Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
WEB

Web Architectures

Version 2020.1-EN; April 20, 2020

Table of Contents

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

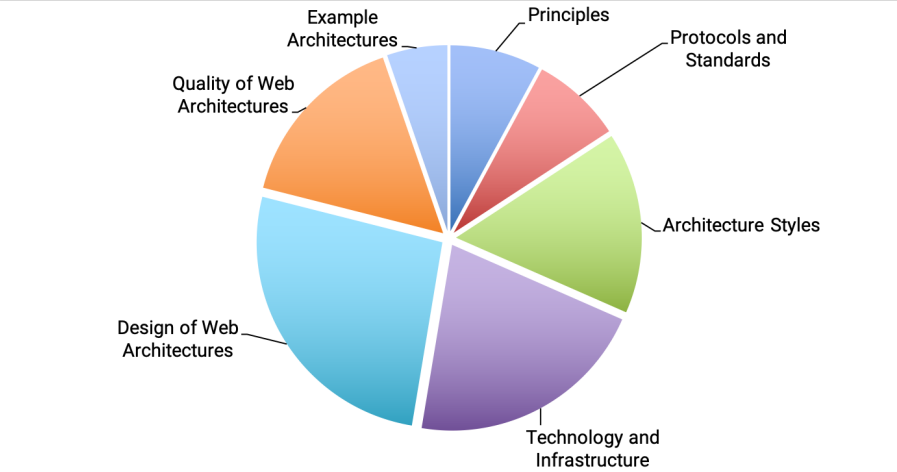
- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
  - existing certifications (for example: Sun/Oracle Java architect, Microsoft CSA) can be credited upon application
- Successful completion of the CPSA-A certification exam

Essentials

Curriculum Structure and Recommended Durations

Content	Recommended minimum duration (minutes)
1. Fundamentals	90
2. Protocols and Standards	90
3. Architecture Styles	180
4. Technology and Infrastructure	240
5. Design of Web Architectures	300
6. Quality in Web Architectures	180
7. Example Architectures	60
Total	1140 (19h)

Allocation of time for the topic areas



Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the WEB module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the WEB module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	0 Points
Technical Competence:	30 Points
Communicative Competence:	0 Points

Requirements

Participants **should** have the following knowledge and/or experience:

- CPSA-F and all associated requirements
- Experience with distributed systems—ideally web apps
- Basic knowledge of web technologies HTML, CSS, JavaScript, and at least one server-side framework

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Fundamentals

Duration: 90 min	Practice time: none

1.1. Terms and principles

Web browser, web server, client, server, proxy, request, response, accessibility, basic-auth, intranet vs. Internet, redirect, TLS, prepocessor, hypermedia, statelessness, pattern libraries, frontend components

1.2. Learning Goals

In this chapter, technology-independent concepts from the Web environment are discussed.

What should participants be able to do?

- Participants can explain the typical request/response process that takes place when an address is entered in the address bar of the browser or when a form is submitted.
- The participants can map the request/response process onto typical components in the web environment: client, server, proxy, reverse proxy, load balancer, DNS server, framework, own application logic (servlet, PHP script, Rails controller).
- Participants can explain the difference between GET and POST requests.
- Participants can draw a diagram of the typical client-server infrastructure of web architectures.
- Participants can explain the general structure of HTTP requests: header (with hostname, content type, etc.), content.
- Participants can explain the general structure of HTTP responses: response with status line, header and content.
- Participants can explain the structure of a URI and interpret the components in the request process that tend to be responsible for it.

What should participants understand?

- The participants understand that the generic term web application can refer to fundamentally different types of IT systems with regard to both technology and content, which require different architectures
- The participants understand that web applications are not restricted to use by browsers, but can also be used by other clients
- The participants understand the difference between the content-related or structural and presentational characterization of content in terms of the separation of concerns.
- The participants understand the general requirements of the Web Content Accessibility Guidelines (WCAG), which are summarized under the term accessibility.
- The participants understand the flow usually used for web applications with single sign-on mechanisms (redirect, login, redirect with token, validation of the token).
- The participants understand that a resource is more than just a database entity (i.e. also a process instance or a process step).
- The participants understand that hypermedia is used to connect resources to each other.
- The participants understand the principle of preprocessors for translating one language into another

or for embedding dependencies or detecting dead blocks of code.

What should participants know?

