
HTTP is your architecture

International PHP Conference Spring Edition

Kore Nordmann (@koredn)
Tobias Schlitt (@tobySen)

May 31, 2011

About us

- ▶ Degree in computer science



About us

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP



About us

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiasts
- ▶ Contributing to various FLOSS projects



About us

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiasts
- ▶ Contributing to various FLOSS projects

Co-founders of



About us

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiasts
- ▶ Contributing to various FLOSS projects

Co-founders of



**We help people to create
high quality PHP
applications.**

About us

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiasts
- ▶ Contributing to various FLOSS projects

Co-founders of



**We help people to create
high quality PHP
applications.**

<http://qafoo.com>

Outline

Introduction

HTTP

Taking it further

Conclusion

Aspects of a web application

- ▶ Scalability
- ▶ Reliability
- ▶ Simplicity
- ▶ Usability
- ▶ Security
- ▶ Standard Compliance

LCoDC\$SS

LCoDC\$SS

- ▶ Who heard of this term before?

LCoDC\$SS

- ▶ Who heard of this term before?
 - ▶ This **is** HTTP. [Fie00]

LCoDC\$SS

Layered CoDC\$SS

Layered Code on Demand C\$SS

Layered Code on Demand Client \$S Server

Layered Code on Demand Client Cached S Server

Layered Code on Demand Client Cached Stateless Server

Outline

Introduction

HTTP

Taking it further

Conclusion

Outline

HTTP

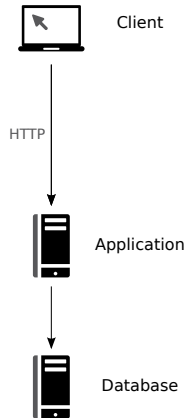
- Layered architecture

- Request semantics

- Stateless server

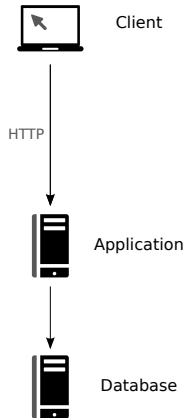
- Code on demand

Layered architecture



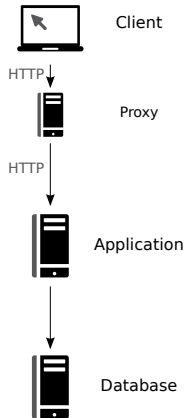
Layered architecture

- ▶ What is required?



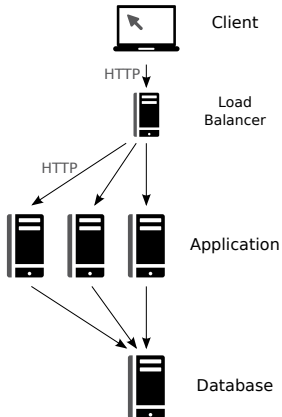
Layered architecture

- ▶ What is required?
 - ▶ Request semantic



Layered architecture

- ▶ What is required?
 - ▶ Request semantic
 - ▶ Stateless server



Outline

HTTP

Layered architecture

Request semantics

Stateless server

Code on demand

HTTP methods

- ▶ Well known
 - ▶ GET
 - ▶ POST

HTTP methods

- ▶ Well known
 - ▶ GET
 - ▶ POST
- ▶ Less known
 - ▶ PUT
 - ▶ DELETE

HTTP methods

- ▶ Well known
 - ▶ GET
 - ▶ POST
- ▶ Less known
 - ▶ PUT
 - ▶ DELETE

- ▶ Mostly unknown
 - ▶ HEAD
 - ▶ OPTIONS
 - ▶ TRACE
 - ▶ CONNECT

HTTP methods

- ▶ Well known
 - ▶ GET
 - ▶ POST
- ▶ Less known
 - ▶ PUT
 - ▶ DELETE

- ▶ Mostly unknown
 - ▶ HEAD
 - ▶ OPTIONS
 - ▶ TRACE
 - ▶ CONNECT
- ▶ WebDAV
 - ▶ MKCOL
 - ▶ PROPSET
 - ▶ PROPGET

