Review Questions Web Engineering

Felix Zailskas

2020/2021

1 Introduction

- **How are the terms HyperText and HyperMedia defined?**
  HyperText: A HyperText is a **non-linear** textform. It contains links to other texts.
  HyperMedia: HyperMedia is a HyperText which can contain more than text. This includes **graphics, video, sound, etc.**.
  Both of the terms were coined by **Ted Nelson in 1965**. HyperText and HyperMedia are concepts not products.

- **What are the key concepts of Project Xanadu? Bonus: how do they defer from their equivalents in the Web?**
  Project Xanadu: Project Xanadu was the first HyperText project founded by **Ted Nelson** in **1960**. Documents were compositions of text fragments, with multiple versions, composed of **bidirectional links**. Founders of the project claimed the World Wide Web to be a trivialisation of the project since it uses a HyperText model with **one-way links** and no management, versions or contents.

- **What are the steps and stakeholders involved in W3C's (World Wide Web Consortium) standardization process?**
  Stakeholders: W3C is the **main international standardization organization** for the world wide web. They try to establish a standard among industry members defined by them. Vendors using outdated HTML versions are causing inconsistencies on how web pages are displayed. W3C tries to make these vendors implement the set of standards set by the consortium.
  Steps:

  1. A member or someone from the general public expresses an interest for a standard.
  2. If the interest reaches sufficient mass then a working group is formed on that topic.
  3. A charter for the working group is defined.

4. The working group produces specifications and guidelines in review cycles.
5. The **advisory committee** decides if/when the specifications and guidelines can be realeased as official recommendations.

2 Foundations of the Web

- **What are the layers of the TCP/IP model and what is their role?**

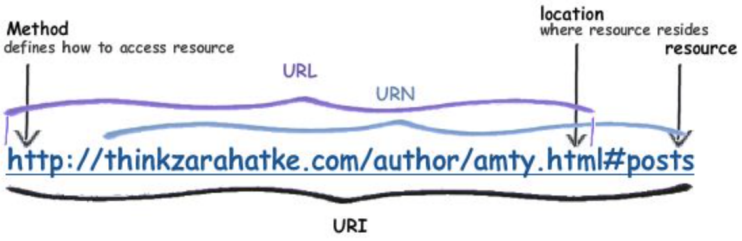
1. Link Layer:
   The link layer is the lowest level layer of communication protocols. It deals with **translating data from higher layers to the physical layer**, and sending information between **hosts in the same local network.**
   Protocols in this layer describe how data interacts with the transmission medium, such as electronic signals send over hardware. Therefore, this **layer is dependent on the used hardware**.
2. Internet Layer:
   Protocols in this layer describe how data is sent and received over the internet. This includes packaging data, addressing and transmitting packages and receiving packages.
   Most commonly IP (internet protocol address) is used for this layer. IP is a **connection less protocol**, which means it does not ensure packages to be sent in the right order, on the same path or even in its entirety. There is IPv4 using 32 bit addresses and IPv6 using 128 bit addresses.
3. Transport Layer:
   The protocols in the transport layer handles the reliability of sent and received packages. The layer encapsulates TCP and UDP.
   TCP is a **connection-oriented transport layer protocol** that prioritizes reliability over latency, or time. TCP describes transferring data in the same order as it was sent, retransmitting lost packets, and controls affecting the rate of data transmission.
   UDP is **connectionless** and can be used to prioritize time over reliability.
4. Application Layer:
   This layer contains protocols communicating with software applications. The specification includes descriptions of the remote login protocol Telnet, the File Transfer Protocol (FTP), and the Simple Mail Transfer Protocol (SMTP). Also included in the application layer are the Hypertext Transfer Protocol (HTTP) and its successor, Hypertext Transfer Protocol Secure (HTTPS).

