UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Lare - A new technology for stateful single-page 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 Juni 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)

Abstract

The aim of this thesis is to improve and benchmark Lare - A new technology for stateful single-page applications. Lare is a front- and backend technology, developed on top of PJAX and the successor of PJAXR to easily improve web sites with the use of AJAX but without the disadvantages of it regarding browser functionality, SEO and user experience.

The thesis first presents Lare and describes how a realisation of it ideally should be structured. In a second stage a PHP backend, called PHP-lare and the matching Twig extension Twig-lare are introduced using those concepts. Additionally the existing JavaScript frontend lare.js and the django backend django-lare are refactored to match those guidelines, too. Finally Lare gets evaluated in form of benchmarks using cURL for plain HTTP requests and Selenium for retrieving web pages including all resources. For this purpose a sample web application is implemented using PHP, Twig and the matching Lare plugins.

The results of this evaluation show that Lare satisfies it's expectations regarding browser functionality and improves the load times of single-page applications.

Zusammenfassung

Ziel dieser Thesis ist es Lare - eine neue Technologie für stateful single-page applications zu verbessern und auf Geschwindigkeit zu testen. Lare ist eine auf PJAX basierende front- und backend Technologie die als Nachfolger von PJAXR entwickelt wurde um einfach web sites mit AJAX Unterstützung zu entwickeln, aber ohne die Nachteile im Bereich Browser Funktionalität, SEO und User Experience.

Die Thesis präsentiert anfangs Lare und beschreibt wie eine Realisierung idealer Weise strukturiert sein sollte. In einem zweiten Schritt werden ein PHP backend, genannt PHP-lare und eine passende Twig extension Twig-lare eingeführt, die diese Konzepte nutzen. Zusätzlich werden das existierende JavaScript frontend lare.js und das django backend django-lare in Hinsicht dieser Richtlinien überarbeitet. Abschließend wird Lare in Form von Benchmarks mit Hilfe von cURL für reine HTTP requests und Selenium zum empfangen kompletter web pages, inklusive aller Ressourcen, evaluiert. Für diesen Zweck wurde eine Beispiel web application basierend auf PHP, Twig und den passenden Lare Plugins implementiert.

Die Ergebnisse dieser Evaluation zeigen, dass Lare die Erwartungen bezüglich Browser Funktionalität erfüllt und die Ladezeiten von single-page applications verringert.

Contents

1 Introduction

At the beginning of the World Wide Web web pages were self-contained. And without a lot of effort they still are.

The content of a web page which is initially loaded is not changed until a new resource is requested by the user. One big change brought the invention of Asynchronous JavaScript and XML (AJAX). It introduced the possibility to change content without the need of requesting a full new web page on a different Uniform Resource Locator (URL). The approach only loading one full web page initially and then changing its content interactively is called single-page application. As presented in [1] single-page applications are more user-friendly than the common designs, e.g. due to lower load times and the elimination of interruption time experienced on common web applications. AJAX is nowadays, other than mentioned in [1] page 27, available in nearly every browser. As stated in [2] in 2013 only 1.1% of the internet users visiting the web site of the UK government did not get their JavaScript enhancements.

As mentioned in [1] a disadvantage of AJAX is that users are not able to save their websites as a bookmark, because while surfing on this page, the URL never changes. Another implication stated there is that the functionality of back and forward buttons in browsers is often not given. Additionally AJAX driven web applications may require multiple small server calls which might produce performance implications.

Single-page applications have another problem in the current time. As mentioned in [3] AJAX websites are difficult to examine for search engines and other crawlers, due to several challenges.

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. This means additional maintenance effort for web sites, especially when they are very dynamic.

In this thesis we will improve and evaluate the performance of a new technology called Lare. Together with Stephan Groß the author developed PJAXR the predecessor of Lare for using AJAX with it's advantages but trying to avoid the disadvantages explained before.

The result of this cooperation was a frontend-side implementation called *jquery-pjaxr*. As PJAXR also needs a backend to be implemented, I developed the first PJAXR backend called *django-pjaxr*.

The libraries lare.js and django-lare are introduced in this thesis as the successors of *jquery-pjaxr* and *django-pjaxr*. Additionally in this thesis the author will introduce a new Lare backend for PHP, *PHP-lare* and *Twig-lare* an extension for the Twig[2]

[1] https://developers.google.com/webmasters/ajax-crawling/docs/getting-started (Accessed: Juli 22, 2015)

[2] http://twig.sensiolabs.org/ (Accessed: June 16, 2015)

template-engine.

Later Lare will be evaluated to see whether the expectations regarding browser functionality are satisfied. Furthermore the single user performance will be tested through benchmarks using cURL and Selenium in combination with the WebDriver API, to see whether Lare is faster than normal web requests.

2 Fundamentals

To understand how Lare works a few technologies are required to know about. HTML, the markup language which is used to create web pages, is the first technology which is required to know about. Afterwards HTTP requests are declared as the fundamental transfer protocol of the World Wide Web. Dynamic content and synchronicity are the topics explained next, which is the problem which is addressed by AJAX and single-page applications, which are presented subsequently. The History API released in October 2014 and required by Lare is presented at the end of this chapter.

2.1 HTML

Hypertext Markup Language is the language which is used to create webpages. It allows to structure a web document semantically, but not to style it. Similar to XML it consists of hierarchically structured tags. Each tag may have attributes. Allowed attributes are defined per tag, e.g. an anchor tag may have an href attribute, which is not allowed on a div tag. There is a special tag, called ID which defines the unique identifier of a tag. Each value may not occur more than once on a page.

2.2 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 and sends it back as the response. The browser receives this web page interprets and renders it. Subsquently it sends HTTP requests to receive the images, CSS and scripts linked in this page.

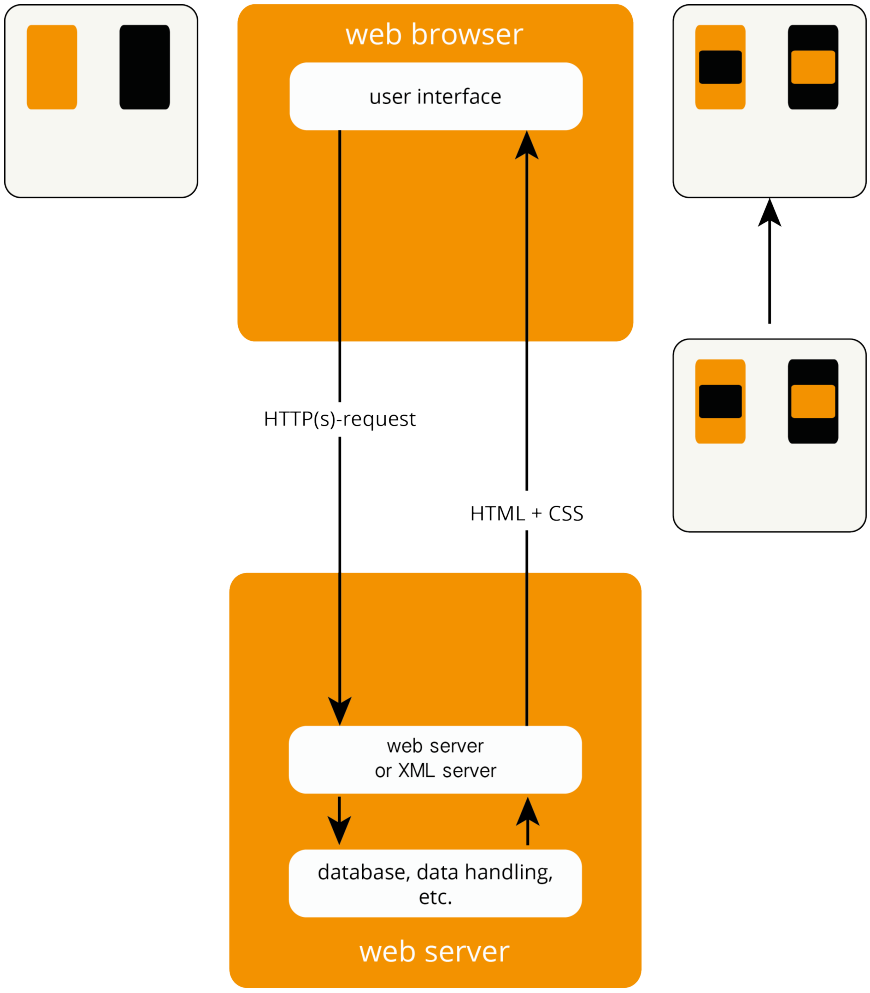

Figure 1: Component and communication diagram of HTTP

2.3 Dynamic content and synchronicity

Dynamic content in websites is content which is changed within an already fully loaded web page, without loading another full page including all resources. The normal HTTP request, which is described in 2.2, does not support dynamic changes of content. To retrieve new information the client has to request another full web page, including all resources and data which is necessary, which makes this model "synchronous". An "asynchronous" model is able to change content dynamically without having to load a whole page. AJAX, as shown in 2.4, is the most used technique for asynchronous web platforms.