- The participants know that legal frameworks may require accessible interfaces. For example, the ‘Ordinance on the Creation of Barrier-Free Information Technology in Accordance with the Act on Equal Opportunities for Disabled Persons’ formulates specific requirements that must be implemented by all websites of the Federal Administration of the Federal Republic of Germany.
- The participants know some of the techniques that lead to accessible content, while, at the same time, ensuring that this content can also be processed more easily by machine.
- The participants know different characteristics of web frameworks:
  - Component and request-response frameworks often use designs influenced by the MVC pattern
  - Request-response frameworks generally have a relatively similar structure: actions accept the request, execute application logic, and finally render the response, usually with templating (HTML) or object serialization (JSON/XML) *The participants know the principle of pattern libraries as a way of documenting frontend components.

2. Protocols and Standards

Duration: 90 min	Practice time: none

2.1. Terms and Principles

URI, URL, URN, HTTP/1.1, HTTP/2, HTTP/3, HTTP verbs (GET, PUT, POST, DELETE), HTTP headers, intermediaries, caching, content types, content negotiation, TLS, PKI, HTML, DOM, web sockets, server-sent events, custom elements, shadow DOM, OAuth 2, OpenID Connect, CORS, Content Security Policy

2.2. Learning Goals

What should participants be able to do?

- The participants can design software systems in such a way that they use the protocols commonly used on the World Wide Web in an effective and resource-friendly manner at the interfaces.
- The participants can selectively use HTTP headers to enable the infrastructure to perform caching.
- The participants can state which HTTP headers refer to caching and how they affect it (keyword validation vs. expiration timestamp).
- The participants can explain the difference between the roles that HTML, CSS, and JavaScript play in the browser.

What should participants understand?

- The participants understand that the protocols and architecture of the web are technology independent.
- The participants understand the relationship between server address and server name and know the effects of creating and using URLs that should be used both inside and outside the system.
- The participants understand the relationship between the transport protocol (TCP, UDP) and the application protocol (HTTP/1.1, HTTP/2, and HTTP/3 & QUIC).
- The participants understand how a name resolution via DNS lookup works.
- The participants understand the effects of cache control headers on intermediaries.
- The participants understand the essential properties of the HTTP protocol and can explain them.
  - The HTTP protocol is a stateless request-response application protocol.
  - HTTP /1.1 is a text-based serial application protocol
  - HTTP/2 multiplexes many requests/responses simultaneously over a TCP connection.
  - HTTP/3 exchanges TCP for UDP and complements the missing reliable connection in the form of the QUIC on the application protocol level.
  - The participants know that HTTP/3, HTTP/2, and HTTP/1.1 implement largely the same semantics based on different line protocols.
  - The HTTP protocol explicitly provides for intermediate processing processes (intermediaries).
  - The client identifies a resource to the server using a URI with the schema http or https. HTTP verbs determine the type of access; the verbs have semantics.

    - The server responds with standardized status codes.
    - HTTP headers are used for additional services, metadata, or extensions.
    - Various HTTP headers jointly determine whether responses are allowed to be cached.
    - Different representations of the same resource are identified via media types (content types) and can be negotiated between the client and server via content negotiation.
    - Compression of the response data can (and should) also be negotiated via an analogous mechanism.
    - The HTTP protocol offers scope for different processes for authenticating the client to the server (e.g., basic authentication).
    - Cookies extend the protocol with a mechanism for annotating the browser session with any information.
    - Based on cookies, the inherently stateless protocol can be used with an assigned status via sessions on the server side.
    - Participants understand the internal structure of HTTP requests and responses.
    - The participants understand that the TLS protocol encrypts at the transport level, where symmetric encryption takes place.
    - The key is negotiated asymmetrically, e.g., with certificates for the server and the client.
    - Client certificates can be used for authentication.
    - Certificates are managed using public key infrastructures (PKI).
- The participants understand that there are different authentication mechanisms for different requirements.
- The participants understand that the structure of the information should be separated from its representation.
- The participants understand how a redirect is processed.
- The participants understand the term idempotence.
- The participants understand the property of safety. They know that an HTTP GET is safe and that the server response to a GET can therefore fundamentally be cached.
- The participants understand why a browser requires a separate confirmation when “reloading” with HTTP POST.
- The participants understand how web applications must be implemented to largely prevent such browser queries.
- The participants understand what HTTPS means and how it works.
- The participants understand the effect of terminating a TLS connection.
- The participants know that username and password are transmitted in plain text with HTTP basic authentication.
- Participants know that confidential information in the web environment should ideally be transferred in an encrypted form during transport. Transport level security (TLS) is used for encryption while in transit.
- The participants understand the effects of TLS on caching.