- **What do the acronyms ISP, POP, and IXP stand for and what is their purpose?**

    1. ISP (Internet Service Provider):
    The ISP acts as a gateway between a user and the internet. It is a company that provides services within and access to the internet. Services can include Internet access, Internet transit, domain name registration, web hosting, Usenet service, and colocation.
    2. POP (Point of Presence):
    The POP is an artificial interface point between communicating entities. It allows users to connect to the internet with their ISPs. A POP typically houses servers, routers, network switches, multiplexers, and other network interface equipment, and is typically located in a data center.
    3. IXP (Internet eXchange Points):
    An IXP is essential technical infrastructure where networks come together to connect and exchange Internet traffic. IXPs reduce the portion of an ISP's traffic that must be delivered via their upstream transit providers, thereby reducing the average per-bit delivery cost of their service.

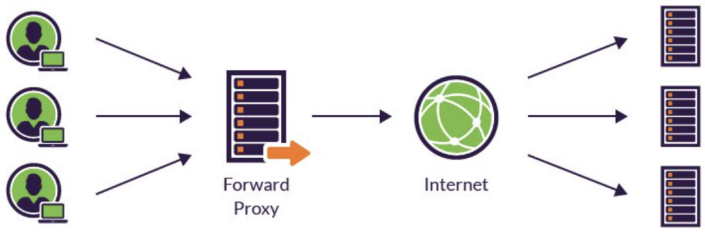
- **How are the terms URI, URL, and URN defined, what is their purpose, and what is the relation between them?**

    1. URI (Uniform Resource Identifier):
    URIs are texts used to uniquely identify any resource or name on the internet. They are subcatigorized into URLs and URNs.
    2. URL (Uniform Resource Locator):
    URL includes location as well as the protocol to retrieve the resource. Protocols could be *ftp://*, *https://* or *ldap://*. In the example below you can see the protocol is *http://* and the location is *thinkzara-hatke.com* we are trying to access the resource *amty.html*.
    3. URN (Uniform Resource Name):
    URN stands for Uniform Resource Name. URN is also the subset of URI. One of the best examples of URN is ISBN number which is used to uniquely identify a book. URN is completely different than URL as it doesn't include any protocol.

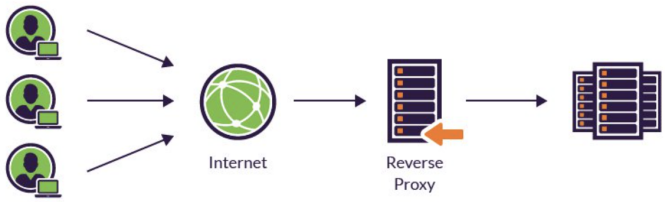


- **Which types of proxy servers are available, and what is their role?**
  A proxy server is a program that acts on behalf of the origin server. A client sends requests to the proxy instead of directly to the origin server. In case that it cannot be computed locally, the proxy sends a request to the origin server and then sends the result to the client. A proxy might change all messages before passing them on.

  1. Forward Proxy:
     A forward proxy gives a user access to the internet in an otherwise firewall-restricted network. The user sends a connection request to it and it retrieves data from the internet.



  2. Reverse Proxy:
     A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.



  3. Web Accelerator:

A web accelerator is a proxy reducing access time to a website. It can be self contained hardware or installable software. They can include a range of methods to achieve this including but not limited to:

(a) Caching
(b) Prefetching
(c) Compression
(d) Optimize code
(e) Filter out ads
(f) Sped up encryption
(g) Maintain TCP connection between client and proxy