2.4 AJAX

Asynchronous JavaScript and XML is a technology to implement dynamic web pages. JavaScript is used to make requests to a web server without loading a full new page. The response then is interpreted by an AJAX-engine.

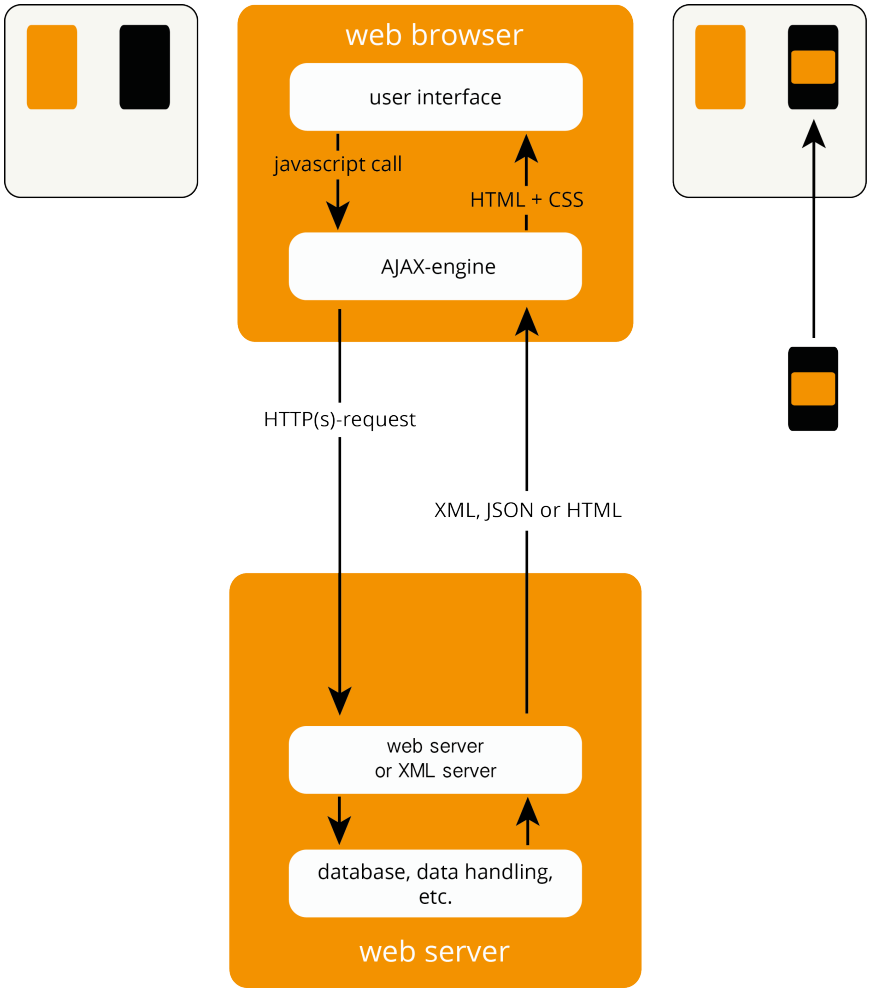

Figure 2: Component and communication diagram of AJAX

As shown in 1 a normal HTTP request by a browser forces the requests always to be synchronous, which means that the client has to wait for a fully loaded page on every request. Through the “Asynchronous JavaScript and XML”, short AJAX, this can be improved. The data flow in AJAX is very similar to the normal browser behaviour, but is using a new, third layer: the “AJAX-engine”. The first request to a web-server using AJAX is the complete same as one without AJAX with the exception, that one of the requested sources is a JavaScript, which instantiates a AJAX-engine. The following requests now are handled by this AJAX-engine, al-

lowing to request web-sources asynchronously. Without using AJAX after receiving HTML, the browser renders the whole page, even if there are only small changes to the page rendered before. AJAX instead only requests small parts of a website, most of the time in XML or JSON format, interprets it and then only adds, replaces or appends old content with the newly received.

2.5 Single-page applications

A single-page application is a web application or web site that only needs to one full web page load. Beside this there is no web page loaded completely at any point in the process anymore. Often content changes are made asynchronous and dynamically by AJAX in response to user actions. A disadvantage of a lot of single-page applications is the lack of browser history support. When changing the content the browser does not interpret it as a new page, but only changed content. This leads into a missing functionality of forward- and back-buttons in browsers.

Additionally a problem of SPAs is, that it has a huge impact on SEO. The web page is often rendered inside the browser, and not structured into different URLs. Those facts make it hard for a search-engine to discover it.

2.6 History API

The history API is part of HTML5 specification by W3C. It describes the API for interface for an History object which is part of the session history. This session history enables functionality like the back-and forward-buttons on browsers. Defined as a list of session history entries, it represents the browsing history. A session history entry may be a URL or a state object and may have additional information.

We need to influence this list in Lare via the History interface. This is possible via the window.history.pushState(data, title[, url]) method, which allows to add a new state into this list.

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 [4] is mentioned that AJAX applications have a better usability than non-AJAX websites. The same conclusion is made in [5], despite of the lack of browser navigation support. Beside the navigation problem another disadvantage is, as presented in [6], crawling AJAX applications is not trivial. One solution of this task is finding clickables and navigating to every page found by this. Nevertheless [6] 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[7]. 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.

3.1 Client-side templates

When building an asynchronous web application, a decision has to be made where to render data. A lot of AJAX applications use a JSON API which already predefines the outcome of this decision: JSON has to be interpreted by the AJAX-engine and rendered into HTML. As plain string modifications are difficult to maintain and on large applications not very handy, client-side templates become more and more widespread. Another advantage of this practice is the strong separation of the logic on the server and views on the client-side.

Besides their benefits, a few disadvantages come with them: After interpreting the first HTTP request other requests have to be made to load the templating engine and the data which should be rendered into the template. This means you have to make three requests:

- First an HTML file containing a link to the AJAX engine and the templates.
- Second the AJAX engine itself.
- Third the data which should be rendered into the templates, requested by the AJAX engine.

To avoid the need to wait on the third request, sometimes the initial requests already contains initial data, which results in the need of backend templates to render it and the client templates for further usage.

Another problem that needs to be solved is the SEO of those pages. Web pages implemented with client-side templates need a method to be visible for search engines. A common way to achieve that is using prerendered sites which are visible to search bots like the Googlebot. Even though Google interprets JavaScript generated content since May 2014[3] they still give the advice to degrade graceful when it comes to JavaScript compatibility.

[3] http://googlewebmastercentral.blogspot.no/2014/05/understanding-web-pages-better.html

3.1.1 Load time analysis

With client-side templating, even with the improvement of initial data, a client needs at least two requests for being able to render data. When further data is needed, it can be requested asynchronous, when not using a client-side MVC like in 3.2. So the client has to wait at least two round trip times or four delays. Additionally the frontend rendering time is relevant. Other than at server-side rendering the frontend rendering can not be cached.

Any further request is then made by the AJAX engine itself. An efficient web application which uses client-side rendering requests one url on which the response contains all needed data. In more complex projects it can happen that multiple requests have to be made until every needed data to render is available.

More requests can be required if more complexity is stored in the client.

3.2 Client-side MVC

In addition to only outsource the templating to the frontend, there are complete client-side MVC frameworks. Those frameworks use the model view controller pattern, where the controller has a connection to the web server.

Built on REST APIs they move all logic into the client-side. This approach is built primarily on the motivation to reduce the web server load and traffic.

In most cases, similar to simple client-side templates, mentioned in 3.1, client-side MVCs need three requests to render the first page.

Further requests then are not made to request URLs in the old fashioned way, but to retrieve objects through a REST API. Object manipulation, logical methods and everything, normally implemented in a server backend should be in the frontend in those frameworks.