What should participants know?

- The participants know the common HTTP status codes and know what cause is assumed and what reaction to it is usually expected.
- The participants understand the responsibilities and tasks of the protocols and components in the web environment:
  - URIs identify resources, URLs additionally localize them at an authority,
  - DNS servers help resolve the authority part of the URI,
  - The HTTP protocol is a generic protocol for accessing resources and provides solutions for several non-technical requirements,
  - Standardized formats are available for different types of data,
  - HTTP allows clients and servers to negotiate the format used if there are several alternatives.
- The participants know the Document Object Model used internally in browsers and know that it can be modified by JavaScript and CSS.
- The participants know different data formats for the representation of information (HTML, XML, JSON, …)
- The participants know how cookies are exchanged and managed between the client and server.
- The participants know the XMLHttpRequest object (XHR for short) as well as the HTML5-fetch API as the basis of AJAX-based applications.
- The participants know the possibilities that result from server-side events.
- The participants know mechanisms that allow the use of the Back and Forward browser buttons without triggering unwanted side effects.
- The participants know the relevant standardization bodies such as IETF, IANA, W3C and their areas of responsibility with regard to web architecture.
- The participants know WebSockets.
- The participants know OAuth 2 as a system for delegating access to third parties.
- The participants know OpenID Connect as an OAuth 2-based system for single sign-on
- The participants know CORS as a mechanism for handling cross-domain restrictions
- The participants know the Content Security Policy as a mechanism to hamper cross-site scripting

2.3. References

[ECMA-262], [Fielding+2000], [HTML-CSS], [Jacobs+2004], [RFC2246], [RFC2616], [RFC3986], [RFC3987], [RFC6265], [Tilkov 2011], [WS-I]

3. Architecture Styles

Duration: 120 min	Practice time: 60 min

3.1. Terms and Principles

Representational state transfer (REST), stateful backend web apps, single-page application, web components

3.2. Learning Goals

What should participants be able to do?

- The participants can explain how the REST style differs from the stateful backend architecture style.
- The participants can distinguish between different web architecture styles and deliberately choose the one that makes the most sense for a set of given requirements and conditions.
- The participants can evaluate a framework, which is also potentially unknown to them, with regard to how well-suited it is for implementing web applications according to the requirements.
- The participants can explain different architecture styles of web architectures and design corresponding systems:
  - REST-compliant web applications
  - Stateful backend applications (for example, component-oriented approaches).

What should participants understand?

- The participants understand the constraints that the REST architecture style places on the design of a system:
  - Identifiable resources
  - Uniform interface
  - Stateless communication
  - Representations
  - Hypermedia.
- The participants understand the constraints imposed by stateful backend architectures:
  - Communication always takes place against the same server instance.
  - With component-oriented approaches, the requests usually contain a command or a discriminator that decides which component is responsible for processing. This dispatch information is usually hard-coded or stored in the main memory (in the session).
- Participants understand that, regardless of the architecture style chosen, the core of the business, such as security and function-relevant tests or calculations, must always be implemented in the server.
- The participants understand how the responsiveness of web applications can be improved via functionality in the client, i.e. via JavaScript in the client (e.g., in conjunction with AJAX).
- The participants understand the standard features of single-page frameworks:

    - Data binding (two-way and one-way)
    - Templating
    - Client-side routing
    - Component abstraction
- The participants understand that web components (custom elements + shadow DOM) can be used to develop independent JavaScript components that are strongly isolated from the page and can be instantiated via DOM.

What should participants know?

- The participants know mechanisms for accepting long-running transactions with corresponding status codes and to report the result using suitable mechanisms as soon as it is available.
- The participants know the effects of mobile clients:
    - Greatly varying bandwidths
    - Fluctuating, partly very high latency.
- The participants know that mobile clients sometimes have reduced performance in terms of main memory and CPU.
- The participants know that resources of mobile clients have to be conserved, especially with regard to the limited energy capacities of devices.

4. Technology and infrastructure

Duration: 180 min	Practice time: 60 min

4.1. Terms and Principles

Client, server, proxy, reverse proxy, content delivery networks/CDN, load balancing, CGI

4.2. Learning Goals

What should participants be able to do?