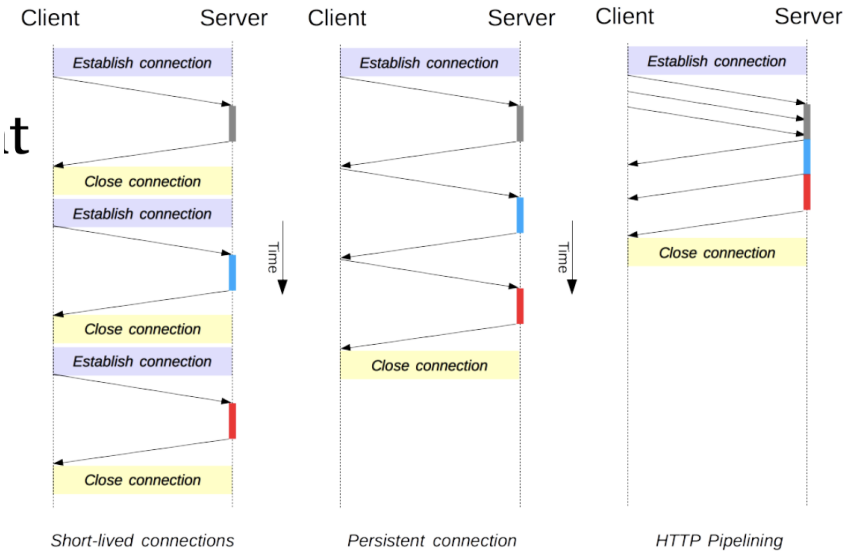
4. Edge Proxy:
The Edge proxy is the endpoint to which the User-Agent (e.g. web browser) connects and it has a few responsibilities:
   (a) Performing the TLS handshake for HTTPS connections
   (b) Routing requests to the corresponding application proxy stack
   (c) Implementing the HTTP/2 protocol
   (d) Request correlation
   (e) Serving custom error pages

- **Which types of content negotiation are available for HTTP (HyperText Transfer Protocol), and what are their advantages and disadvantages?**
Content negotiation handles what data representation should be delivered to a client. The client requests the resource and the content negotiation mechanism determines what representation should be returned.

1. Server-driven:
In this type of content negotiation the client reports the favoured content type using a header in their request. The server then tries to deliver resources as requested.
Pros: A second request from the client specifying content type is avoided.
Cons: The needed content type depends on what the user wants to do with the resource (view, print,...) but this cannot be declared in a header.
2. Client-driven:
In this type of content negotiation the server responds to a request with a list of possible variants. The user then chooses one of the variants that best suites the intended use.
Pros: The intended use can be taken into account when choosing a variant. This works well with scripting.
Cons: Every request needs at least two requests.

- **Describe the different connection management strategies used by the HTTP versions 1.0, 1.1 and 2**

  1. HTTP 1.0 In this version of HTTP connections are short lived. For each request a new connection is established between the client and the server, which is closed as soon as the server has sent a response.
  2. HTTP 1.1 In this version of HTTP multiple requests can be sent in a consistent connection. Request processing is pipe-lined and the connection is closed after the requests have been processed.
  3. HTTP 2.0 In this version of HTTP only one connection is established over which multiple multiplexed requests are send and responded to.



- **Under which conditions are HTTP requests safe/idempotent? Which HTTP methods are considered safe/idempotent?**

  Safe: A HTTP request is considered safe if it does not alter the server state. This means it leads to read-only operations.
  The requests GET, HEAD, OPTIONS, TRACE are safe.

  Idempotent: Idempotent requests have no side effects. This means the same request can be repeated any number of times yielding the same result, the server will stay in the same state.
  The requests GET, HEAD, OPTIONS, PUT, DELETE are idempotent.

NOTE: All safe methods are idempotent but not vice versa. POST and PATCH are neither.

- **What are the differences between PUT and POST in terms of request URI semantics, and pragmatics (i.e. how they are to be used)?**
  PUT: URI in a PUT request identifies the entity enclosed with the request. PUT should be used to create new entities or replace an old one on the server.
  N requests of PUT will result in 1 entity with the provided data.
  POST: URI in a POST request identifies the resource that will handle the entity enclosed in the request.
  POST should be used to send data to the server and replace entities. N requests of POST will result in N different entities with the provided data.