Using this pattern, you will still have the same problems as with client-side templates, but on another level. The web server will not gather all information which is needed, send it to the templating engine which then renders those. The client itself decides which information it needs and requests it from the server.

Load and traffic of the server in this pattern is relatively low, because it will only create, update, delete or display objects in the database. More logical functions on the server-side are not needed.

Clients, especially mobile devices and slow computers, might struggle with the load of work instead.

3.2.1 load time analysis

As shown before, this methods needs 3 requests to display the initial web page. To get into more detail, the client needs to wait for the first request to be completed and then the AJAX engine has to be loaded completely. After this third request by the client is made. He then waited three round trip times, or six delays, plus the load time by the server and download time of those three requests.

Any further request is then made by the AJAX engine itself. Normally on strict client-side MVCs per web page multiple small requests have to be made. On each of this method strikes the delay two times.

3.3 Hash-Bang URLs

In order to have a good SEO it is recommended by Google to have hashbang URLs. This pattern defines that in the URL the *hash part* should start with "#!" instead of only "#". Finding this combination let crawlers know that the site provides the AJAX application and additionally full page snapshots. When a crawler e.g. finds a site as http://example.com/#!site=test it crawls
http://example.com/?_escaped_fragment_=site=test instead. This is done, because everything after the "#" will not be sent to the server, but is only recognized by the browser. To let the server know that you want to request a specific page this URL modification is needed. At this "?_escaped_fragment_=" URL a snapshot of the full page should be available. This means, instead of only providing a few parts of the page required by AJAX, the whole HTML DOM should be delivered.

This technique was developed for URL changing by JavaScript without a full page reload. Old browsers without the implementation of the History API are not able to change the URL without a full load of a page. To gain navigation functionality on single-page applications the only way was to change the hash value in the URL, which does not call a page load, but updates the URL and pushes it to the navigation history.

3.4 HIJAX

One way to implement single-page applications using AJAX is to use the pattern of HIJAX.

When using this pattern, you plan the web site with AJAX. Changes of the HTML markup should be avoided. A good way to do that is using classes in the link, which later gains semantics in JavaScript. But when implementing the web site you first implement it as a traditional non AJAX web site. Every web page should be delivered fully and links should be linking to real web pages. No JavaScript should be needed to run the site.

When the web site is implemented completely the event listeners on the links will be *hijacked* and processed by JavaScript. This script then creates a new XMLHttpRequest which requests only the updated parts of the page and renders the response.

When this pattern is used the web site degrades gracefully. This means even when JavaScript is not available or blocked the web site still works completely.

3.5 PJAX

PJAX, introduced 2011 by Chris Wanstrath, is a jQuery plugin that uses AJAX and pushState to deliver a fast browsing experience with real permalinks, page titles, and a working back button.[4]

PJAX let the browser replace a container in a page by requesting a URL in certain way like adding a "?pjax" query parameter. When requesting a page on a server like "http://example.com/test/" a full page is responded, with "http://example.com/test/?pjax" only one container will be rendered. In combination with HIJAX it is possible to have links like *test* but PJAX performs a request to */test/?pjax* instead. This lets crawlers which are not able to interpret JavaScript crawl all normal pages. But normal browsers will have the advantages of AJAX with only replacing one container instead.

PJAX has the ability to either send one specific container or a full page per URL. It has no ability to send containers according to the browsers previous page.

[4] https://github.com/defunkt/jquery-pjax#introduction

4 Lare

4.1 Introduction

Lare, lightweight AJAX replacement engine, is built on top of PJAX and successor of PJAXR.

The idea of Lare, previous PJAXR, is to have the advantages of AJAX while trying to avoid it's disadvantages. Introduced in Juli 2013, as an extended version of PJAX, it allows to replace multiple containers with a single request, instead of the limit to replace only one container. Matching by the ID of an container it passed the duty to define which container should be replaced from the client-side to the server-side with setting the correct IDs. Introducing the tags <lare-head>, besides a <lare-body> element, it reaches the ability to change meta tags, the page title and replace containers with only one request.

This means it achieves the same UX improvements and reduced load times as classical AJAX. On the other hand single-page applications using Lare are easily crawlable by the most used crawlers without additional efforts. As successor of PJAXR it also uses the pushState function of the History interface to achieve full functionality of browsers incl. back- and forward buttons. Lare is a generic solution for single-page applications and can be implemented in nearly every web application.

4.2 Concept

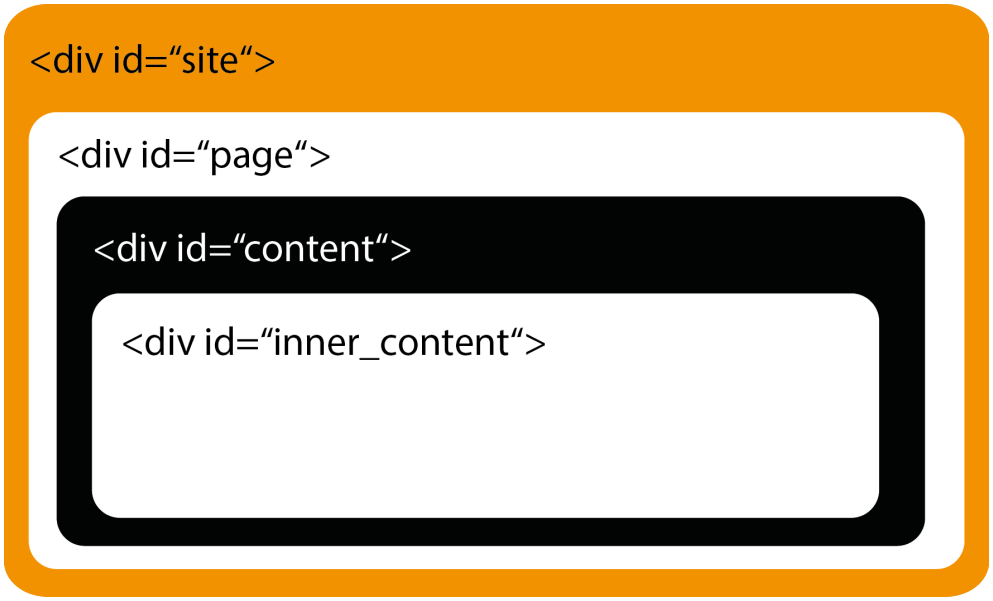

Figure 3: Basic HTML structure of a web application using Lare

Every single-page application which uses Lare has a hierachical namespace structure. Typically the namespace consists of 4 levels: 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 3, a container with an ID telling which part of the namespace it belongs to.

After the Lare frontend is initialized it hijacks events like clicks on links and enriches the request with the current website's namespace. A Lare 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 Lare frontend retrieves the response, replaces the containers at the website and updates the current namespace.