- The participants can design server-side web applications so that they use the infrastructure effectively.
- The participants can improve the runtime behavior of a system through the targeted use of reverse proxies.
- The participants can name various intermediaries and explain their effects on the architecture.
  - Proxies store responses for the client; the HTTP protocol provides explicit rules for this.
  - Reverse proxies are proxies on the server side and act as caches for the web application.
  - Load balancers distribute queries on the server side to different servers.
- The participants know different scaling strategies (horizontal and vertical) and can select the appropriate scaling strategy.
  - Vertical scaling due to more powerful hardware.
  - Vertical scaling due to different components – such as web and application servers – being distributed on separate infrastructure components.
  - Vertical scaling due to cluster solutions that simulate an application running on only one node even though there are multiple server nodes to run it.
  - Horizontal scaling by duplicating a server system and distributing the load as randomly and evenly as possible over the existing servers. This procedure must be potentially infinite for horizontal scaling.

What should participants understand?

- The participants understand that caching can reduce the load on server systems.
- The participants understand how data from many web applications can be made available to a few users (client or forward proxy) or many users for a few web applications (reverse proxy).
- The participants understand how access to resources can be controlled by proxies.
- The participants understand how reverse proxy can be used to rewrite authentication schemes (for example, from form-based to basic authentication).
- The participants should understand which infrastructure components in the request/response cycle affect data throughput and latency.
- The participants should understand how browser clients load resources and, in particular, evaluate HTML pages and query referenced resources.

- The participants understand how outsourcing resources to a content delivery network (CDN) helps to reduce the load on the internal infrastructure.
- The participants understand that not all web browsers support the standards to the same extent.
- The participants understand that the various web servers sometimes differ greatly in terms of resource consumption

What should participants know?

- The participants know various standard mechanisms that can be used to expand web servers with application logic (CGI, servlets, …).
- The participants know different approaches for load balancing a web application (e.g., DNS-based load balancer or proxies).
- The participants know different strategies such as graceful degradation and progressive enhancement to adapt the user interface to the capabilities of the web browser.
- The participants know frameworks for both CSS and JavaScript that can facilitate development.
- The participants know that HTML5 includes various technology packages that contain new structures for HTML and new APIs for JavaScript.
- The participants know the importance of unobtrusive JavaScript: this includes principles and practices that clearly separate content and behavior.
- The participants know how web firewalls work and know what types of attacks they protect against.

4.3. References

[BITV 2011], [Brewer 2004], [Buschmann+1996], [Daswani+2007], [ESI], [HTML5-SSE], [Kopparapu 2002], [OWASP], [Pritchett 2002], [Stoneburner+2004], [Waldo+1994], [WCAG 2008], [Websockets], [W3C-Int]

5. Design of Web Architectures

Duration: 180 min	Practice time: 120 min

5.1. Terms and Principles

Data modeling, functional decomposition, representation, distributed system, CAP theorem, ACID, security of web applications, authentication, authorization, accessibility, internationalization/localization, HTML, CSS, JavaScript, separation of content, presentation and behavior, graceful degradation, progressive enhancement, unobtrusive JavaScript.

5.2. Learning Goals

What should participants be able to do?

- The participants can clearly differentiate between representation, formatting, and interaction in markup and take these into account when designing web applications (separation of concerns).
- The participants can assign different tasks of the client interface to formats from the canon of web standards:
  - HTML provides the structural data
  - CSS defines the presentation. Selectors select elements from the HTML structure and assign presentation properties to them.
  - JavaScript controls the behavior, provides interactivity via event handlers, and enables asynchronous server communication (AJAX).

What should participants understand?

- The participants understand the need for rules that ensure that only desired functionalities are implemented in the views/templates when using request-response frameworks.
- The participants understand that the responsiveness of the system can often be improved by using asynchronous processing appropriately.
- The participants understand the basic statement of the CAP theorem and know that a tradeoff must be sought between scalability/availability/distribution and consistency/up-to-dateness of the data.
- The participants understand how network errors in distributed systems can cause inconsistencies or special error states.
- The participants understand how architectures for consistent systems and systems with partition tolerance differ.
- The participants understand how attackers can exploit vulnerabilities using SQL injection, cross-site scripting (XSS) or client-side request forgery (CSRF).
- The participants understand the transaction principle ACID – atomicity, consistency, isolation, and durability.
- The participants understand the principle of eventual consistency (consistent after some delay).
- The participants understand that, in addition to technical aspects, the internal architecture of the server side is also influenced by
  - The philosophy of the framework used,

    - Strategies for caching static and dynamic content, and
    - Strategies for scaling the application.
- The participants understand that integration patterns such as messaging for loose coupling of the web application to background processing are suitable for long-running processes.
- The participants understand that any application that can be accessed from public networks will be attacked.