- **Name and describe the main/ support methods used in HTTP**
  Main:

  1. DELETE: This request is used to delete an entity from the server. It modifies the server state therefore it is not safe. Since the server cannot delete the same entity twice, DELETE is idempotent. The servers response only indicates that the resource has been marked for deletion, not that it has been deleted.
  2. GET: This request is used to retrieve a representation of a resource from the server. The resource is only retrieved and not changed at all. This makes GET a safe and idempotent request.
  3. POST: This request is used to send data to the server to update or create an entity. The data sent is stored in the body of the HTTP request. This means that this request is not safe, POST is not idempotent since 2 POST requests with the same data create 2 distinct entities.
  4. PUT: This request is also used to send data to the server to update or create an entity. The data sent is stored in the body of the HTTP request. This means that this request is not safe. However, opposed to POST, 2 PUT requests with the same data only create 1 entity with the data, this makes PUT idempotent.
  5. PATCH: This request is used to send data to the server and partially update an entity. The data sent is stored in the body of the HTTP request. This means that this request is not safe. A PATCH request can be idempotent but does not have to be. An example of a non idempotent PATCH request would be appending data to an entity, while a PATCH request of the form *PATCH /users/42 {"name": "john doe"}* would be idempotent.

  Support:

1. CONNECT: This request establishes a tunnel to the server identified by the target resource. It is not safe nor idempotent.
2. HEAD: This request is almost identical to the GET request, however it does not retrieve the response body. This means if *GET /users* would return a list of users, then *HEAD /users* would make the same request but not return the list of users. HEAD is useful to check what a GET request would return without actually making a GET request (e.g. downloading a large file) or to validate cached response messages. Like GET this request is safe and idempotent.
3. OPTIONS: This requests returns the possible communication options of the target resource. This request is safe and therefore also idempotent.
4. TRACE: This request returns only status codes resulting from the request. It performs a message loop-back test along the path to the target resource, providing a useful debugging mechanism. This method is safe and therefore also idempotent.

- **What are the different status codes for HTTP responses and when are they used?**

1. 1xx Informational: Informs client that the request has been received and will be processed further.
2. 2xx Sucess: Informs client that the request has been successfully received, accepted and understood.
3. 3xx Redirection: Informs the client that further action must be taken in order to complete the request.
4. 4xx Client Error: Informs the client that the request contains bad syntax or cannot be fulfilled.
5. 5xx Server Error: Informs the client that the server failed to fulfill an apparently valid request

- **Name and describe the some of the common general/ entity/ request/ response headers used in HTTP**
  General:

1. Chache-Control: The Cache-Control HTTP header holds directives (instructions) for caching in both requests and responses. A given directive in a request does not mean the same directive should be in the response.
2. Connection: The Connection general header controls whether or not the network connection stays open after the current transaction finishes. If the value sent is *keep-alive*, the connection is persistent and not closed, allowing for subsequent requests to the same server to be done. The server will drop connection after response if *close* is set.

3. Transfer-Encoding: The Transfer-Encoding header specifies the form of encoding used to safely transfer the message body to the user. If set to *chunked* then the message body is sent as sequence of chunks. If set to *gzip* then the message body is coded in gzip-format.
4. Via: The Via general header is added by proxies, both forward and reverse proxies, and can appear in the request headers and the response headers. It is used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of senders along the request/response chain.
5. Date: The Date general HTTP header contains the date and time at which the message was originated.

Entity:

1. Content-Encoding: The Content-Encoding entity header is used to compress the media-type (e.g. zip, jpeg). When present, its value indicates which encodings were applied to the entity-body. It lets the client know how to decode in order to obtain the media-type referenced by the Content-Type header.
2. Content-MD5: The Content-MD5 header is used by servers to provide a message-integrity check for the message body. Allows to detect at server whether resource has been changed
3. Expires: The Expires header contains the date/time after which the response is considered stale. Invalid dates, like the value 0, represent a date in the past and mean that the resource is already expired.
4. Last-modified: The Last-Modified response HTTP header contains the date and time at which the origin server believes the resource was last modified. It is used as a validator to determine if a resource received or stored is the same.

