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)

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 which is initially loaded is not changed until a new URL is requested by the user. One big change brought the invention of AJAX. It introduces 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 the performance of a new technology called Lare. Together with Stephan Groß I developed PJAXR, a technique 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 I will introduce a new Lare backend for PHP, *PHP-lare* and *Twig-lare* an extension for the Twig[2] template-engine.

[1]https://developers.google.com/webmasters/ajax-crawling/docs/getting-started
[2]http://twig.sensiolabs.org/

2 Fundamentals

To understand how Lare works a few technologies are required to know about. fundamentals introduction

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 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.

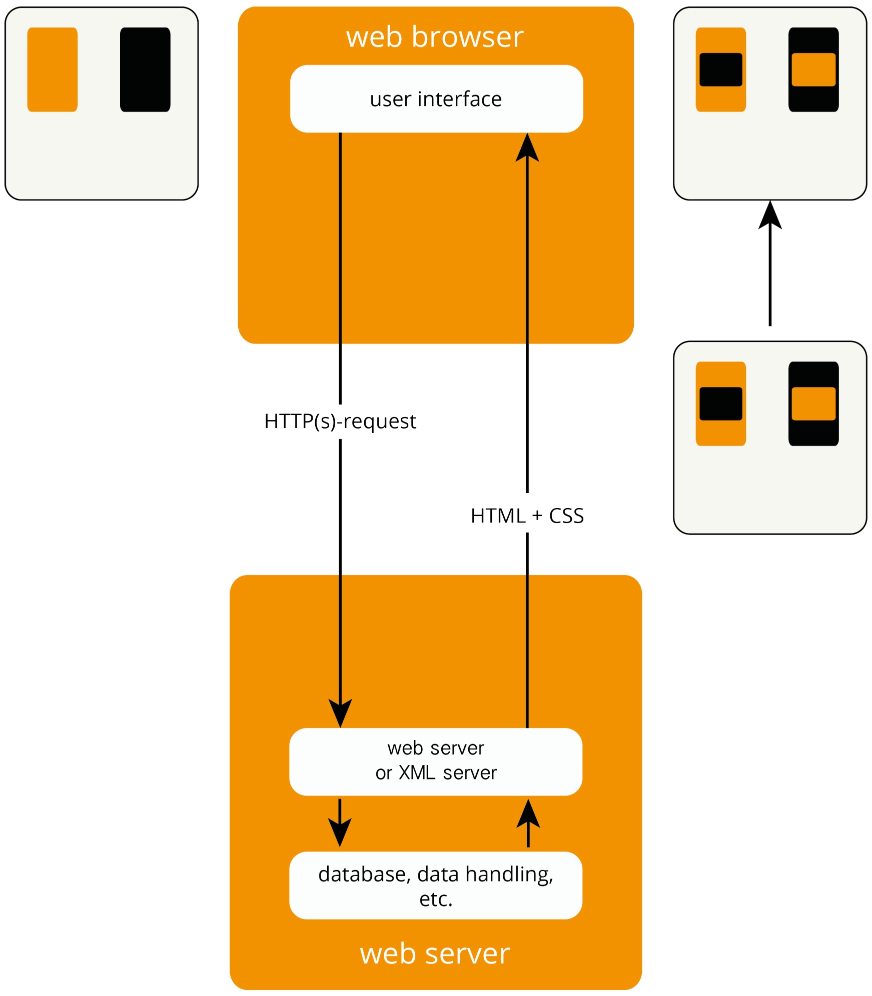

Figure 1: Component and communication diagram of HTTP

2.2 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 2.1, 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 2.5, is the most used technique for asynchronous web platforms.

2.3 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.

2.4 Single-page application

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.

2.5 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.