HTTP methods

- ▶ Well known
 - ▶ GET
 - ▶ POST
- ▶ Less known
 - ▶ PUT
 - ▶ DELETE
- ▶ Mostly unknown
 - ▶ HEAD
 - ▶ OPTIONS
 - ▶ TRACE
 - ▶ CONNECT
- ▶ WebDAV
 - ▶ MKCOL
 - ▶ PROPSET
 - ▶ PROPGET
- ▶ Use any you want...

“Safe” HTTP methods

[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. [RF99]

“Safe” HTTP methods

[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. [RF99]

- ▶ ... so it is *safe* for spiders to call them.

“Safe” HTTP methods

[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. [RF99]

- ▶ ... so it is *safe* for spiders to call them.
- ▶ Since nothing is modified, the result can be cached.
 - ▶ Proxies can use that automatically
 - ▶ Varnish / Squid
 - ▶ Company application proxies

“Safe” HTTP methods

[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. [RF99]

- ▶ ... so it is *safe* for spiders to call them.
- ▶ Since nothing is modified, the result can be cached.
 - ▶ Proxies can use that automatically
 - ▶ Varnish / Squid
 - ▶ Company application proxies
- ▶ Layered Code on Demand Client **Cached** Stateless Server

Is your search form broken?

```
1 <form action="/search" method="POST">
2   <input type="text" name="term" />
3   <input type="submit" value="Search!" />
4 </form>
```

Is your search form broken?

```
1 <form action="/search" method="POST">
2   <input type="text" name="term" />
3   <input type="submit" value="Search!" />
4 </form>
```

- ▶ Broken semantics
 - ▶ Search results may not be cached

Is your search form broken?

```
1 <form action="/search" method="POST">
2   <input type="text" name="term" />
3   <input type="submit" value="Search!" />
4 </form>
```

- ▶ Broken semantics
 - ▶ Search results may not be cached
- ▶ No bookmarking of search results

POST vs. PUT

POST vs. PUT

- ▶ POST

- ▶ PUT

POST vs. PUT

- ▶ POST

- ▶ *... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]*

- ▶ PUT

POST vs. PUT

- ▶ POST

- ▶ *... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]*
- ▶ Appends to an existing resource!
 - ▶ e.g. an existing collection of documents

- ▶ PUT

POST vs. PUT

- ▶ POST

- ▶ ... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]
- ▶ Appends to an existing resource!
 - ▶ e.g. an existing collection of documents
- ▶ Examples
 - ▶ Posting a message to a bulletin board
 - ▶ Extending a database through an append operation

- ▶ PUT

POST vs. PUT

▶ POST

- ▶ *... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]*
- ▶ Appends to an existing resource!
 - ▶ e.g. an existing collection of documents
- ▶ Examples
 - ▶ Posting a message to a bulletin board
 - ▶ Extending a database through an append operation

▶ PUT

- ▶ *... requests that the enclosed entity be stored under the supplied Request-URI. [RF99]*

POST vs. PUT

▶ POST

- ▶ *... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]*
- ▶ Appends to an existing resource!
 - ▶ e.g. an existing collection of documents
- ▶ Examples
 - ▶ Posting a message to a bulletin board
 - ▶ Extending a database through an append operation

▶ PUT

- ▶ *... requests that the enclosed entity be stored under the supplied Request-URI. [RF99]*
- ▶ Creates or replaces a resource!

POST vs. PUT

▶ POST

- ▶ *... is used to request that the origin server accept the entity enclosed in the request as a **new subordinate** ... [RF99]*
- ▶ Appends to an existing resource!
 - ▶ e.g. an existing collection of documents
- ▶ Examples
 - ▶ Posting a message to a bulletin board
 - ▶ Extending a database through an append operation

▶ PUT

- ▶ *... requests that the enclosed entity be stored under the supplied Request-URI. [RF99]*
- ▶ Creates or replaces a resource!
- ▶ Examples
 - ▶ Updating account data using (PUT /users/42)
 - ▶ Create a new resource with known identifier

HTML 5 is broken!

