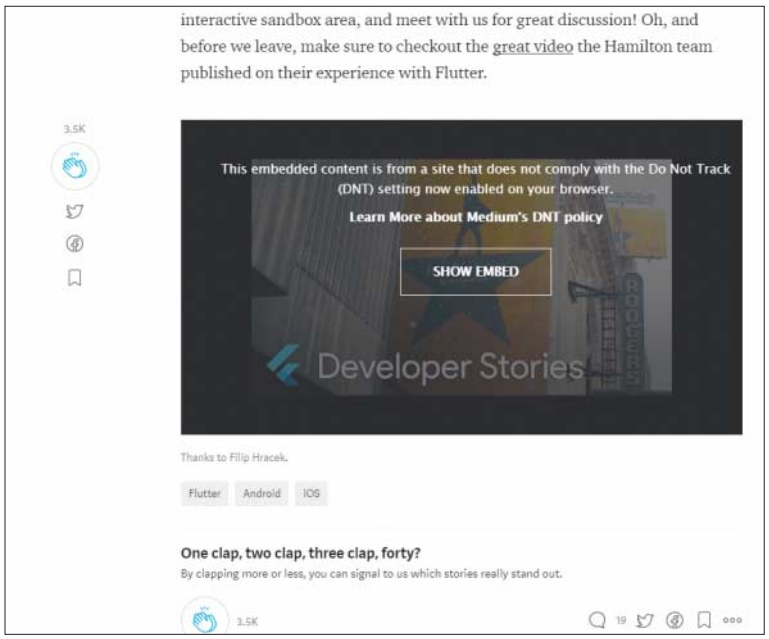
Routen

Um Anfragen zu verarbeiten, greift der Server auf das Verzeichnis routes zurück. Express wandelt ja nicht wie etwa Apache URLs in lokale Dateipfade um, sondern braucht für jede Anfrage eine mit einer Aktion verknüpften Route.

So sorgt die Datei routes/index.js dafür, dass nur gültige Anfragen von legitimen Quellen durchkommen, und reicht Aufrufe an andere Skripte im Verzeichnis weiter. Für eine URL, die mit video/youtube/ beginnt, übernimmt erst video.js und dann video/youtube.js. Dort beant-



Hat man eine eigene Twitter-App angelegt, stellt Embetty Tweets dar, ohne dass der Browser des Nutzers mit den Twitter-Servern kommunizieren muss.



Medium.com kennt das Datenschutzproblem mit externen Inhalten auch und löst es auf seine Weise.

wortet eine einfache Funktion die Anfragen:

```
router.get('/:id-poster-image', ...)
```

`:id` ist ein Platzhalter für eine YouTube-Video-ID – http://localhost:8080/video/youtube/m6UOo2YGbIE-poster-image ist also für Embetty eine gültige URL, die es mit einem Video-Vorschaubild beantwortet.

Der eigentliche Kern des Servers steckt in node_modules/@heise/embetty-base/lib, wo embetty.js die erste Anlaufstelle ist. Hier sind die Methoden hinterlegt, die in routes aufgerufen werden, um Inhalte von den Social-Media-Diensten abzurufen, aufzubereiten und zu cachen.

Eingebettet

Höchste Zeit, Embetty auszuprobieren. Dafür brauchen Sie einen weiteren Webserver, der die Seiten mit den eingebetteten Inhalten ausliefert. Zu diesem Zweck eignet sich zum Beispiel eine XAMPP-Umgebung. Dort hinterlegen Sie eine entsprechend mit dem Embetty-Skript präparierte Webseite. Letzteres sowie eine Vorlage für die Seite erhalten Sie im Embetty-Repository:

```
git clone https://github.com/↵
↳heiseonline/embetty.git
cd embetty
npm i
npm run build
```

Falls Ihnen Yarn mehr zusagt als npm, finden Sie eine Anleitung auf der Projektseite. In jedem Fall sollten Sie nach dem Kompilieren im Verzeichnis „dist“ das Client-Skript embetty.js verwenden können. In dieses Verzeichnis kopieren Sie am besten auch gleich die in „example“ liegenden Dateien – eine Gruppe von HTML-Beispieldateien.