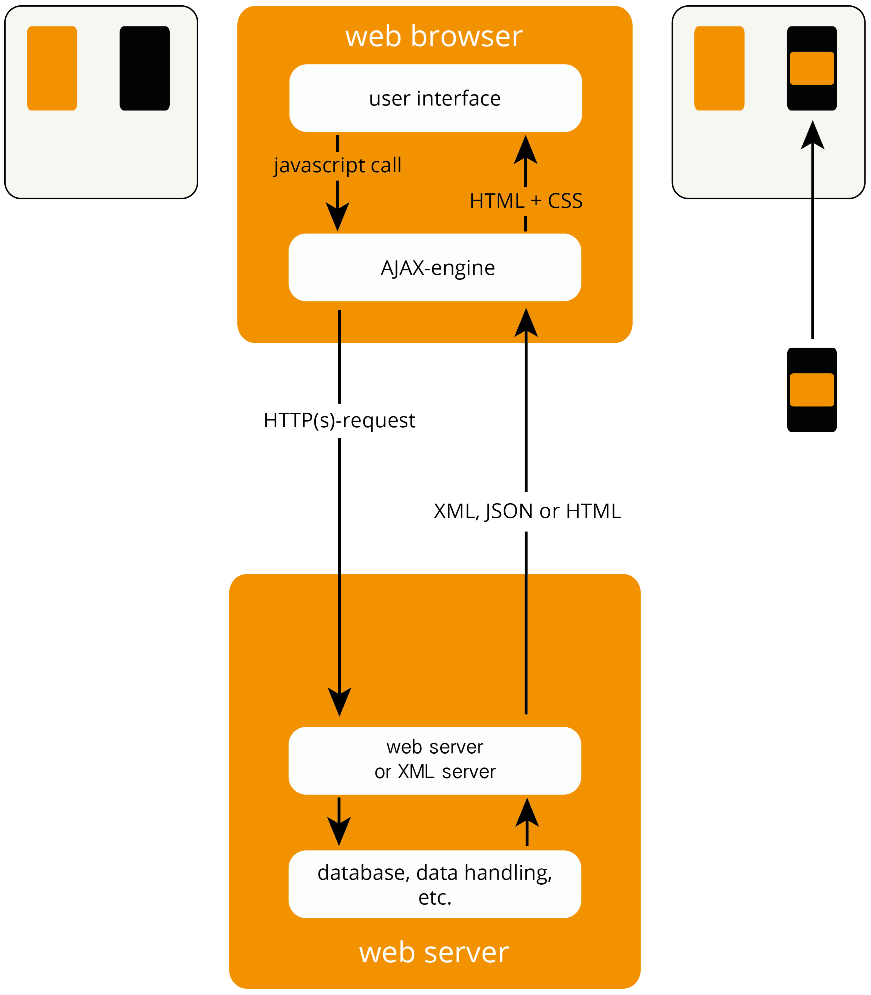

Figure 2: Component and communication diagram of AJAX

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, allowing 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.

2.6 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 [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.

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

explain hash bang URLs

3.4 HIJAX

One 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.

3.5 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 only has the opportunity to either request a full page, or request a per URL defined list of containers.

[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

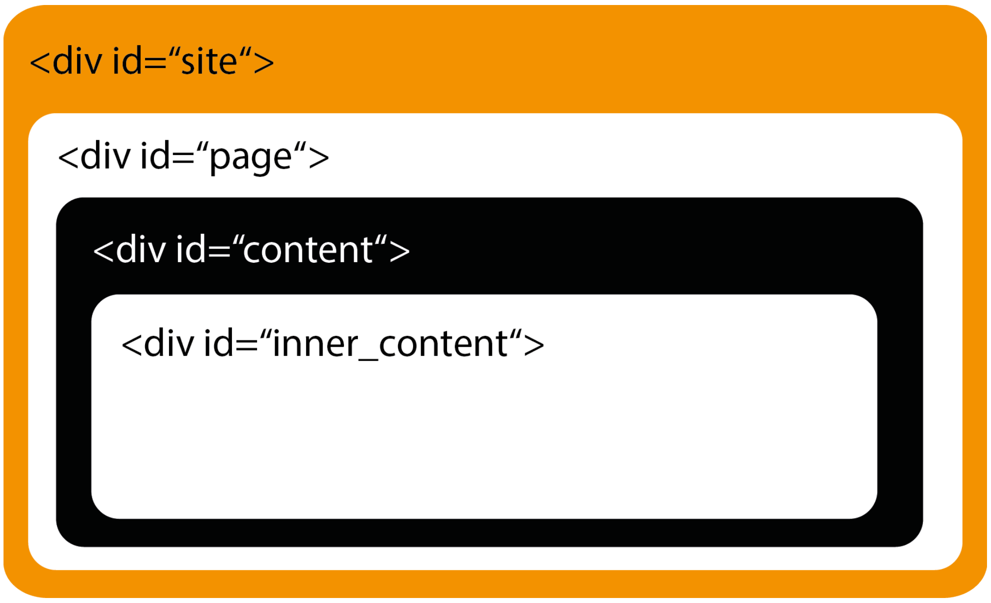

Figure 3: Basic HTML structure of a web application using Lare

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.