Using PUT and DELETE as HTTP methods for the form element is no longer supported. [vK10]

Idempotent methods

$$f(x) = f(f(x))$$

Idempotent methods

$$f(x) = f(f(x))$$

- ▶ Everything but POST has to be idempotent
 - ▶ Executing the request again, should not change anything.

Idempotent methods

$$f(x) = f(f(x))$$

- ▶ Everything but POST has to be idempotent
 - ▶ Executing the request again, should not change anything.
 - ▶ This includes PUT and DELETE

Idempotent methods

$$f(x) = f(f(x))$$

- ▶ Everything but POST has to be idempotent
 - ▶ Executing the request again, should not change anything.
 - ▶ This includes PUT and DELETE
- ▶ Really useful to just resend request, if one failed due to network problems

Idempotent methods

$$f(x) = f(f(x))$$

- ▶ Everything but POST has to be idempotent
 - ▶ Executing the request again, should not change anything.
 - ▶ This includes PUT and DELETE
- ▶ Really useful to just resend request, if one failed due to network problems
- ▶ Idempotence is a useful property in all messaging systems

HTTP method fail in PHP

- ▶ `$_GET` contains the request parameters
- ▶ `$_POST` contains the request body

HTTP method fail in PHP

- ▶ `$_GET` contains the request parameters
- ▶ `$_POST` contains the request body
- ▶ All HTTP requests may contain body **and** parameters

HTTP method fail in PHP

- ▶ `$_GET` contains the request parameters
- ▶ `$_POST` contains the request body
- ▶ All HTTP requests may contain body **and** parameters
 - ▶ Yes, even GET!

HTTP method fail in PHP

- ▶ `$_GET` contains the request parameters
- ▶ `$_POST` contains the request body
- ▶ All HTTP requests may contain body **and** parameters
 - ▶ Yes, even GET!
- ▶ You may want to use something like
 - ▶ `$request->parameters`
 - ▶ `$request->body`

Do you speak HTTP?

- ▶ GET and HEAD **must** be supported

Do you speak HTTP?

- ▶ GET and HEAD **must** be supported
- ▶ All other methods are **optional**

Do you speak HTTP?

- ▶ GET and HEAD **must** be supported
- ▶ All other methods are **optional**
- ▶ But if you implement them, they must obey to semantics

Do you speak HTTP?

- ▶ GET and HEAD **must** be supported
- ▶ All other methods are **optional**
- ▶ But if you implement them, they must obey to semantics
- ▶ Sorry, your website is not HTTP, if you ...
 - ▶ ...are using POST for a search form.
 - ▶ ...are using POST for data updates.

Outline

HTTP

Layered architecture

Request semantics

Stateless server

Code on demand

Stateless server

- ▶ No persistent connection
- ▶ Each request contains all information to be processed

Stateless server

- ▶ No persistent connection
- ▶ Each request contains all information to be processed
 - ▶ Cookies

Stateless server

- ▶ No persistent connection
- ▶ Each request contains all information to be processed
 - ▶ Cookies
- ▶ Servers can be exchanged transparently

Stateless server

- ▶ No persistent connection
- ▶ Each request contains all information to be processed
 - ▶ Cookies
- ▶ Servers can be exchanged transparently
 - ▶ Mind the sessions and static data

Stateless server

- ▶ No persistent connection
- ▶ Each request contains all information to be processed
 - ▶ Cookies
- ▶ Servers can be exchanged transparently
 - ▶ Mind the sessions and static data
- ▶ Layered Code on Demand Client Cached Stateless Server

