UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Server-side implementations of the PJAXR framework for smart Ajax applications

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Jonas Braun

Erstgutachter:	Prof. Dr. Steffen Staab Institute for Web Science and Technologies
Zweitgutachter:	René Pickhardt Institute for Web Science and Technologies

Koblenz, im Februar 2015

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz verfügbar.	☐	☐
Der Quellcode ist unter einer Creative Commons Lizenz verfügbar.	☐	☐
Die erhobenen Daten sind unter einer Creative Commons Lizenz verfügbar.	☐	☐

. . . .

(Ort, Datum) (Unterschrift)

Contents

1 Introduction

At the beginning of the World Wide Web websites were self-contained. The content which was initially loaded was not changed until a new URL was requested by the user. One big change was the invention of AJAX. It introduced the possibility to change content without the need of requesting a new URL. This approach only loading one website initially and then changing its content interactively is called single-page application. Single-page applications are more user-friendly than the common designs, e.g. due to lower load times in combination with not being able to indicate clearly that a new website is being loaded. A big disadvantage is that users are not able to save their websites as a bookmark, because while surfing on this page, the URL never changes. AJAX is available in nearly every browser which is a reason that a lot of different frameworks gain this functionality, improving and enhancing it.

Single-page applications have a problem in the current time: Search engines and other crawlers trying to examine websites will find nothing more than content which was provided initially. Every further change of content is not easily accessible. As Google is the most used search engine in the World Wide Web they have a design pattern[1] for implementing a crawlable AJAX web application. In this guideline it is recommended to have snapshots available under non-user-friendly URLs, called Hash-Bang URLs.

In this thesis we will improve and evaluate a new technology called PJAXR. Together with Stephan Groß we developed this technique for using AJAX with it's advantages but trying to avoid the disadvantages explained before. The result of cooperation is the frontend-side implementation called *jquery-pjaxr*. As PJAXR also needs a backend to be implemented, I developed the first PJAXR backend called *django-pjaxr*. Additionally in this thesis I will introduce a new PJAXR backend for PHP, *php-pjaxr*.

1.1 Background and motivation

explain

1.2 Goals of this thesis

explain

1.3 Thesis outline

explain

[1]https://developers.google.com/webmasters/ajax-crawling/docs/getting-started

2 Fundamentals

2.1 HTTP request

The world wide web has one main protocol to let web-browsers and web-servers communicate, the Hypertext Transfer Protocol (HTTP), which is built on top of the Transmission Control Protocol (TCP). TCP and so HTTP requests always start with a handshake to establish a connection before data is transferred. After this handshake, the client sends the request data to the web-server. This recognizes and interprets the request and if the requested resource is available, sends the according data back, otherwise it sends an error. Typically it renders data out of a database into a HTML-Template or to JSON and sends it back as the response. Afterwards the browser receives, interprets and displays the response data, which is most of the time HTML, CSS, images or scripts.

2.2 HTML

explain

2.3 Single-page application

explain

2.4 AJAX

explain

2.5 History API

explain

3 State of the art

AJAX is a widely used technique in the internet to build web applications because of the user experience improvements it brings. In [1] is mentioned that AJAX applications have a better usability than non-AJAX websites. The same conclusion is made in [2], despite of the lack of browser navigation support. Beside the navigation problem another disadvantage is, as presented in [3], crawling AJAX applications is not trivial. One solution of this task is finding clickables and navigating to every page found by this. Nevertheless [3] states also that this only generates a snapshot of the full application. Even search engines are avoiding to crawl websites because of it's difficulty.[4], page 81 Currently the task of building a crawlable single-page application using AJAX is often avoided, instead crawling algorithms are getting improved and in focus of research.

A common way to implement AJAX in websites is to use the pattern of HIJAX. It encourages developers to have AJAX in mind from the start of building the website. But when they start to implement the single-page application they should do not implement AJAX. This should only be done after the website is finished without AJAX. This could then be done by *hijacking* an event like a click to then handle it by an additional script.

