Success Story Architektur:

Performance-Optimierung bei Alice

Projektbeschreibung und Kontext

Performance stellte bislang bei marketinggetriebenen Websites lediglich eine Anforderung dar, die die Nutzerzufriedenheit steigert, aber nicht kritisch ist, wie es z.B. bei Websites für zeitnahe Aktienkäufe oder Aktionsangebote.

> Performance matters! Performance ist eine Triple-Win-Story: Ein besseres Kundenerlebnis, höhere Konversions-Raten und niedrigere Infrastruktur-Kosten. Bei einem Produkt, das sich maßgeblich durch Geschwindigkeit auszeichnet, ist dies der erste Eindruck, den der Kunde von unserem Unternehmen erhält.
>
> Markus Leptien,
> Telefónica Germany,
> VP Online & VAS Development

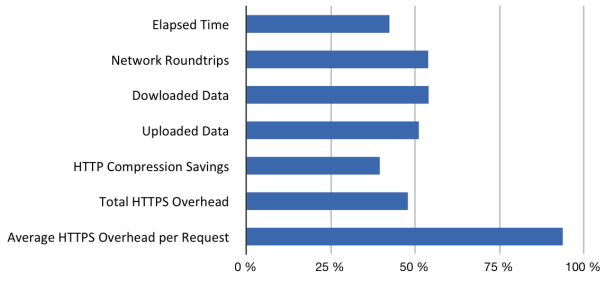
Neuere Studien belegen nun, dass die Geschwindigkeit einer Website entscheidend für den Umsatz mitverantwortlich ist. Bei Amazon wurde im Rahmen vergleichender Tests (sogenannter A/B-Tests) die Geschwindigkeit der Site kontrolliert verringert. Dabei ergab sich, dass pro 0,1 Sekunden Verzögerung ca. ein Prozent Umsatz weniger generiert wurde. Zu ähnlichen Ergebnissen kamen Google, Yahoo und Bing, die vergleichbare Studien betrieben hatten[1].

Hansenet bzw. Telefónica Germany betreibt unter der Marke Alice zwei große Portale: ein Neukunden- und ein Bestandskundenportal. Über ersteres werden Bestellungen von Neukunden abgewickelt, zweiteres bietet Self-Service und Bestellmöglichkeiten für Bestandkunden.

Der Erstkontakt des Kunden findet dabei meist über das Neukundenportal statt. Im stark umkämpften Telko-Markt, in dem Mitbewerber ähnliche Produkte zu vergleichbaren Preisen anbieten, kann eine optimale User Experience entscheidend für die Anbieterauswahl sein.

Ziel des Projekts war es, die Performance des Neukundenportals signifikant zu steigern und die dabei gewonnen Erkenntnisse auch direkt in das Bestandskundenportal einfließen zu lassen. Weiterhin sollte der Kundenlogin, der den Einstieg im Bestandskundenportal darstellt, beschleunigt werden.

Der Erfolg des Projekts hat alle Erwartungen übertroffen. Der durchschnittliche Ladevorgang bei gleichem Inhalt wurde auf ca. 40 % der ursprünglichen Zeit verringert, die Netzwerkzugriffe und die bewegten Daten um ca. die Hälfte minimiert[2].



2

Elapsed Time	6.739 seconds	2.860 seconds
Network Round Trips	89	48
Downloaded Data	584651 bytes	294356 bytes
Uploaded Data	41213 bytes	21547 bytes
HTTP Compression Savings	100455 bytes	396079 bytes
Total HTTPS overhead	13675 bytes	6552 bytes
Avg. HTTPS overhead/request	153 bytes	136 bytes

Seitenladezeit -57%
Downloaded Bytes -50%
Uploaded Bytes -48%

1 http://velocityconf.com/velocity2009/public/schedule/detail/8523

Fachliche Benefits

Performance-Optimierung erhöht die Nutzerzufriedenheit. Auf das Laden einer Seite zu warten, ist enervierend für Nutzer, und je mehr Seiten – z.B. in einem komplexen Bestellprozess – aufgerufen werden, desto frustrierender kann das Nutzungserlebnis werden.

Die gefühlte Leistung lässt sich dabei nicht immer objektiv messen. Wichtig ist, dem Nutzer möglichst rasch ein Feedback auf seine Aktion (z.B. dem Klick auf einen Link oder Button) zu bieten. Eine Website hat keinen Fortschrittsbalken, der den Aufbau der Seite darstellt. Während sich eine Website aufbaut, benötigt der Nutzer ein visuelles Feedback, damit er sichergehen kann, dass der Aufruf funktioniert. Die Seite selbst fungiert als Fortschrittsbalken: Statt dass der Nutzer vor einem weißen Bildschirm sitzt, der sich nach der Wartezeit in einem einzigen Augenblick aufbaut, sollte sich die Seite im Browser nach und nach aufbauen. Durch das Feedback erscheint für den Nutzer die Ladezeit subjektiv kürzer.