What should participants know?

- The participants know the patterns command query separation, CQS, and command query responsibility separation, CQRS, in order to specifically fulfill both functional and non-functional requirements.
- The participants know that the consistency from “ACID” is a different consistency than the consistency from CAP.
- The participants know the basic mechanisms used to prevent SQL injection, cross-site scripting (XSS), and client-side request forgery (CSRF) in their own technology portfolio.
- The participants know different strategies for managing session-related data, e.g., in the cookie, in the main memory of the server, in a database, to meet quality requirements.
- The participants know the difference between internationalization and localization.
- The participants know various strategies for informing a client about the result of an activity that was started asynchronously on the server. For example:
    - Polling (client pull)
    - HTML5 server-sent events
    - WebSockets.
- The participants know that many server-side frameworks allow plug-ins, analogous to pipe-and-filter patterns, such as servlet filters in Java Servlet API-based frameworks or HTTP modules in ASP.NET.
- The participants know at least one server-side framework that can be used to implement web applications.
- The participants know different types of server-side web frameworks:
    - Component frameworks attempt to transfer an event-based programming model from the field of desktop applications.
    - Action or request-response frameworks directly map the HTTP protocol.
    - Data-driven frameworks map database tables in the user interface for data entry.
- The participants know authentication mechanisms such as OpenID Connect or CAS.

5.3. References

[abbott], [BITV 2011], [Brewer 2004], [Buschmann+1996], [Daswani+2007], [ECMA-262], [Evans 2004], [Fowler 2011], [Hohpe+2003], [HTML5], [HTML-CSS], [HTML5-SSE], [Nottingham 1998], [Olsson 2007], [OWASP], [Pritchett 2002], [Stoneburner+2004], [Waldo+1994], [WCAG 2008], [Websockets], [W3C-Int]

6. Quality in Web Architectures

Duration: 120 min	Practice time: 60 min

6.1. Terms and Principles

Security, scalability, web-scale, availability, operability, accessibility

6.2. Learning Goals

What should participants understand?

- The participants understand the basics of risk analysis and threat modeling.
- The participants understand the principles of secure software design:
  - Principle of Least Privilege: give users and components only as many rights as they absolutely need.
  - Defense in Depth: do not trust that other security measures have worked – use redundant checks.
  - Secure by Default: rights must be granted explicitly.
  - Don’t trust anybody: user input and data from external sources must be validated.
  - Compartmentalize: separate components that require different rights from each other.
  - Fail securely: if errors occur, no security-relevant information may be disclosed.
- The participants understand that the number of accesses to the Internet can increase in leaps and bounds beyond all expected levels.
- The participants understand that backend systems (such as databases or enterprise information systems) can easily become overloaded when the number of requests increases because one request often leads to a lot of queries to the backend.
- The participants understand that different persistence strategies (for example, relational, aggregate/document-oriented, column-oriented, hierarchical, object-oriented, graph-oriented) can support different usage scenarios with varying degrees of success.
- The participants understand that quality depends on both the appropriateness of the chosen technology and the specific implementation.
- The participants understand that internationalization means more than just offering content in different languages. For example:
  - Formats for numbers or dates or the order in which words are sorted depend on the culture.
  - Text directions may differ.
  - Layouts must take into account that texts in different languages take up different amounts of space.
  - The legal framework in different countries must be taken into account.
  - Cultural differences affect, for example, the choice of colors.

What should participants know?

- The participants know the key figures that are relevant to the operation of a web application:

- Number of processes/threads
- RAM usage
- Average resource consumption per request
- Number of open connections
- Average response time

7. Examples

Duration: 60 min	Practice time: none

7.1. Terms and Principles

Specific examples of different areas of web architecture must be presented, discussed, and evaluated within each accredited training course:

- Dealing with infrastructure and, in particular, reverse proxies.
- A rough design of a web backend application.
- One or more examples of single-page framework features such as data binding or client-side routing.
- Examples of different database models.

Type and characteristics of the presented examples can depend on the training course or the interests of the participants and are not specified by the iSAQB.

References

This section contains sources that are referenced in the curriculum in whole or in part.