3.1 HIJAX

explain

3.2 Hash-Bang URLs

explain

4 PJAXR

4.1 Introduction

explain

4.1.1 Requirements

The idea of PJAXR is to have the advantages of AJAX while trying to avoid it's disadvantages. This means it should have the same UX improvements, and reduced load times as classical AJAX. On the other hand single-page applications using PJAXR should be easily crawlable by the most used crawlers without additional efforts. E.g. it should not be necessary to have several endpoints with the same content for crawlers and normal users. Additionally PJAXR should be a generic solution for single-page applications, not just for one specific.

4.2 Concept

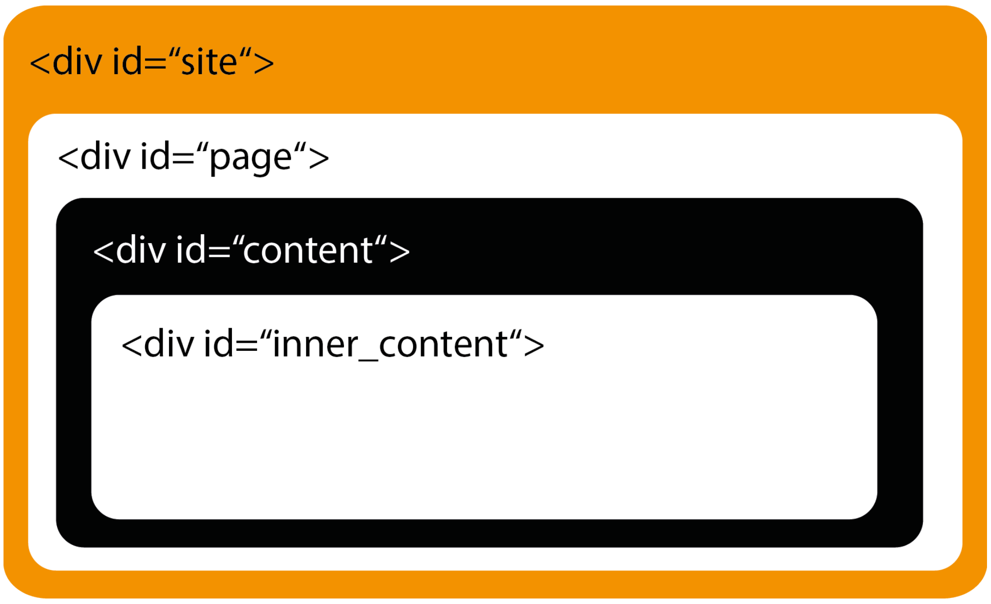

Figure 1: Basic HTML structure of a web application using PJAXR

Every single-page application which uses PJAXR has a hierachical namespace structure. Typically the namespace consists of a prefix, a site id, a page id, a content id and an inner-content id. Every level in the hierarchy has it's counterpart on the website as shown in figure 1, a container with an id telling which part of the namespace it belongs to. After the PJAXRfrontend is initialized it hijacks events like clicks on links and enriches the request with the current website's namespace. A PJAXR backend analyzes the namespace of the requested website and the one sent in the request. For every hierarchy level it checks if both namespaces match. If on one level the namespaces don't match the containers according to this level will be responded to the request. An optimized web application only grabs the data necessary for those containers and render them afterwards. The PJAXR frontend retrieves the response, replaces the containers at the website and updates the current namespace.