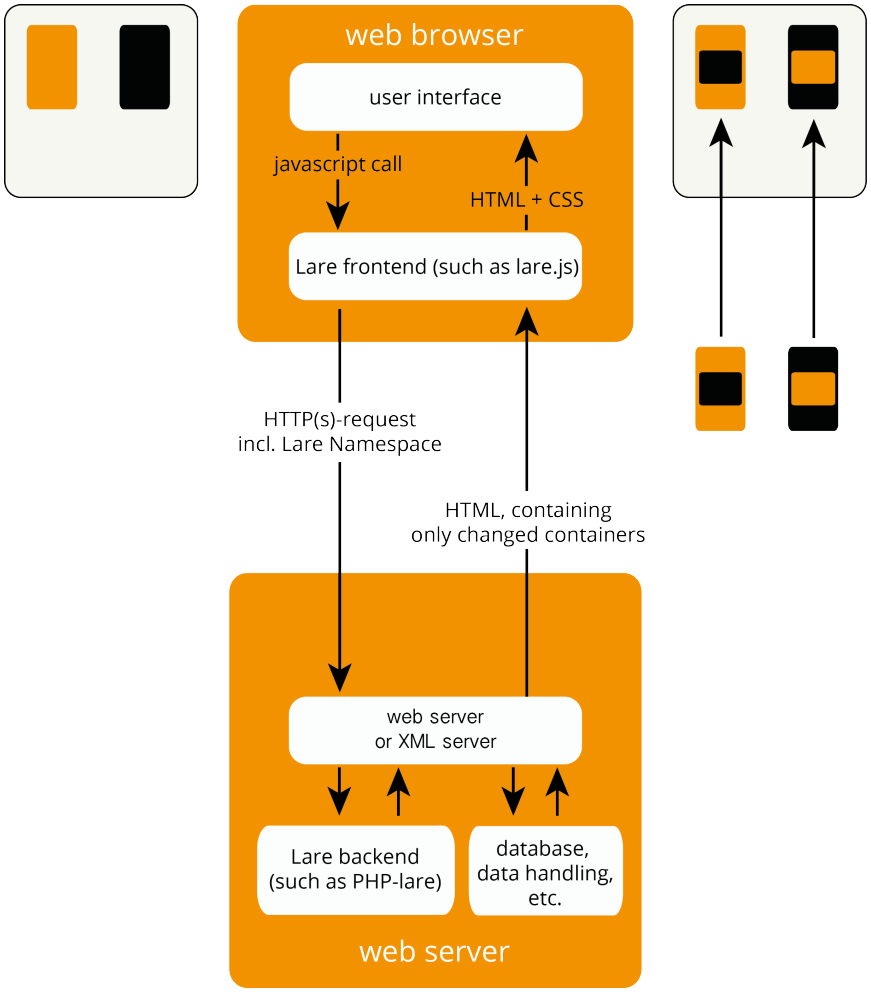

Figure 4: Component and communication diagram of Lare

As shown in fig. 4 the overall structure of Lare is similar to normal AJAX requests (compare fig. 2). The AJAX engine of Lare is lare.js. In addition to AJAX, Lare has a backend which helps to reduce server load and maps responses to the given namespace. Optimally this analyzation is done as first action when retrieving a request by the web server. Backend queries such database queries and such can then by left out if they are not necessary in the current namespace situation.

4.3 Realization

To load a page, the first request is a normal HTTP request followed by a JavaScript script initializing the Lare frontend module. Further requests to the same host are then initiated by this module. The HTTP header of these requests is extended by the namespace of the current website. The web server using a Lare backend compares the namespace of the requested resource and the namespace in 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 Lare 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. X-Path are not as generic in it's position on the page as the ID.

Usage of 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.

Normal HTTP requests and Lare requests request the same URL which makes it easy to crawl. While on normal requests a full web page is responded Lare requests only get the changed containers as response. Search engines and other crawlers are able to crawl every link it can find on a page and interpret it as normal web pages without any deficit.

4.3.1 Lare frontend

A Lare frontend has as mentioned before a few things to be implemented. The first requirement is the ability to hijack page changes. This can be implemented by e.g. replacing the default event listeners for anchor-tags. A new event listener then has to implement an enrichment of the HTTP header by the namespace using the "HTTP-X-LARE" key. On initial requests the current namespace should be served as an attribute at the <body> tag called data-lare-namespace. Lare requests serve a new namespace as content of a tag <pjaxr-namespace>.

As heart of Lare dynamic URL changes have to be implemented after getting the response. This should be done by using the pushstate function.

The biggest functionality of the Lare frontend is the replacement. A response of a Lare request is divided into a <lare-head> tag, a <lare-body> tag and the <lare-

namespace> tag. Elements of the <lare-head> should be a new <title> tag and <meta> tags matching the new content. Additionally scripts or styles can be linked in this section when they are needed by the new content.

The <lare-body> tag will contain the new content which should be replace old one. Each container inside the <lare-body> will be searched by it's ID inside the current page and will then replace it's predecessor.

4.3.2 Lare backend

A Lare backend should first interpret the "HTTP-X-LARE" item in the HTTP header. Every layer of the namespace should have it's own name. As a default naming convention the layers should have the names *site*, *page*, *content*, *inner_content* from start to the end. Per layer a variable should save the matching state. Those variables have to be accessible by views and controllers to give them the possibility to decide which backend requests should be made and which templates should be used.

4.3.3 Lare templating

To avoid a lot of overhead when using Lare a specific templating system is recommended.

A default template for the first request could be like this:

```
<!Doctype html>
<html>
<head>
  <title></title>
  ...
</head>
<body data-lare-namespace="Lare.Namespace">
  <div id="site">
    ...
  </div>
</body>
</html>
```

As shown above, it is a normal HTML5 template. The only specific code you have to write when using Lare is the *data-lare-namespace* attribute at the body tag.

The Lare template has to be formed like this:

```
<lare-head>
  <title></title>
  ...
</lare-head>
<lare-body>
  <div id="site">
    ...
    <div id="page">
      ...
    </div>
    ...
  </div>
</lare-body>
<lare-namespace>Lare.Namespace</lare-namespace>
```

The <lare-head> tag is the conterpart to the <head> tag, <lare-body> to <body>. Instead of an attribute in the <lare-body> tag, the namespace will be delivered in the <lare-namespace> tag.

For performance improvements it is intended to have a hierarchical structure as seen in fig. 3.

When e.g. the first namespace matches, the <lare-body> could only deliver the page conatiner:

```
<lare-body>
  <div id="page">
    ...
  </div>
</lare-body>
<lare-namespace>Lare.AnotherNamespace</lare-namespace>
```

5 Implementation

decisions Lare Object Considering the MVC pattern we have all logic in one object, in this case the *Lare* object. It is a singleton in the request scope. When receiving a request the server creates it and analyzes if the current request is a Lare request and checks if there is a namespace.

5.1 PHP-lare

PHP-lare is a general Lare backend to used in PHP. It builds the base of every other Lare module in PHP, especially for template engines.

Lare is a implemented as a request-scoped Singleton. In PHP a singleton is implemented as a class, which provides only static methods and static attributes.

5.1.1 API

When including the Lare.php automatically a Singleton named Lare will be created.

- Lare::is_enabled()

  Returns true if the current request is a Lare request, otherwise false.

- Lare::set_current_namespace($namespace)

  Sets the namespace of the current request to $namespace.

- Lare::get_current_namespace()

  Returns the namespace of the current request.

- Lare::get_matching($extension_namespace = null)

  $extension_namespace is an optional paramenter, to check the matching to a given namespace. If $extension_namespace is not given, the matching will be done against the namespace of the current request.

  Returns the most specific matching namespace level.

- Lare::matches($extension_namespace = null)

  $extension_namespace is an optional paramenter, to check the matching to a given namespace. If $extension_namespace is not given, the matching will be done against the namespace of the current request.

  Returns true if the whole namespace is matching, otherwise false.

5.2 Twig-lare

Twig-Lare brings Lare functionality to the template engine Twig[5]. Twig is used by Symfony, a framework which is used in e.g. Drupal 8, eZPublish, phpBB and Sylius.

Twig-Lare is a implemented as a Twig extension. It consists of a TwigTokenParser, a anonymous TwigSimpleFunction and a global variable. The TwigTokenParser Twig_Lare_TokenParser_LareExtends is the heart of the extension. It provides the possibility to use the tag {% lare_extends %} in the way it may be used in django. To prevent multiple extend tags, including the default twig tag *{% extends %}* it throws a Error if either the default tag or {% lare_extends %} was already used. Additionally we ensure that it is not called inside a block tag.

```
{% lare_extends "::__base.twig" "Lare.Namespace" "::__lare.twig" %}
{% block page %}
  <div id="page">
    ...
  </div>
{% endblock page %}
{% block lare-namespace %}{{ current_lare_namespace }}{% endblock lare-nam
```

The example above shows the usage of this tag. When the current namespace is not inside Lare.Namespace it extends to the __base.twig template because the second namespace does not match then. But when the current namespace is *Lare.Namespace*, then it extends __lare.twig.

As the namespace matching occurs on the second level in this example, the overriden block should be the second, with the naming in this thesis *page*. Inside this block a container with the according ID has to be placed.

5.2.1 API

- {% lare_extends $default_template %}

  Extends $default_template like the original {% extends $default_template %}.

- {% lare_extends $default_template $lare_namespace %}

  Extends ::__lare.html if $lare_namespace is matching, otherwise it extends $default_template.

- {% lare_extends $default_template $lare_template $lare_namespace %}

  Extends $lare_template if $lare_namespace is matching, otherwise it extends $default_template.

[5] http://twig.sensiolabs.org/

5.3 django-lare

django-lare was the first backend of Lare. It was introduced as a single object containing logic and template tools. After implementing PHP-lare we decided to change the structure of the django backend towards the new segmentation.