Request:

1. Host: The Host request header specifies the host (URI or IP) and port number of the server to which the request is being sent. If no port is included, the default port for the service requested (e.g., 443 for an HTTPS URL, and 80 for an HTTP URL) is implied.
2. If-Match: The If-Match HTTP request header makes the request conditional. For GET and HEAD methods, the server will send back the requested resource only if it matches one of the listed entity tags. For PUT and other non-safe methods, it will only upload the resource in this case.
3. If-Modified-Since: The If-Modified-Since request HTTP header makes the request conditional: the server will send back the requested resource, with a 200 status, only if it has been last modified after the given date. If the request has not been modified since, the response will be a 304 without any body.

4. User-Agent: The User-Agent request header is a characteristic string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent.

Response:

1. Age: The Age header contains the time in seconds the object has been in a proxy cache. It is usually calculated as a difference between the proxy's current date and the Date general header included in the HTTP response.
2. Server: The Server header describes the software used by the origin server that handled the request — that is, the server that generated the response.
3. Accept-Ranges: The Accept-Ranges response HTTP header is a marker used by the server to advertise its support of partial requests. The value of this field indicates the unit that can be used to define a range. In presence of an Accept-Ranges header, the browser may try to resume an interrupted download, rather than to start it from the start again.
4. Retry-After: The Retry-After response HTTP header indicates how long the user agent should wait before making a follow-up request. There are three main cases this header is used:
   - When sent with a 503 (Service Unavailable) response, this indicates how long the service is expected to be unavailable.
   - When sent with a 429 (Too Many Requests) response, this indicates how long to wait before making a new request.
   - When sent with a redirect response, such as 301 (Moved Permanently), this indicates the minimum time that the user agent is asked to wait before issuing the redirected request.
5. WWW-Authenticate: The HTTP WWW-Authenticate response header defines the authentication method that should be used to gain access to a resource. The WWW-Authenticate header is sent along with a 401 *Unauthorized* response.

- **Which proxy types are common on the Web, and what is their function?**

1. Forward Proxy (see above)
2. Reverse Proxy (see above)
3. Tunneling: Tunneling transmits private network data and protocol information through public network by encapsulating the data. HTTP tunneling is using a protocol of higher level (HTTP) to transport a lower level protocol (TCP).

- **What are the benefits of caching?**
  Caching is the re usage of previously fetched resources. Web caches reduce latency and network traffic and thus lessen the time needed to display a representation of a resource. The cache returns a resource that it stores when it contains it, instead of redirecting the request to the main server. Therefore, it is crucial that the resources only stay in the cache as long as they did not change.
  There are two types of caches:

  1. Private cache: This type of cache is dedicated to one user only. A browser cache holds all documents downloaded via HTTP by the user.
  2. Shared cache: This type of cache stores data to be re used by more then one user. For example, an ISP or your company might have set up a web proxy as part of its local network infrastructure to serve many users so that popular resources are reused a number of times, reducing network traffic and latency.

- **How is the principle of Semantic Transparency for HTTP defined? Under which conditions is it violated by the use of caches?**
  The principle of semantic transparency describes that the usage of a proxy or cache cannot have impact on the user or origin server. Each request should yield the same result as if it has been served from the main server. Now if we imagine that a cache holds on to a resource longer than that resource is valid, it will give the user a wrong resource. The user's request result deviated from what the origin server would have produced, and so the principal of semantic transparency has been violated.

- **How can a cache determine if it is fresh in the absence of an Expires header?**
  There are multiple ways for a cache to determine if resources are still valid. A resource can carry an expire header containing a date. This date specifies when the resource has become invalid.
  A response can carry a directive *max-age* this determines the time in seconds that the response is still valid. *max-age* overwrites the expire header. The cache uses a freshness period (e.g. date + max-age > current date) to determine when a resource has become outdated.

- **How can a server stop a proxy or client from caching a response?**
  A server should stop a caching when the resource is known to constantly change. This can be done in two ways.
  The header can carry a past date as the expiry date.
  The *cache-control* header field is set to *must-revalidate*.