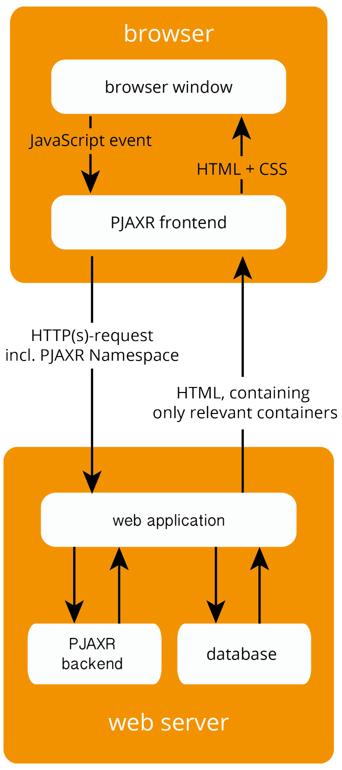

Figure 2: Component and communication diagram of PJAXR

4.3 Realization

To load a page, the first request is a normal HTTP request followed by a JavaScript script initializing the PJAXR frontend module. Further requests to the same host are then initated by this module. The HTTP headers of these requests are extended by the namespace of the current website. The webserver using a PJAXR backend detects the first namespace of the requested resource not matching the namespace of the HTTP header. It then decides which data is needed to be gathered and which template should be used to render those. The content delivered by the PJAXR backend enriched web server is interpreted by the frontend module and replaces the related containers. This replacement is implemented using the ID attribute. Other methods identifying corresponding containers like e.g. XPath are not as generic in it's position on the page as the ID. This is important because single-page applications should be able to include content dynamically in every position on the page. PJAXR in combination with the History API makes it then possible to update the content and change the URL like it would be made by normal requests. This is possible with the use of the pushState method introduced in the History API. Back- and forward-buttons and bookmarks in a browser will work like on a normal request using this function.

4.4 Usage

4.5 PJAXR backends

5 Implementation

5.1 Sample PHP web application

5.2 PHP PJAXR backend

5.3 Spring PJAXR backend

5.4 jquery-pjaxr adjustments

5.5 concluding remarks

6 Evaluation

As one traditional testing model, we will evaluate the PJAXR sample application via blackbox tests. Testing AJAX is not trivial due to multiple programming- and markup-languages influencing it. One possibility to test web applications, as suggested in [5], is Selenium[2]. With this tool it is possible to generate automated tests for web applications.

To evaluate whether the application is crawlable or not is an important criteria whether PJAXR fulfills it goals. Finding all the content delivered in all different URLs in the sitemap should be the target to acquire. The crawled content should be similar to a non-dynamic HTML file, defined for every URL. Content which is not directly provided via an URL but asynchronously, like via an autocompletion, should not be relevant.

One way to crawl AJAX web applications, recommended in [6] is to use Crawljax[3]. It explores AJAX-based web applications by following every link recursively and saving the associated content. In this thesis the three endpoints of the sample project will be crawled by Crawljax to see whether all endpoints provide the same content or not.

Another way to evaluate whether PJAXR fulfils its goals, is testing if the Googlebot[4] will discover all the content provided. Again, all three endpoints will be tested to check, if all data is found by this technique. While Crawljax is intended to find not easily accessible content, Googlebot is intended to find content, matching design patterns[5] by Google. This fact makes it more challenging for PJAXR, not implementing these, to have good results in this test.

[2]http://www.seleniumhq.org/

[3]http://crawljax.com/

[4]http://google.com/bot.html

[5]https://developers.google.com/webmasters/ajax-crawling/

6.1 Selenium

6.2 Crawljax

6.3 Googlebot

6.4 Results

6.5 concluding remarks

7 Conclusion and future work

References

[1] Youri op't Roodt (2006). *The effect of Ajax on performance and usability in web environments*. Master Thesis. University of Amsterdam

[2] Kluge, Jonas and Kargl, Frank and Weber, Michael (2007). *The effects of the AJAX technology on web application usability*. WebIST 2007, Barcelona

[3] Mesbah, Ali (2009). *Analysis and Testing of Ajax-based Single-page Web Applications*. Ph.D. Thesis. TU Delft

[4] Duda, Cristian and Frey, Gianni and Kossmann, Donald and Matter, Reto and Zhou, Chong (2009). *AJAX Crawl: Making AJAX Applications Searchable*. Data Engineering, 2009. ICDE '09, Shanghai

[5] Lundmark, Simon (2011) *Automatic Testing of Modern Web Applications in an Agile Environment* Bachelor Thesis, Stockholm

[6] Mesbah, Ali and van Deursen, Arie and Lenselink, Stefan *Crawling Ajax-based Web Applications through Dynamic Analysis of User Interface State Changes* ACM Transactions on the Web (TWEB) 2012