Elemente, die nicht sichtbar sind, sollten erst dann geladen werden, wenn die Seite bereits optisch aufgebaut ist. Die Zeit, die der Nutzer benötigt, um sich auf der vollständig dargestellten Seite zu orientieren, kann genutzt werden, um unsichtbare Inhalte, die für die Funktionalität der Seite notwendig sind, zu laden, ohne dass der Nutzer dies bemerkt.

Die messbare Performance ist ebenso wichtig. Bei den meisten dynamisch generierten Websites benötigt das Rendering des HTMLs auf dem Server nur einen Bruchteil der Zeit (in unserem Fall nur wenige Millisekunden), die der Benutzer mit dem Warten auf die komplette Site verbringt. Der Rest der Zeit wird damit verbracht, Assets wie Stylesheets, Grafiken und JavaScripte über das Netzwerk zu transportieren. Es gilt, eine reduzierte Anzahl davon in möglichst wenigen Requests hin zum Nutzer zu übertragen, um diese messbare Performance ebenfalls zu optimieren.

Technische Lösungen

In den folgenden Abschnitten werden die Tasks vorgestellt, die wir im Wesentlichen implementierten, um die Performance zu steigern.

Rendering so früh wie möglich erlauben

Wir ermöglichten es dem Web-Browser technisch, heruntergeladene Inhalte so früh wie möglich darzustellen. Dazu lieferten wir ihm möglichst früh alle nötigen Informationen zum Rendering der Seite.

Dimensionen von Grafiken und Bildern festlegen

Wenn man für Bilder die Höhe und Breite schon im Image-Tag vorgibt, kann der Browser mit der Berechnung der Seite schon beginnen, bevor die eigentlichen Daten heruntergeladen wurden. Dies wurde erreicht, indem wir im CMS diese Informationen als Pflichtfelder aufgenommen haben.

CSS in den Seitenkopf einbauen

Wir binden Stylesheets bereits im Dokument-Kopf ein und ermöglichen so dem Browser, möglichst früh mit dem Rendering zu beginnen.

CSS vor JavaScript einbinden

Browser blockieren JavaScripte, bis alle CSS-Dateien geladen sind, was daran liegt, dass erstere Informationen aus Style-Attributen verwenden könnten. Darum muss der Browser vor der Ausführung von JavaScript warten, bis die CSS-Dateien geladen sind. Also referenzierten wir, um eine Blockierung

zu vermeiden, CSS-Dateien auch im HTML-Code vor JavaScript-Dateien.

Reduziere die Anzahl der HTTP-Requests

Je weniger einzelne Ressourcen ein Browser laden muss, desto schneller können diese heruntergeladen werden, da eine unnötige Verzögerung durch HTTP bzw. durch TCP-Round-Trips entfällt. Daher fassten wir viele kleine Ressourcen zu wenigen großen zusammen.

Parallelisiere Downloads über verschiedene Hosts

Ein Browser kann pro Host nur eine bestimmte Anzahl an Verbindungen öffnen. Darum verteilten wir die Ressourcen einer Site auf mehrere Hosts, um die Anzahl an Verbindungen zu vergrößern. Für statische Inhalte legten wir zwei (virtuelle) Hosts an, über die der entsprechende Content ausgeliefert wird.

> Für eine dauerhafte, nachhaltige Performance-Optimierung müssen die Maßnahmen automatisiert bzw. in den Build-Prozess integriert werden
>
> Markus Leptien,
> Telefónica Germany,
> VP Online & VAS Development

Reduziere DNS-Lookups

Das Einbinden von Ressourcen sehr vieler Hosts, wie sie bei Zählpixeln üblich sind, versuchten wir zu vermeiden,was sich jedoch nicht in jedem Fall erreichen ließ. Daher luden wir Zählpixel asynchron nach, nachdem die Seite inklusiver aller benötigten Scripte bereits geladen und vom Benutzer verwendbar war. So erscheint die Seite subjektiv schneller, da sie schon benutzbar ist, während Zählpixel noch nachgeladen werden. Die Anzahl der DNS-Lookups reduziert sich dadurch nicht messbar, subjektiv erscheint die Site trotzdem schneller.

Reduziere SSL-Handshakes

Wir prüften, ob bei Seiten, die über HTTPS ausgeliefert werden, es sich überhaupt lohnt, verschiedene parallele Hosts zu verwenden.

Verwende ein CDN

Content Delivery Networks bieten ein Netzwerk aus Servern, die an verschiedenen Standpunkten gehostet werden. Nutzern, die eine Ressource aus dem CDN anfordern, wird diese automatisch über einen räumlich nahe liegenden Server ausgeliefert. Über ein solches Netzwerk verteilten wir Ressourcen wie Videos oder große Downloads.