- **How does the Basic authentication scheme for HTTP work? Under which conditions should it be used?**
  In order to access resources requiring authentication a request is sent with

additional credentials. These are user ID and password pairs encoded in base64 encoding. This method is effectively not secure because the credentials are plain text. It is usually used in conjunction with HTTPS to ensure proper confidentiality.

- **What are the benefits of using CDNs?**
  A content delivery network (CDN) provides fast delivery of internet content. A CDN allows for the quick transfer of assets needed for loading Internet content including HTML pages, javascript files, stylesheets, images, and videos. They can be seen as "reverse" edge proxys.
  Upsides of using CDNs:

  1. Reduced latency (less network hops)
  2. Scaling to demand
  3. Increased reliability
  4. Abstraction from data transfer
  5. Security against DDOS attacks

3 REpresentational State Transfer (REST)

- **Which constraints define the REST architectural style, and how are they related with each other?**
  REST can essentially be seen as a set of constraints, which should be adhered when designing an API. The restraints should improve scalability, remixability, usability and accessibility of the program.

  1. Resource Identification (RI): Every resource that can be accessed should be named. This also means that anything that can be named is a resource. In the web URIs provide a way global addressing space for resource and service discovery.
  2. Uniform Interface (UI): There should be a small set of operations (GET, POST, DELETE,...) that can be applied to all resources. This results in a small set of verbs and a large set of nouns. Verbs can be expanded if needed. Operations should adhere to the CRUD (Create, Retrieve, Update, Delete) principle.
  3. Self-describing Messages (SDM): Resources are abstract entities, they cannot be accessed directly, we identify them (RI) and access them (UI). The resources are accessed by accessing a representation of the resource. Which representation is used (e.g. JSON, XML, CSV, XHTML, SVG, RDF,...) is made clear, then this is sufficient.
  4. Hypermedia as the Engine of Application State (HATEOAS): The resource representation (SDM) contains a link to the identifiable resource (RI). Now the resources and their representations can be accessed through link navigation. RESTful applications navigate with

traversal paths contained in the resource representation. Link semantics determine the navigation to the next resource.

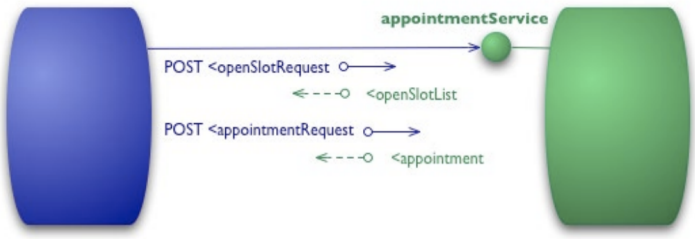
5. Stateless Interactions (SI): States on the server side of the application should be avoided. No history of requests is kept the server should treat every request like a new one. The resource state can be managed by the server it is the same for all clients and can be changed by the client. Client state managed by the client itself. Each client manages its own state.

- **How are the REST style constraints related to its goals?**

1. Scalability: By making the system stateless, it is easy to scale the systems user base. Also by the UI it is easy to add new resources since they only have to interact with a few words.
2. Simplicity: RESTful desing adheres to well established stadards. This simplifies the design and implementation process.
3. Data independence: Because of the SDM a plethora of users can access the resources in a way that suites their needs. Each resource can be represented in multiple ways.
4. Performance: The usage of lightweight message formats like JSON (SDM) allows for better performance.

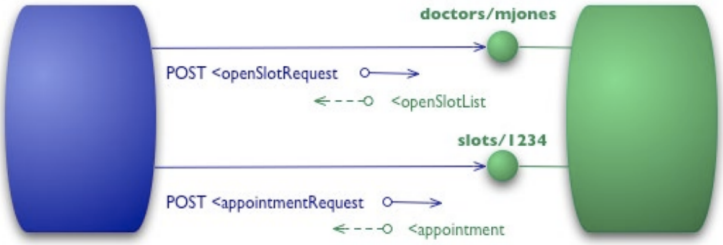
- **What are the maturity levels in Richardson's model? Which REST principles are they related to?**

1. Level 0: POX (Plain old XML)
   HTTP is used as a transport system for remote interactions. No web mechanics are used, HTML is a tunneling mechanism for RPC. Resources are identifiable and can be represented in any way. Data and meta-data is contained in the message body.