A

- [Abbot+2010] Abbott, M. L., M. T. Fisher: The Art of Scalability, Addison-Wesley 2010

B

- [BITV 2011] Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz (Barrierefreie-Informationstechnik-Verordnung - BITV 2.0) – http://www.gesetze-im-internet.de/bitv_2_0/BJNR184300011.html
- [Brewer 2004] Brewer, E.: Towards Robust Distributed Systems. Keynote zur PODC (Symposium on Principles of Distributed Computing) 2000. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- [Buschmann+1996] Buschmann, F, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: A System of Patterns. Pattern Oriented Software Architecture. Wiley 1996

C

- [Crockford+2008] Crockford, Douglas: JavaScript – The good Parts. O'Reilly, 2008

D

- [Daswani+2007] Daswani, N., C. Kern, A. Kesavan: Foundations of Security: What Every Programmer Needs to Know. Apress 2007

E

- [ECMA-262] ECMAScript Language Specification – http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm
- [Evans 2004] Evans, E.: Domain Driven Design, Addison Wesley 2004
- [ESI] ESI Language Specification 1.0. W3C Note 2001 - http://www.w3.org/TR/esi-lang

F

- [Fielding+2000] Fielding, R., R. Taylor: Principled Design of the Modern Web Architecture. In Proceedings of the 2000 International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 2000 - http://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf
- [Fowler 2011] Fowler, M.: CQRS. http://martinfowler.com/bliki/CQRS.html

H

- [Hohpe+2003] Hohpe, G, Woolf, B: Enterprise Integration Patterns, Addison-Wesley 2003
- [HTML5] Web Hypertext Application Technology Working Group (WHATWG): HTML - Living Standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/
- [HTML-CSS] W3C Standards für das Webdesign – http://www.w3.org/standards/webdesign/htmlcss

- [HTML5-SSE] Server-Sent Events. – http://www.whatwg.org/specs/web-apps/current-work/multipage/comms.html#server-sent-events

J

- [Jacobs+2004] Jacobs, I., N. Walsh: Architecture of the World Wide Web, Volume One. W3C Recommendation 2004, - http://www.w3.org/TR/2004/REC-webarch-20041215/

K

- [Kopparapu 2002] Kopparapu, C.: Load Balancing Servers, Firewalls, and Caches. Wiley 2002

N

- [Nottingham 1998] Nottingham, M: Caching Tutorial. http://www.mnot.net/cache_docs/

O

- [Olsson 2007] Olsson, T.: Graceful Degradation & Progressive Enhancement. http://accessites.org/site/2007/02/graceful-degradation-progressive-enhancement/
- [OWASP] Open Web Application Security Project (OWASP) - https://www.owasp.org/

P

- [Pritchett 2002] Pritchett, D.: BASE an ACID alternative. 2008 - http://queue.acm.org/detail.cfm?id=1394128

R

- [RFC2246] Dirks, T., C. Allen: RFC 2246, The TLS Protocol. http://www.ietf.org/rfc/rfc2246
- [RFC2616] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: RFC 2616, Hypertext Transfer Protocol — HTTP/1.1. https://tools.ietf.org/html/rfc2616
- [RFC3986] Berners-Lee, T., R. Fielding, L. Masinter: RFC 3986, Uniform Resource Identifier (URI): Generic Syntax. http://tools.ietf.org/html/rfc3986
- [RFC3987] Duerst, M., M. Suignard: RFC 3987, Internationalized Resource Identifiers. http://tools.ietf.org/html/rfc3987
- [RFC6265] Barth, A.: RFC 6265, HTTP State Management Mechanism. http://tools.ietf.org/html/rfc6265

S

- [Stoneburner+2004] Stoneburner, G., C. Hayden, A. Feringa: Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A. NIST Special Publication 800-27 Rev A 2004 – http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf

T

- [Tilkov 2011] Tilkov, S.: REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien. Dpunkt 2011

W

- [W3C-Int] W3C Internationalization Activity - http://www.w3.org/International/
- [Waldo+1994] Waldo, J., G. Wyant, A. Wollrath, S. Kendall: A Note on Distributed Computing. Sun Microsystems 1994 – http://labs.oracle.com/techrep/1994/smli_tr-94-29.pdf
- [WCAG 2008] Web Content Accessibility Guidelines Working Group: Web Content Accessibility Guidelines. W3C 2008 – http://www.w3.org/WAI/intro/wcag
- [Websockets] Web sockets - http://www.whatwg.org/specs/web-apps/current-work/multipage/network.html#network
- [WS-I] WS-I Profiles - http://ws-i.org/deliverables/Default.aspx