Öffnen Sie nun eine der HTML-Dateien, zum Beispiel index.html. Aufs Wesentliche reduziert, sieht der Code etwa so aus:

```
<html>
 <head>
  <title>Embetty</title>
  <meta data-embetty-server=↵
   ↳"http://localhost:8080">
  <script async src="embetty.js">
  </script>
 </head>
 <body>
  <embetty-tweet
  status="928365837123227654">
  </embetty-tweet>
  <embetty-video type="youtube"
  video-id="m6UOo2YGbIE">
  </embetty-video>
  </body>
</html>
```

Embetty-Inhalte betten Sie in Form der selbstdefinierten HTML-Elemente `<embetty-tweet>` und `<embetty-video>` ein. Das Video-Element kennt die Typen `youtube`, `vimeo` und `facebook`. Die `video-id` beziehungsweise beim Tweet das `status`-Attribut benennt die ID des jeweiligen Inhalts. Solche Custom-Elemente sind zwar schon seit Jahren als Webstandard-Entwurf im Gespräch [2], haben sich aber noch nicht überall durchgesetzt – bis heute unterstützen nur Chromium-Browser und Safari diese Technik von Haus aus. Daher enthält embetty.js die JavaScript-Bibliothek webcomponents.js, mit der auch andere Browser klaglos solche Elemente verarbeiten können.

Ansonsten benötigt die Seite nur noch einen Hinweis auf die Adresse des zuständigen Embetty-Servers. Diese legen Sie in einem `<meta>`-Tag im Attribut `data-embetty-server` fest. In den Beispieldateien ist bereits der Localhost eingetragen, aber mit dem voreingestellten Port 3000.

Werfen Sie nun den lokalen Webserver sowie den Embetty-Server mit den oben genannten Parametern an und rufen Sie diese Seite auf. Das Ergebnis ist ein halber Erfolg. Das Video erscheint in Form eines Vorschaubildes und lässt sich ganz normal durch Anklicken abspielen. Bis dahin nimmt der Browser keinerlei Kontakt mit dem YouTube-Server auf. Das Skript fordert das Bild von Embetty an, wie ein Blick in die Browser-Entwicklerwerkzeuge beweist.

Beim Tweet ist allerdings Fehlanzeige; die Browser-Konsole beklagt sich über gebrochene Versprechen und unautorisierte Zugriffe. Um Tweets von außerhalb abzurufen, braucht man nämlich eine registrierte Twitter-Anwendung.

Diese ist jedoch schnell erstellt. Rufen Sie apps.twitter.com auf, loggen Sie sich mit einem Twitter-Account ein und klicken Sie auf „Create New App“. Lassen Sie sich einen Namen und eine Beschreibung einfallen und tragen Sie Ihre Website-URL ein (nicht unbedingt die, in der Sie die App benutzen wollen). Nach dem Speichern stellen Sie im Reiter „Permissions“ „Read only“ ein.

Im Reiter „Keys and Access Tokens“ hat Twitter für Sie API-Schlüssel und -Geheimnis hinterlegt. Über eine Schaltfläche auf dieser Seite müssen Sie nun noch ein Zugangs-Token und das dazugehörige Geheimnis erzeugen.

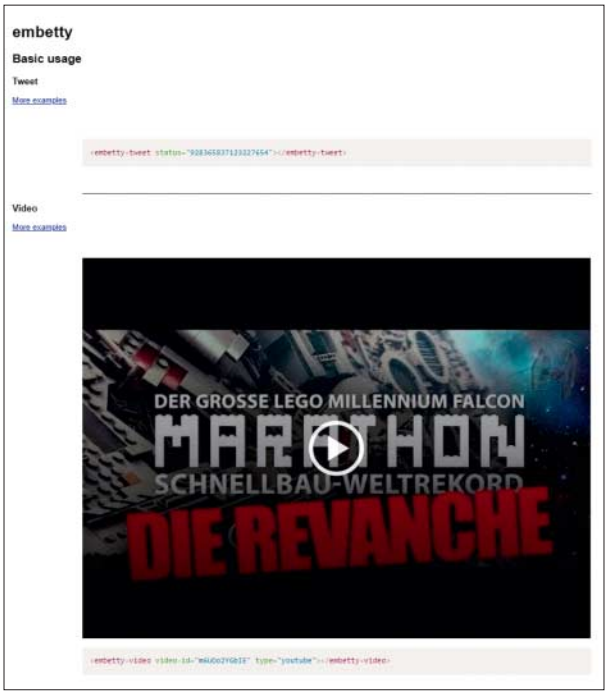
Diese vier Schlüssel teilen Sie dem Embetty-Server in Form von Umgebungsvariablen mit:

```
PORT=8080
VALID_ORIGINS=http://localhost
TWITTER_ACCESS_TOKEN_KEY=...
```

```
TWITTER_ACCESS_TOKEN_SECRET=...
TWITTER_CONSUMER_KEY=...
TWITTER_CONSUMER_SECRET=...
npm start
```

Falls Sie die Docker-Variante benutzen, kennzeichnen Sie wie im Beispiel oben jede dieser Umgebungsvariablen mit dem Parameter `-e`. Nach dem Neustart des Servers mit diesen Daten sollte der Aufruf einer URL wie http://localhost:8080/tweet/928365837123227654 den Tweet abrufen, allerdings in Form roher JSON-Daten. Unter Windows ohne Cygwin klappte das im Test jedoch nicht – Embetty antwortete mit einem Fehler 400 („Bad Request“).