Drawbacks / Benefits

- ▶ Drawbacks
- ▶ Benefits

Drawbacks / Benefits

- ▶ Drawbacks
 - ▶ Users **do** have state
- ▶ Benefits

Drawbacks / Benefits

- ▶ Drawbacks
 - ▶ Users **do** have state
- ▶ Benefits
 - ▶ Scalability

Drawbacks / Benefits

- ▶ Drawbacks
 - ▶ Users **do** have state
- ▶ Benefits
 - ▶ Scalability
 - ▶ Failover

Drawbacks / Benefits

- ▶ Drawbacks
 - ▶ Users **do** have state
- ▶ Benefits
 - ▶ Scalability
 - ▶ Failover
 - ▶ Simplicity

Share nothing!

- ▶ PHP itself follows a *share nothing* architecture
 - ▶ Clean PHP instance for each request
 - ▶ By default no shared resources

- ▶ App servers usually suck!

Share nothing!

- ▶ PHP itself follows a *share nothing* architecture
 - ▶ Clean PHP instance for each request
 - ▶ By default no shared resources
- ▶ It's your job to obey to it!

- ▶ App servers usually suck!

Share nothing!

- ▶ PHP itself follows a *share nothing* architecture
 - ▶ Clean PHP instance for each request
 - ▶ By default no shared resources
- ▶ It's your job to obey to it!
 - ▶ Session storage?

- ▶ App servers usually suck!

Share nothing!

- ▶ PHP itself follows a *share nothing* architecture
 - ▶ Clean PHP instance for each request
 - ▶ By default no shared resources
- ▶ It's your job to obey to it!
 - ▶ Session storage?
 - ▶ Database server?

- ▶ App servers usually suck!

Share nothing!

- ▶ PHP itself follows a *share nothing* architecture
 - ▶ Clean PHP instance for each request
 - ▶ By default no shared resources
- ▶ It's your job to obey to it!
 - ▶ Session storage?
 - ▶ Database server?
 - ▶ File access?
- ▶ App servers usually suck!

Outline

HTTP

Layered architecture

Request semantics

Stateless server

Code on demand

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

- ▶ Do you deliver code on demand?

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

- ▶ Do you deliver code on demand?
- ▶ You do!

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

- ▶ Do you deliver code on demand?
- ▶ You do!
 - ▶ JavaScript

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

- ▶ Do you deliver code on demand?
- ▶ You do!
 - ▶ JavaScript
 - ▶ HTML is also just code
although not turing complete

Do you?

In the code-on-demand style, a client component . . . receives that code, and executes it locally. [Fie00]

- ▶ Do you deliver code on demand?
- ▶ You do!
 - ▶ JavaScript
 - ▶ HTML is also just code
 - although not turing complete
- ▶ Layered **Code on Demand Client** Cached Stateless Server

Code generation is the root cause for webb application security problems.

Code generation is the root cause for webb application security problems.

- ▶ Escape for target context / language (XSS)