Similar to the combination of PHP-lare and Twig-lare, django-lare consists of a Lare object as in the PHP backend and implements the same templating tools as the Twig extension. Still as one package it is available via *pip install django-lare* command.

5.3.1 API

- {% lare_extends $default_template %}

  Extends $default_template like the original {% extends $default_template %}.

- {% lare_extends $default_template $lare_namespace %}

  Extends ::__lare.html if $lare_namespace is matching, otherwise it extends $default_template.

- {% lare_extends $default_template $lare_template $lare_namespace %}

  Extends $lare_template if $lare_namespace is matching, otherwise it extends $default_template.

5.4 lare.js

lare.js, as successor of jquery-pjaxr, is the frontend engine for Lare, using AJAX to communicate to the server. jQuery-pjaxr was introduced as an extended version of jquery-pjax, allowing to replace multiple containers with one request, instead of only one. Matching by the ID of an container it was not the job of the frontend to define which container should be replaced, but the backend with giving the correct IDs.

Introducing lare.js achieved the ability differ the current page via namespaces. describe lare.js

5.5 concluding remarks

6 Evaluation

describe test design Lare is tested based on a sample web application. It provides different type of sites which are designed to perform the different aspects of this evaluation.

To evaluate Lare we first test it's functionality. We check if the desired content is delivered and if Lare is actually performing like expected.

It is not easy to test AJAX web applications. As seen in [12] there is a lack of good testing tools, especially when it comes to white-box testing. Selenium is mentioned there as a good tool for black-box testing. It is also suggested in [10] for a non-abstract HTTP request performance testing, which can represent the requests of a single user.

In [11] a new automated testing technique is introduced, again based on Selenium.

As to evaluate Lare there is no need to actually test the whole application, but only to check whether Lare works, Selenium in our case is sufficient.

We make specific requests and want an specific answer of it. Especially we want to have the same content rendered through Lare as through normal HTTP requests.

Selenium additionally makes it possible to use different WebDrivers, in this thesis FireFox and Chrome are used. This is important because of the different implementations of browsers' features.

To test the performance, we will use two technologies. [13] First of all cURL based tests will be done. Those tests will focus on the first response, containing the markup. This will show how Lare influences the webserver.

Additionially the webapp will be benchmarked by using the Chrome Network Tools. This method provides the possiblity to check whether further requests for scripts, images, etc. are influenced. The Chrome Network Tools will show the actual load time the user has to wait for, until the whole page is loaded.

To make the tests as representive as possible caching in the relevant layers will be enabled and disabled if to see whether it influences the results or not. Some caching algorithms can not be easily disabled. E.g. hardware and hard drive caching will not be disabled in our tests. As those caching algorithms will not effect the results much, we let them enabled.

Mysql's caching method Query Caching will be enabled and disabled. Twig, the template engine allows caching, which will be enabled and disabled as well. Additionally all tests are performed on a local machine and a remote server, to see whether the latency takes effect in the performance of Lare.

We will distinguish between static pages without database queries and dynamic pages which have those. Every page relevant for the test will be requested in different modes. First of all every site will be requested normally. After that it will be requested with Lare enabled. As Lare should only influence subsequent requests, every page will be tested with HTTP headers from different sources, imitating those requests.

6.1 Sample web application

The sample web application used to test Lare in this thesis is implemented in PHP. To implement Lare we used lare.js in the frontend, Twig-lare and PHP-lare for the backend.

We use the MVC design pattern, but with a slightly different naming. For web applications it is common to use the name *template* for what is called view in the MVC. As "controller" is not a common name PHP web applications as well, we use the phrase *view* for those. With this we follow the naming of django[6].

Instead of using models for this benchmark evaluation, we are using raw SQL queries to ensure the same queries are made every time. Nevertheless it is prepared to inherit from the BaseModel class to be create models for later presentation purposes.

The sample web application consists of 2 static pages */home/* and */imprint/*.

To demonstrate dynamic Web pages we used the Delicious Dataset[7]. This is available under */tags/*.

On the left side is a list to sub pages where the tags are categorized by it's starting character. Per alphabetic character one of those pages exists, additionally one page for tags starting with numbers and one starting with non alphanumeric characters. Behind each *category* the count of tags in this category is displayed. This list is available on every sub page under the */tags/* url.

On the right side a paginated list of all distinct tags in the current category is shown. It is sorted alphabetically and 30 tags per page are displayed.

We are using the namespaces like shown in tab. 1.

URL	Site	Page	Content
/	Lare	Home	
/imprint/	Lare	Imprint	
/tags/	Lare	Tags	all
/tags/a/	Lare	Tags	a
/tags/.../	Lare	Tags	...

Table 1: Namespaces of the sample web application

6.2 Tests

To test the performance of Lare a few different types of requests are needed to benchmark. The reference level will be a normal HTTP request to each web page.

The tested pages are available at the URLs:

[6]https://docs.djangoproject.com/en/1.8/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names

[7]http://fabianabel.de/data/mypes-www2010.html

- /
- /imprint/
- /tags/p/
- /tags/p/2/

Additionally the web pages will be requested via Lare. To achieve this at cURL we extend the request's header with the corresponding *HTTP-X-LARE* namespace.

With Selenium we will later test the pages with real web browsers to get results which are closer to reality. The tested page to page requests are:

- Self:
  - / to /
  - /imprint/ to /imprint/
  - /tags/p/ to /tags/p/
  - /tags/p/2/ to /tags/p/2/
- Static page matching Site-Namespace:
  - / to /imprint/
  - /imprint to /
- Dynamic page matching Site-Namespace:
  - / to /tags/p/
  - /imprint/ to /tags/p/
- Dynamic page to static page:
  - /tags/p/ to /
  - /tags/p/2/ to /
  - /tags/p/ to /imprint/
  - /tags/p/2/ to /imprint/
- Dynamic page matching Page-Namespace:
  - /tags/p/ to /tags/p/2/
  - /tags/p/2/ to /tags/p/

6.3 cURL

curl usage of test design We are using cURL to test the plain load times of the initial request of normal HTTP requests and Lare requests. Those cURL requests do not include additional data like images, scripts and stylesheets, just the plain HTML.

The bash script "curl_tests.sh" provides the functionality to run those cURL tests. To use it run "/.../evaluation/curl/curl_tests.sh URL MAX_RUNS LARE_NAMESPACE CACERT".

URL dedicates where the web application is available, e.g. *http://lare.iekadou.com*, MAX_RUNS defines how many cURL requests per page should be made. LARE_NAMEPSACE defines the namespace, which should be used when testing the Lare requests, e.g. Lare.Tags.p. CACERT is the only optional paremeter, which should only be set when requesting an SSL server. It then should be the path to the CACERT file.

To test all relevant sites we defined a variable PAGES inside the script, containing URLs */*, */imprint/*, */tags/p/* and */tags/p/2/*. For each page of those we make a normal request, followed by a Lare request, with the namespace defined in LARE_NAMESPACE. To get better test results we repeat those two requests *MAX_RUN* times. The output represents then the load times needed in seconds.

For the results in this thesis, markes as local, we use a Macbook Pro Retina Early 2013 (2,7 GHz, 16GB Ram). As web server we are running an Apache via Mamp Pro 3.1 for Mac OSX with PHP 5.3.6. As database server we are using MySQL in version 5.5.42.

When running the benchmarks on an external server we are using a virtual server running OS: Ubuntu 12.04.5 LTS. It has a Intel(R) Xeon(R)CPU E5520, 4 GB ram, PSA version 11.5.30, Mysql version 5.5.41 and php version 5.3.10.

Additionally we test every request with database caching (DBC) enabled, disabled and template caching(TC) enabled and disabled.

6.3.1 Results

You can see the full results of our benchmarks in tables 2 and 3.

First we will take a look at the results of the local tests.

Those tests show that the load times of Lare requests of static pages in general are quite the same then on normal HTTP requests. Lare does not seem to have a huge effect on static pages without a lot of heavy operations like backend queries. It makes the responses a bit smaller, but no logical operations can be avoided.

Template caching has not a big effect on the difference between Lare and normal requests, but the absolute load times get a bit down. Database caching has not even on the absolute times an effect at those static pages, because no backend queries are made.

When requesting a dynamic page and sending the namespace of a static page like in our tests "Lare.Tags.Imprint" the results of Lare are still similar to normal requests. This can be explained by the fact that almost the whole page needs to be changed, including every database query.