Für die Darstellung dieses Tweets ist das Embetty-Client-Skript zuständig: Ein neuer Aufruf der vorhin präparierten Testseite rendert den Tweet ähnlich, wie man es von twitter.com gewohnt ist, inklusive eines darin enthaltenen Bildes oder einer URL-Vorschau.



Halber Erfolg: Embetty zeigt das Vorschaubild des Videos, aber der Tweet erfordert noch einen weiteren Arbeitsschritt.

Embetty-Praxis

Für den Einsatz im Produktivsystem werden Sie wahrscheinlich noch an ein paar Schräubchen drehen müssen. So sollte der Embetty-Server auch als Service laufen, damit nicht das versehentliche Schließen der Konsole oder ein Strg+C lauter hässliche Leerstellen auf Ihrer Website hinterlässt. Ein Prozess-Manager kann den Embetty-Server als Dienst starten. Die bekanntesten Lösungen dafür sind Forever, PM2 und StrongLoop.

Zum Cachen benutzt Embetty per Default node-lru-cache, eine Node.js-Implementierung des LRU-Algorithmus, der den „least recently used“ Eintrag aus dem Cache löscht, wenn der Cache voll ist. Mit der Umgebungsvariable `EMBETTY_CACHE` können Sie das Caching-Verfahren konfigurieren, zum Beispiel auf `lru://max:100`, um Platz für 100 Elemente zu reservieren. Für größere Installationen empfiehlt es sich, Redis einzusetzen, eine beliebte und flexible In-Memory-Datenbank. Embetty ist dafür vorbereitet. Sie müssen Embetty über eine Umgebungsvariable die Adresse des Servers mitteilen: `redis://...`

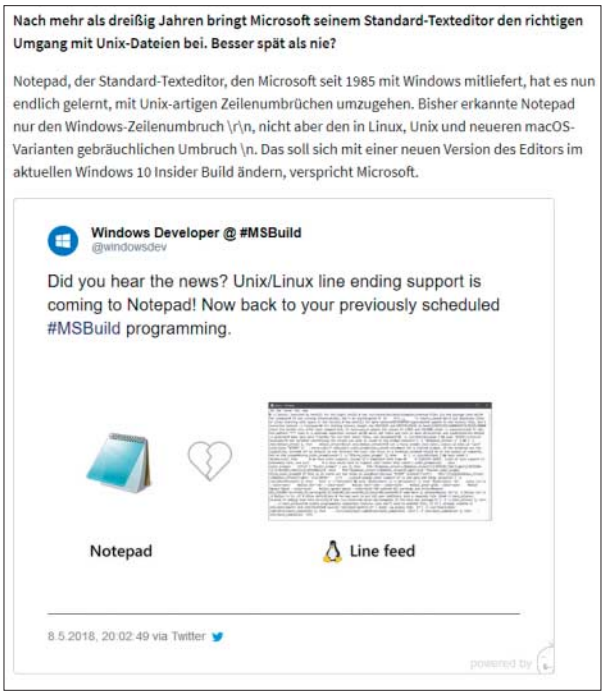
Schließlich würde man in einem professionellen Umfeld Embetty hinter einem Reverse Proxy verstecken. Eine beliebte Lösung dafür ist Nginx: Diese Kombination aus Webserver, Load Balancer und Reverse Proxy nimmt Anfragen aus dem Web entgegen. Richtet sich eine Anfrage beispielsweise an ihre.domain/embetty/, leitet Nginx diese an den Embetty-Server weiter, der nach außen hin nicht sichtbar ist.

Embetty ist eine flexible Lösung, mit der Webmaster die Privatsphäre ihrer Besucher besser schützen können, ohne auf Social-Media-Content zu verzichten. Der Rahmen, den Embetty bereitstellt, ließe sich auch für andere externe Inhalte nutzen. So könnte Embetty mit entsprechenden Erweiterungen auch den c’t-Shariff ersetzen oder Instagram-Storys, Pinterest-Pins oder Facebook-Posts datenschutzfreundlich in die eigene Website holen. Entwickler, die sich an diesem Open-Source-Projekt beteiligen möchten, sind willkommen! *(jo@ct.de)* c't

Literatur

[1] Peter Siering, Docker-Einstieg, Antworten auf die häufigsten Fragen, c’t 10/2018, S. 160
[2] Herbert Braun, HTML maßgeschneidert, Eigene Elemente und Templates definieren, c’t 26/2013, S. 182

Download und weitere Infos: ct.de/y9rj



Auf heise online ist Embetty seit Kurzem im Einsatz.