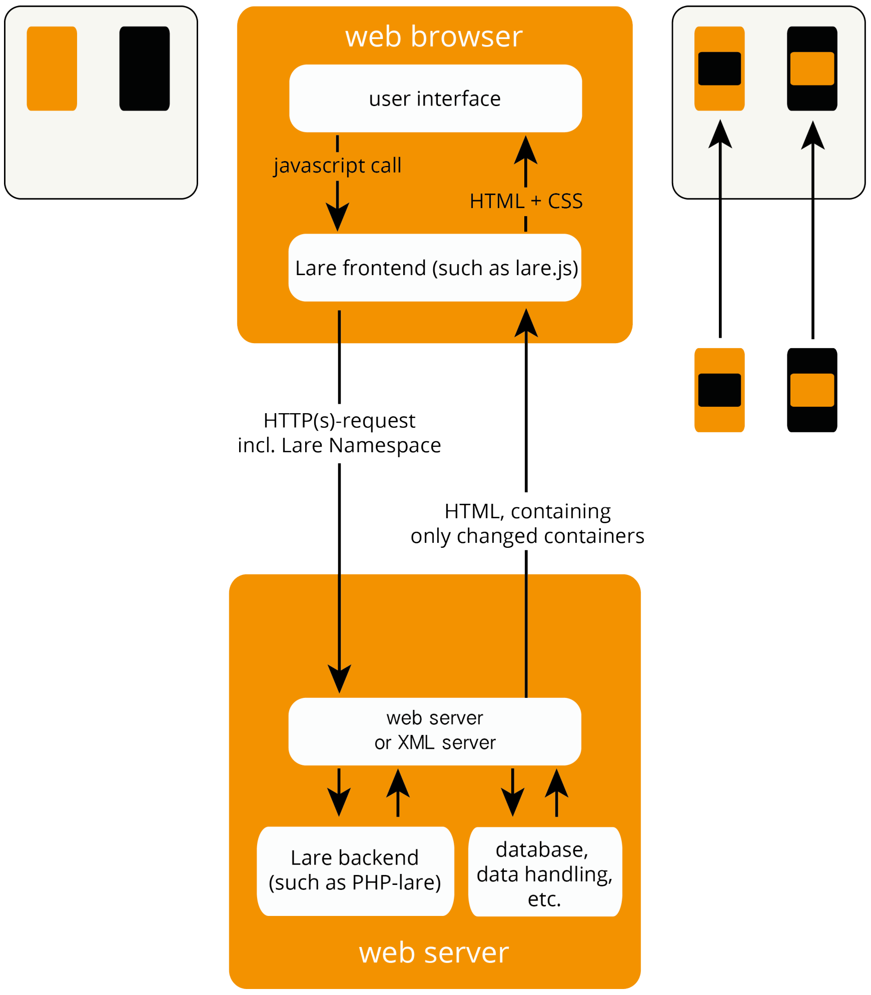

Figure 4: Component and communication diagram of Lare

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 We decided to implement the Lare backend in two parts. 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-na
```

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 <div> 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.

5.4 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

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 [6] there is a lack of good testing tools, especially when it comes to white-box testing. For black-box testing a good tool is selenium.

In [7] the same authors introduce a new automated testing technique, 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. [8]. 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 every dimension will be enabled and disabled to see whether it influences the results or not. As I use Mysql as database I will enable and disable the query caching. 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 not do load tests with multiple users as the amount of concurrent users should be up to 10000 to be representative[9]. In this thesis this we will not be able to do that, due to the lack of a Supercomputer such as used in [10].

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,

[6]http://selab.fbk.eu/tonella/papers/wqvv2007.pdf

[7]http://selab.fbk.eu/tonella/papers/icst2008.pdf

[8]http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2008-009.pdf

[9]http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2008-009.pdf

[10]http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2008-009.pdf

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. We use the MVC design pattern, but with a slightly different naming. The Models are called Classes.

6.2 test layout

- Initial-Requests:
  - Static:
    - /
    - /imprint/
  - Dynamic:
    - /tags/p/
    - /tags/p/2/

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

6.3 Selenium

describe selenium tests

6.4 Curl

describe curl test

6.5 Chrome Network Tools

describe chrome network tools

6.6 Results

6.7 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

Glossary

AJAX Asynchronous JavaScript and XML [11]. i, 1, 3–5, 7–10, 12, 18, 22, 23

django-lare The Lare backend for django. i, 1, 17

HIJAX Hijax. i, 9

HTML HTML stands for Hyper Text Markup Language and is the language that is used in the Web.. i, 2, 4, 5

HTTP request Hypertext Transfer Protocol requests build the foundation for data communication in the World Wide Web.. i, 2, 3, 7, 12, 18, 23

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, 1, 2, 6, 10–13, 15, 17–19, 22, 23

lare.js lare.js is the Lare frontend and the AJAX engine for Lare.. i, 1, 12, 17

PHP-lare PHP-lare is the Lare backend for PHP.. i, 1, 15, 17, 23

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

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, 1, 4, 7, 9–11, 22

Twig Twig. 1

Twig-lare Twig-lare is the Lare backend for Twig and uses PHP-lare.. i, 1, 16, 17

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

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.[12]. 6

web application A web application is a application that generates web pages.. 18

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.. 1, 2, 4, 7–9, 12, 23

web site A web site is a collection of web pages that are linked to each other.. 4

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