The results on dynamic pages with a related namespace, in our case "Lare.Tags.p" differ a lot. When database caching is enabled the load times of Lare are around 90% with template caching enabled and about 80% of the normal load times with disabled template caching.

Without database caching this these times they are **7%** with template caching and **10%** without.

Those improvemnts are caused by the amount of database queries that can be avoided. This also explains the differences between enabled and disabled database caching settings.

When we take a look at the external tests we can see the same pattern. The effect of Lare on dynamic pages is a bit smaller. Wihtout database caching they are about **12%** with template caching and **16%** without.

Why does it have not this huge effect anymore?

This can be explained with the latency. In our tests it was 40ms. HTTP requests need two Round Trip Times, or four times the latency for connection establishment. When we substract those 160ms from the normal and the Lare requests we see ratios similar to the local tests.

In general we can say, that Lare has more effect the more time of backend logic like database queries can be avoided.

URL	Normal loadtime	Lare loadtime	Namespace	TC	DBC
/	0.03740	0.03600 (96,26%)	Lare.Tags.p	+	+
/imprint/	0.03590	0.03620 (100,84%)	Lare.Tags.p	+	+
/tags/p/	0.03950	0.03480 (88,10%)	Lare.Tags.p	+	+
/tags/p/2/	0.03940	0.03550 (90,10%)	Lare.Tags.p	+	+
/	0.03370	0.03610 (107,12%)	Lare.Tags.Imprint	+	+
/imprint/	0.03570	0.03450 (96,64%)	Lare.Tags.Imprint	+	+
/tags/p/	0.03910	0.04080 (104,36%)	Lare.Tags.Imprint	+	+
/tags/p/2/	0.04040	0.03960 (98,02%)	Lare.Tags.Imprint	+	+
/	0.03400	0.03380 (99,41%)	Lare.Tags.p	+	-
/imprint/	0.03430	0.03480 (101,46%)	Lare.Tags.p	+	-
/tags/p/	**1.02180**	**0.07230 (7,08%)**	**Lare.Tags.p**	**+**	**-**
/tags/p/2/	1.03130	0.07310 (7,09%)	Lare.Tags.p	+	-
/	0.03440	0.03650 (106,10%)	Lare.Tags.Imprint	+	-
/imprint/	0.03540	0.03480 (98,31%)	Lare.Tags.Imprint	+	-
/tags/p/	1.02390	1.04130 (101,70%)	Lare.Tags.Imprint	+	-
/tags/p/2/	**1.03440**	**1.04980 (101,49%)**	**Lare.Tags.Imprint**	**+**	**-**
/	0.07210	0.06660 (92,37%)	Lare.Tags.p	-	-
/imprint/	0.06690	0.06410 (95,81%)	Lare.Tags.p	-	-
/tags/p/	1.08420	0.10720 (9,89%)	Lare.Tags.p	-	-
/tags/p/2/	**1.09660**	**0.10900 (9,94%)**	**Lare.Tags.p**	**-**	**-**
/	0.06920	0.06860 (99,13%)	Lare.Tags.Imprint	-	-
/imprint/	0.06840	0.06520 (95,32%)	Lare.Tags.Imprint	-	-
/tags/p/	1.09020	1.09620 (100,55%)	Lare.Tags.Imprint	-	-
/tags/p/2/	1.09000	1.07740 (98,84%)	Lare.Tags.Imprint	-	-
/	0.07070	0.06630 (93,78%)	Lare.Tags.p	-	+
/imprint/	0.06710	0.06510 (97,02%)	Lare.Tags.p	-	+
/tags/p/	0.08440	0.06990 (82,82%)	Lare.Tags.p	-	+
/tags/p/2/	0.08550	0.06930 (81,05%)	Lare.Tags.p	-	+
/	0.07010	0.06790 (96,86%)	Lare.Tags.Imprint	-	+
/imprint/	0.06770	0.06510 (96,16%)	Lare.Tags.Imprint	-	+
/tags/p/	0.08540	0.08230 (96,37%)	Lare.Tags.Imprint	-	+
/tags/p/2/	**0.08390**	**0.08220 (97,97%)**	**Lare.Tags.Imprint**	**-**	**+**

Table 2: cURL results on a local machine

URL	Normal loadtime	Lare loadtime	Namespace	TC	DBC
/	0.16520	0.14070 (85,17%)	Lare.Tags.p	+	+
/imprint/	0.14360	0.12850 (89,48%)	Lare.Tags.p	+	+
/tags/p/	0.15420	0.13180 (85,47%)	Lare.Tags.p	+	+

/tags/p/2/	0.14610	0.14200 (97,19%)	Lare.Tags.p	+	+
/	0.14050	0.13350 (95,02%)	Lare.Tags.Imprint	+	+
/imprint/	0.13690	0.13630 (99,56%)	Lare.Tags.Imprint	+	+
/tags/p/	0.14470	0.14170 (97,93%)	Lare.Tags.Imprint	+	+
/tags/p/2/	0.14230	0.13310 (93,53%)	Lare.Tags.Imprint	+	+
/	0.14860	0.14580 (98,11%)	Lare.Tags.p	+	-
/imprint/	0.14990	0.13830 (92,66%)	Lare.Tags.p	+	-
/tags/p/	**1.55500**	**0.19070 (12,26%)**	**Lare.Tags.p**	**+**	**-**
/tags/p/2/	1.52150	0.19600 (12,88%)	Lare.Tags.p	+	-
/	0.14160	0.14080 (99,44%)	Lare.Tags.Imprint	+	-
/imprint/	0.14300	0.13790 (96,43%)	Lare.Tags.Imprint	+	-
/tags/p/	1.50930	1.54400 (102,30%)	Lare.Tags.Imprint	+	-
/tags/p/2/	**1.51030**	**1.56460 (103,59%)**	**Lare.Tags.Imprint**	**+**	**-**
/	0.20450	0.18600 (90,95%)	Lare.Tags.p	-	-
/imprint/	0.20350	0.18130 (89,09%)	Lare.Tags.p	-	-
/tags/p/	1.64180	0.30550 (18,61%)	Lare.Tags.p	-	-
/tags/p/2/	**1.65950**	**0.24830 (14,96%)**	**Lare.Tags.p**	**-**	**-**
/	0.20390	0.19800 (97,11%)	Lare.Tags.Imprint	-	-
/imprint/	0.20440	0.18360 (89,82%)	Lare.Tags.Imprint	-	-
/tags/p/	1.64260	1.62680 (99,04%)	Lare.Tags.Imprint	-	-
/tags/p/2/	1.63810	1.67440 (102,22%)	Lare.Tags.Imprint	-	-
/	0.20940	0.20020 (95,61%)	Lare.Tags.p	-	+
/imprint/	0.20300	0.19040 (93,79%)	Lare.Tags.p	-	+
/tags/p/	0.22250	0.20760 (93,30%)	Lare.Tags.p	-	+
/tags/p/2/	0.20940	0.20160 (96,28%)	Lare.Tags.p	-	+
/	0.20800	0.20500 (98,56%)	Lare.Tags.Imprint	-	+
/imprint/	0.19970	0.19480 (97,55%)	Lare.Tags.Imprint	-	+
/tags/p/	0.21420	0.20200 (94,30%)	Lare.Tags.Imprint	-	+
/tags/p/2/	**0.21630**	**0.22510 (104,07%)**	**Lare.Tags.Imprint**	**-**	**+**

Table 3: cURL results on a external machine

6.4 Selenium

Selenium usage of test design To benchmark full page loads including all resources we are using Selenium. This *record and play* tool builds an interface for a usage of multiple WebDrivers. Those WebDrivers are provided by most modern browsers like Chrome and Firefox. For the results in this thesis we used the Firefox WebDriver.

We request a page via a normal HTTP request first. Afterwards we request the same page again via Lare. We are using a lot of different page to page requests, to

see where Lare has a bigger or smaller effect on the load times.

Like in 6.3 we test every page on a local server and an external webserver and we disabled and enable database and template caching.

6.4.1 Results

The results of the Selenium WebDriver tests are different to the ones made with cURL.

We first will take a look at the local tests again. Other than in the cURL test results we see a huge effect of Lare even on the static pages. An average Lare load time of about 35% with disabled template caching and 9% with enabled template caching of the normal load time can be seen here. 2 Other than on a normal request, the browser does not need to load the resources like images, scripts and styles. This means Lare requests often only need one request to make a page change. A normal requests needs to load, or at least compare, about 9 resources for the same action.