Outline

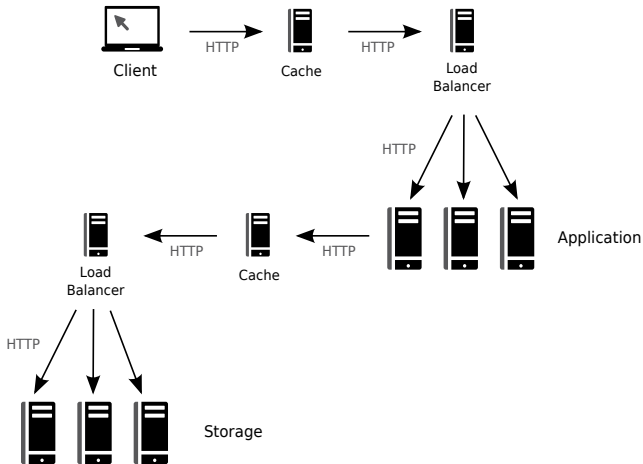
Introduction

HTTP

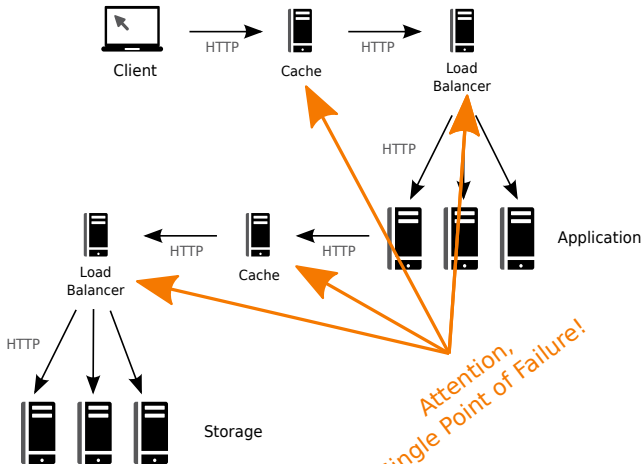
Taking it further

Conclusion

Embrace HTTP



Embrace HTTP



Use HTTP actively

- ▶ Semantic HTTP methods
- ▶ URIs (address resources)
- ▶ Status codes
- ▶ Headers
 - ▶ Cache control
 - ▶ Content negotiation
 - ▶ ...

What about REST?

- ▶ What is REST actually?

What about REST?

- ▶ What is REST actually?
 - ▶ “New” style web services

What about REST?

- ▶ What is REST actually?
 - ▶ “New” style web services
 - ▶ Follow HTTP / LCoDC\$SS
 - ▶ Follow resources / concept character of URIs
 - ▶ Use proper status codes

What about REST?

- ▶ What is REST actually?
 - ▶ “New” style web services
 - ▶ Follow HTTP / LCoDC\$SS
 - ▶ Follow resources / concept character of URIs
 - ▶ Use proper status codes
 - ▶ **HATEOAS**

What about REST?

- ▶ What is REST actually?
 - ▶ “New” style web services
 - ▶ Follow HTTP / LCoDC\$SS
 - ▶ Follow resources / concept character of URIs
 - ▶ Use proper status codes
 - ▶ **HATEOAS**
 - ▶ Hypermedia as the Engine of Application State [Fie00]

What about REST?

- ▶ What is REST actually?
 - ▶ “New” style web services
 - ▶ Follow HTTP / LCoDC\$SS
 - ▶ Follow resources / concept character of URIs
 - ▶ Use proper status codes
 - ▶ **HATEOAS**
 - ▶ Hypermedia as the Engine of Application State [Fie00]
 - ▶ Short: Use hyper links to delegate client

Outline

Introduction

HTTP

Taking it further

Conclusion

Conclusion

Use HTTP!

Conclusion

Use HTTP, **properly!**

Questions? Comments? Feedback?

Thanks for listening

Please rate this talk at
<http://joind.in/3502>
and (optionally) give us some feedback right now

Thanks for listening

Please rate this talk at
<http://joind.in/3502>
and (optionally) give us some feedback right now

This is very important for ...

- ▶ Speakers
- ▶ Organizers
- ▶ You!

Thanks for listening

Please rate this talk at
<http://joind.in/3502>
and (optionally) give us some feedback right now

Stay in touch

- ▶ Kore Nordmann
- ▶ kore@qafoo.com
- ▶ @koredn / @qafoo
- ▶ Tobias Schlitt
- ▶ toby@qafoo.com
- ▶ @tobySen / @qafoo

Rent a PHP quality expert:

<http://qafoo.com>

Bibliography I

- [Fie00] R. Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, University of California, Irvine, USA, 2000.
- [RF99] et al. R. Fielding, *Hypertext transfer protocol – http/1.1*, <http://tools.ietf.org/html/rfc2616>, June 1999.
- [vK10] Anne van Kesteren, *Html5 differences from html4*, <http://www.w3.org/TR/2010/WD-html5-diff-20101019/>, October 2010.