Verwende einen Cookie-freien Host für statischen Content

Cookies werden bei jedem Request der Cookie-Domain vom Browser zum Server übertragen. Das Übertragen dieser Cookies ist für viele Inhalte unnützer Overhead, zumal die Bandbreite vom Browser zum Server oft recht klein ist. Content, der statisch ist, also keine Session-Informationen o.ä. aus Cookies benötigt, lieferten wir darum von einer zweiten, Cookie-freien Domain aus. .

Optimiere Stylesheets

Die Anzahl der HTTP-Requests reduzierten wir, indem wir CSS-Dateien zu wenigen oder zu einer einzigen Ressource zusammenfassten. Überflüssige White-Spaces ließen sich in den Stylesheet-Dateien über Kompressoren entfernen, Hintergrund-Grafiken fassten wir zu CSS-Sprites zusammen.

Optimiere JavaScript

JavaScripts, die auf verschiedenen Seiten benötigt werden, lieferten wir nicht inline im HTML-Dokument sondern als eigene Ressource aus. So wurde die Größe des einzelnen HTML-Dokuments reduziert. JavaScripts, die nur auf einer Seite verwendet werden, ließen wir eingebettet im HTML-Dokument stehen, um die Anzahl der Requests zu reduzieren. Das Zusammenfassen von JavaScript-Bibliotheken zu einer einzigen Ressource erfolgte sowohl im Build-Prozess, die Kompression zur Laufzeit für Inline-JavaScripts. JavaScripts, die nicht schon vor dem Rendern der Seite benötigt werden, packten wir in den Fuß des Dokuments. So werden sichtbare Elemente vor den JavaScript-Ressourcen geladen und die Seite wird bereits dargestellt, während JavaScript-Bibliotheken, die im Fuß der Seite eingebunden werden, noch nachladen. Elemente, die schwergewichtig sind, weil sie z.B. DNS-Lookups benötigen, werden erst nach dem eigentlichen Laden der Seite dynamisch nachgeladen.

Verwende den Expires-Header

Beim ersten Besuch einer Seite muss der Browser alle Inhalte herunterladen. Diese Inhalte lassen sich im Browser-Cache zwischenspeichern. Wenn nun allerdings statischer Inhalt, der im Cache des Browsers liegen soll, auf dem Server geändert wird, so muss technisch dafür gesorgt werden, dass der veränderte Content eine neue URL bekommt. Wir implementier-

ten im CMS einen Mechanismus, der für jede Version eines Dokuments eine neue URL erzeugt.

Komprimiere Ressourcen

Textuelle Ressourcen (HTML, JavaScript und CSS) übertragen wir komprimiert.

Ermögliche Proxy-Caching

Wir motivierten das Caching durch Proxies durch HTTP-Header.

Minimiere Backend-Calls

Anfragen vom Frontend an das Backend wurden durch die Einführung von Caches und High- Availability-Proxies minimiert. Anschließend wurden große, atomare, langläufige Backend-Calls durch mehrere kleinere, aber schnellere Backend-Calls ersetzt. So wurde auch die Login-Zeit verkürzt.

Automatisiere die Optimierung

Die oben beschriebenen, umgesetzen Performance-Tasks wurden in einem Build-Prozess automatisiert. Nur so lassen sich einmal erreichte Performance-Steigerungen auch über einen langen Zeitraum erhalten.

Methodisches Vorgehen

Ein interdisziplinäres Team aus Testern, Administratoren, dem Betrieb, Architekten und Entwickeln wurde als Task-Force gebildet, das auf einem Testsystem systematisch Performance-Tests unter realistischen Bedingungen durchführte. Dabei wurden Schwachpunkte und Engpässe ermittelt und diskutiert sowie gemeinschaftlich Engineering-Tasks erarbeitet, um diese zu beseitigen. Sobald eine ausreichende Menge von Tasks implementiert war, wurden die Auswirkungen erneut gemessen. Aus diesen Messungen wurden weitere Aufgaben abgeleitet, die wiederum im Backlog gesammelt und dann von den Teammitgliedern abgearbeitet wurden.
Dieser iterative, positive Regelkreis ermöglichte eine sukzessive Steigerung der Performance zeitgleich zur eigentlichen Entwicklung der Site.

Unsere Aufgaben und Rollen

Ein Mitarbeiter von Holisticon war als Mitglied des interdisziplinäres Teams wesentlich an der iterativen Prozessplanung, an der Analyse und der Implementierung beteiligt.

Telefónica Germany GmbH & Co. OHG
Online & VAS Development
Überseering 33a
22297 Hamburg

Holisticon AG
Griegstraße 75 / Haus 25
22763 Hamburg
Telefon: (040) 5074 2722
Telefax: (040) 5074 2724
E-Mail: info@holisticon.de