Again requesting a static page does not show any changes when it comes to database caching.

When it comes to dynamic pages we have to take a more detailed look to the requests. Requesting a dynamic page from another Lare namespace with disabled database caching takes almost the same time like a normal request. This is because the same backend requests need to be done. Those uncached database requests take as we can see in table 2 the majority of time.

With disabled caching in the database and in a related namespace instead, Lare has a huge effect. Enabled template caching improving the load times from 5% with a disabled template caching to 3,5%.

When requesting the same pages with enabled database caching we see a Lare load time of about 16% with unrelated namespaces. Related namespaces in the same configuration lead to values of about 10%.

Enabled template caching lower the load times of the normal and Lare requests, but not of the resources. This makes Lare more effective, because the load times relative to the normal requests incl. all resources decrease.

From	To	Normal loadtime	Lare loadtime	TC	DBC
/tags/p/	/tags/p/2/	60.0648983ms	7.3076173ms (12.17%)	+	+
/tags/p/2/	/tags/p/	51.5302378ms	5.7457083ms (11.15%)	+	+
/tags/p/	/	58.0190003ms	5.2543307ms (9.06%)	+	+
/tags/p/2/	/	45.1883941ms	4.159042ms (9.20%)	+	+
/tags/p/	/imprint/	46.8359346ms	4.3459044ms (9.28%)	+	+
/tags/p/2/	/imprint/	47.1970978ms	4.2372439ms (8.98%)	+	+
/	/	56.6017328ms	5.4929997ms (9.70%)	+	+
/imprint/	/imprint/	43.3243723ms	4.1624412ms (76.67%)	+	+
/tags/p/	/tags/p/	54.4607062ms	5.5379726ms (10.17%)	+	+
/tags/p/2/	/tags/p/2/	52.5258287ms	5.7707781ms (10.99%)	+	+

/	/tags/p/	66.8794388ms	10.4380685ms (15.61%)	+	+
/imprint/	/tags/p/	52.089504ms	9.1059678ms (17.48%)	+	+
/	/imprint/	57.6817111ms	5.884066ms (10.20%)	+	+
/imprint/	/	45.9628159ms	4.6333061ms (10.08%)	+	+
/tags/p/	/tags/p/2/	1243.4891679ms	46.5173918ms (3.74%)	+	-
/tags/p/2/	/tags/p/	1236.3124832ms	43.4368424ms (3.51%)	+	-
/tags/p/	/	53.4060902ms	5.1304958ms (9.61%)	+	-
/tags/p/2/	/	45.4183183ms	4.206772ms (9.26%)	+	-
/tags/p/	/imprint/	46.2665835ms	4.175386ms (9.02%)	+	-
/tags/p/2/	/imprint/	46.0661066ms	4.2264232ms (9.17%)	+	-
/	/	53.5617939ms	5.4352539ms (10.15%)	+	-
/imprint/	/imprint/	45.0215085ms	4.4406ms (65.68%)	+	-
/tags/p/	/tags/p/	1234.5362779ms	44.369364ms (3.59%)	+	-
/tags/p/2/	/tags/p/2/	1235.7906042ms	45.1390014ms (3.65%)	+	-
/	/tags/p/	1246.2746233ms	1197.6706971ms (96.10%)	+	-
/imprint/	/tags/p/	1231.1385665ms	1191.9845789ms (96.82%)	+	-
/	/imprint/	49.8334108ms	6.1112746ms (12.26%)	+	-
/imprint/	/	44.540915ms	4.2382487ms (9.52%)	+	-
/tags/p/	/tags/p/2/	1281.1725193ms	69.312793ms (5.41%)	-	-
/tags/p/2/	/tags/p/	1271.3418256ms	67.1285484ms (5.28%)	-	-
/tags/p/	/	81.102332ms	27.2707205ms (33.63%)	-	-
/tags/p/2/	/	69.681426ms	25.9004252ms (37.17%)	-	-
/tags/p/	/imprint/	68.4034544ms	24.2320953ms (35.43%)	-	-
/tags/p/2/	/imprint/	66.3318697ms	23.254195ms (35.06%)	-	-
/	/	80.4801214ms	27.0665394ms (33.63%)	-	-
/imprint/	/imprint/	66.282428ms	23.5402204ms (35.52%)	-	-
/tags/p/	/tags/p/	1277.0515153ms	67.8621459ms (5.31%)	-	-
/tags/p/2/	/tags/p/2/	1266.6883795ms	68.0796981ms (5.37%)	-	-
/	/tags/p/	1274.8229527ms	1224.9995695ms (96.09%)	-	-
/imprint/	/tags/p/	1273.1817246ms	1221.7685327ms (95.96%)	-	-
/	/imprint/	71.5289704ms	25.8452978ms (36.13%)	-	-
/imprint/	/	70.3443044ms	26.8150602ms (38.12%)	-	-
/tags/p/	/tags/p/2/	96.6278896ms	31.5540318ms (32.66%)	-	+
/tags/p/2/	/tags/p/	84.6051469ms	29.4296049ms (34.78%)	-	+
/tags/p/	/	78.0617908ms	28.0351468ms (35.91%)	-	+
/tags/p/2/	/	70.2056237ms	26.3221693ms (37.49%)	-	+
/tags/p/	/imprint/	66.3406013ms	23.5038ms (35.43%)	-	+
/tags/p/2/	/imprint/	67.9293146ms	23.1398233ms (34.06%)	-	+
/	/	79.6650023ms	27.2197206ms (34.17%)	-	+
/imprint/	/imprint/	65.1834701ms	23.0573013ms (35.37%)	-	+
/tags/p/	/tags/p/	89.4712661ms	29.4440881ms (32.91%)	-	+

/tags/p/2/	/tags/p/2/	86.7669023ms	30.3473338ms (34.98%)	-	+
/	/tags/p/	101.7855178ms	43.100569ms (42.34%)	-	+
/imprint/	/tags/p/	84.2801923ms	42.5754955ms (50.52%)	-	+
/	/imprint/	70.6183847ms	25.6731125ms (36.35%)	-	+
/imprint/	/	69.7824729ms	25.9579276ms (37.20%)	-	+

Table 4: Selenium benchmark results on a local machine

From	To	Normal loadtime	Lare loadtime	TC	DBC
/tags/p/	/tags/p/2/	378.7378772ms	100.0062951ms (26.41%)	+	+
/tags/p/2/	/tags/p/	149.1767846ms	100.5256257ms (67.39%)	+	+
/tags/p/	/	177.8015667ms	101.0766767ms (56.86%)	+	+
/tags/p/2/	/	145.9762462ms	100.3458274ms (68.74%)	+	+
/tags/p/	/imprint/	145.1269058ms	99.0440198ms (68.25%)	+	+
/tags/p/2/	/imprint/	143.6989026ms	96.5441573ms (67.19%)	+	+
/	/	184.0143055ms	96.8061586ms (52.61%)	+	+
/imprint/	/imprint/	146.1839403ms	102.3730996ms (70.03%)	+	+
/tags/p/	/tags/p/	158.4086978ms	96.0358653ms (60.63%)	+	+
/tags/p/2/	/tags/p/2/	146.5099972ms	99.6919812ms (68.04%)	+	+
/	/tags/p/	172.8212049ms	104.4932832ms (60.46%)	+	+
/imprint/	/tags/p/	146.4761414ms	108.3417488ms (73.97%)	+	+
/	/imprint/	154.5478889ms	99.9289012ms (64.66%)	+	+
/imprint/	/	141.8673753ms	93.5422212ms (65.94%)	+	+
/tags/p/	/tags/p/2/	1759.8420863ms	165.503113ms (9.40%)	+	-
/tags/p/2/	/tags/p/	1565.4801458ms	153.9259203ms (9.83%)	+	-
/tags/p/	/	183.7747229ms	99.9362148ms (54.38%)	+	-
/tags/p/2/	/	142.2313467ms	113.300735ms (79.66%)	+	-
/tags/p/	/imprint/	143.6316948ms	104.1513079ms (72.51%)	+	-
/tags/p/2/	/imprint/	145.2610703ms	99.569909ms (68.55%)	+	-
/	/	174.5883816ms	95.7787123ms (54.86%)	+	-
/imprint/	/imprint/	145.7255202ms	95.7090602ms (65.68%)	+	-
/tags/p/	/tags/p/	1526.8233515ms	147.9105317ms (9.70%)	+	-
/tags/p/2/	/tags/p/2/	1542.0292612ms	148.6008572ms (9.64%)	+	-
/	/tags/p/	1587.1344271ms	1474.1047079ms (92.88%)	+	-
/imprint/	/tags/p/	1556.204205ms	1544.0254721ms (99.22%)	+	-
/	/imprint/	154.0036756ms	98.7195222ms (64.10%)	+	-
/imprint/	/	146.4833535ms	94.3328383ms (64.40%)	+	-
/tags/p/	/tags/p/2/	1668.684526ms	229.4092215ms (13.75%)	+	-
/tags/p/2/	/tags/p/	1616.4873064ms	214.1759213ms (13.25%)	+	-
/tags/p/	/	221.7372628ms	154.6111857ms (69.73%)	+	-
/tags/p/2/	/	196.4610228ms	161.2432577ms (82.07%)	+	-
/tags/p/	/imprint/	202.10608ms	144.7256252ms (71.61%)	+	-
/tags/p/2/	/imprint/	203.7251935ms	146.0406372ms (71.69%)	+	-
/	/	224.4458685ms	154.079448ms (68.65%)	+	-
/imprint/	/imprint/	200.6108772ms	153.1330621ms (76.33%)	+	-
/tags/p/	/tags/p/	1665.8789629ms	206.5612557ms (12.40%)	+	-
/tags/p/2/	/tags/p/2/	1576.9212358ms	222.5106568ms (14.11%)	+	-
/	/tags/p/	1689.5422182ms	1546.2859407ms (91.52%)	+	-
/imprint/	/tags/p/	1621.0944534ms	1573.9399932ms (97.09%)	+	-
/	/imprint/	211.8133351ms	153.5218091ms (72.48%)	+	-

/imprint/	/	197.1394442ms	149.1897551ms (75.68%)	+	-
/tags/p/	/tags/p/2/	233.9591823ms	152.9520268ms (65.38%)	+	-
/tags/p/2/	/tags/p/	220.6901163ms	159.6925743ms (72.36%)	+	-
/tags/p/	/	230.4939319ms	158.3108659ms (68.68%)	+	-
/tags/p/2/	/	216.9803952ms	158.1357452ms (72.88%)	+	-
/tags/p/	/imprint/	196.3278417ms	164.4535144ms (83.76%)	+	-
/tags/p/2/	/imprint/	222.7822235ms	150.0005896ms (67.33%)	+	-
/	/	236.6149375ms	156.5104787ms (66.15%)	+	-
/imprint/	/imprint/	195.0586462ms	149.5430947ms (76.67%)	+	-
/tags/p/	/tags/p/	221.6735031ms	155.7583656ms (70.26%)	+	-
/tags/p/2/	/tags/p/2/	215.5333872ms	161.8046043ms (75.07%)	+	-
/	/tags/p/	282.1626445ms	185.7336999ms (65.83%)	+	-
/imprint/	/tags/p/	222.8544114ms	168.8039173ms (75.75%)	+	-
/	/imprint/	207.511026ms	153.2500877ms (73.85%)	+	-
/imprint/	/	204.5646389ms	151.5592111ms (74.09%)	+	-

Table 5: Selenium benchmark results on a external machine

7 Conclusion and future work

We tested Lare to check if the browser functionality is working like on common web applications. The results show that those expectations regarding back and forward buttons in the browser are satisfied. Additionally the URL changes like on normal web applications which makes it possible to bookmark web pages.

The benchmark results show that the performance increases with Lare. The effect on static pages is always quite similar and fine.

When looking at the results for dynamic pages it is a bit more complex. The performance improvements by Lare vary a lot. Load times from 3% to 99% relatively to normal requests display this.

Besides other reasons the biggest difference between those low and high load times is the changed content. Requesting a very dynamic page with a not related namespace has nearly no improvements relatively to normal requests. Being on a very dynamic page, requesting a page with only a few changes makes it possible to experience a bigger benefit of Lare.

What does that mean for the usage of Lare, when is it a good idea to use it?

Static pages have a better performance with Lare than without. This makes it a good to use scenario.

When having dynamic content on a page and only a few elements change when requesting a new web page Lare strikes most.

web sites that are a collection of unrelated web pages are not taking a lot of benefit off Lare.

The more content stays from page to page, the more Lare scores.

Load tests with multiple users as the amount of concurrent users should be up to 10000 to be representative[13]. In the future it would be good to do such a performance benchmark, to see how Lare effects such amounts of users.

Additionally it is recommended to test every new Lare backend in a way similar to the one presented in this thesis. The results in different programming languages and web application designs may differ from the ones displayed here.

References

[1] David Jonsson (2009). *Database driven user friendly web application using Ajax* Master Thesis. Umeå University

[2] Peter Herlihy (2013). *How many people are missing out on JavaScript enhancement?* https://gds.blog.gov.uk/2013/10/21/how-many-people-are-missing-out-on-javascript-enhancement/

[3] Reto Matter *AJAX Crawl: Making AJAX Applications Searchable* Master Thesis. Swiss Federal Institute of Technology Zurich

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

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

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

[7] 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

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

[9] 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

[10] Jukka Palomäki *Web Application Performance Testing* Master Thesis. University of Turku (2009)

[11] Alessandro Marchetto, Paolo Tonella and Filippo Ricca *State-Based Testing of Ajax Web Applications* ICST '08 Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation

[12] Alessandro Marchetto, Paolo Tonella and Filippo Ricca *Testing Techniques applied to AJAX Web Applications* International Journal on Software Tools for Technology Transfer (STTT)

[13] Engin Bozdag, Ali Mesbah and Arie van Deursen *Performance Testing of Data Delivery Techniques for AJAX Applications* TUD-SERG Technical Report Series, Journal of Web Engineering

Glossary

AJAX Asynchronous JavaScript and XML [8]. i, ii, 1, 3–9, 11, 13, 19, 33

cURL Client for URLs (cURL) is an open source command line tool and library for transferring data with URL syntax.. i, ii, 2, 19, 21, 22, 24–26

django-lare The Lare backend for django. i, ii, 1, 18

HIJAX Hijax describes a technique to develop AJAX web applications that degrade gracefully.. ii, 9, 10

HTML HTML stands for Hyper Text Markup Language and is the language that is used in the Web.. ii, 3, 6

HTTP request Hypertext Transfer Protocol requests build the foundation for data communication in the World Wide Web.. i, ii, 3, 4, 7, 13, 19, 20, 22, 23, 25, 34

Lare Lightweight asynchronous replacement engine is a technology for stateful single-page applications. It consists of a Lare frontend as ajax engine, and a Lare backend which is plugged into the web application.. i, ii, 1–3, 6, 11–14, 16, 18–26, 29, 31, 33

lare.js lare.js is the Lare frontend and the AJAX engine for Lare.. i, ii, 1, 13, 18, 20

PHP-lare PHP-lare is the Lare backend for PHP.. i, ii, 1, 16, 18, 20, 33

single-page application A single-page application is a web application or web site that retrieves one full web page. Beside this first page load, the web site is not loaded completely at any point in the process anymore. Often content changes are made asynchronous and dynamically by AJAX in response to user actions.. i, ii, 1, 3, 6, 7, 9, 11, 12, 33

Twig Twig. 1

Twig-lare Twig-lare is the Lare backend for Twig and uses PHP-lare.. i, ii, 1, 17, 18, 20

URL A Uniform Resource Locator identifies and defines the location of a resource, e.g. a web page.. 1

[8] `https://developer.mozilla.org/de/docs/AJAX`

W3C The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards. Led by Web inventor Tim Berners-Lee and CEO Jeffrey Jaffe, W3C's mission is to lead the Web to its full potential.[9]. 6

web application A web application is a application that generates web pages.. i, 1, 19, 31

web page A web page is a single document and part of a web site. Every page should be accessible over the Internet and has it's own URL. A web browser can retrieve web pages by making HTTP requests and it can render them afterwards.. i, 1, 3, 5–9, 13, 20, 21, 31, 33, 34

web site A web site is a collection of web pages that are linked to each other.. i, 1, 6, 9, 31

[9] http://www.w3.org/Consortium/