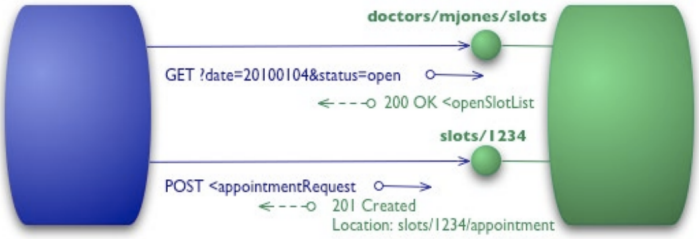


2. Level 1: Resources (RI)
   In this level all resources are uniquely identifiable. For example doctors or appointment slots. This directly corresponds to the resource

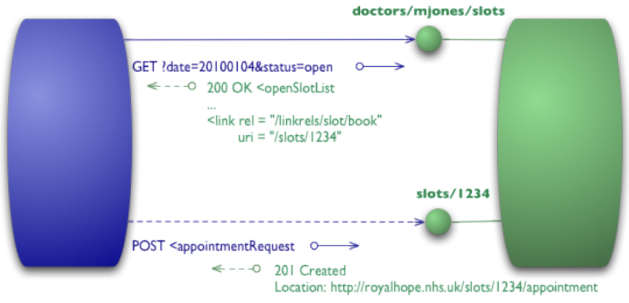
identification restraint.



3. Level 2: HTTP verbs (UI & SDM)
   This level adds the usage of HTTP verbs. This means that resources are called with a verb based on how the user wants to interact with the resource. Responses always carry a status code telling the user about success or failure of the request. Meta-data is used to identify the resource in URI. This corresponds to the uniform interface and self describing messages restraints.



4. Level 3: Hypermedia Controls (HATEOAS & SI)
   This level introduces hypermedia controls. This means that it provides the representation (link) to the next valid state change operations of the resource. It also provides the URI of the resource in the response. This allows for dynamic adjustment of links to resources. This corresponds to the Hypermedia as the Engine of Application State and Stateless Interactions restraints.

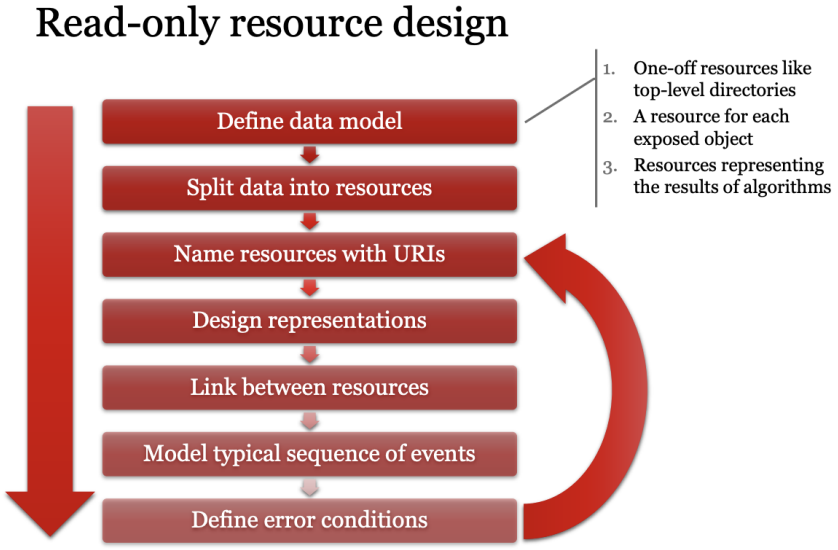


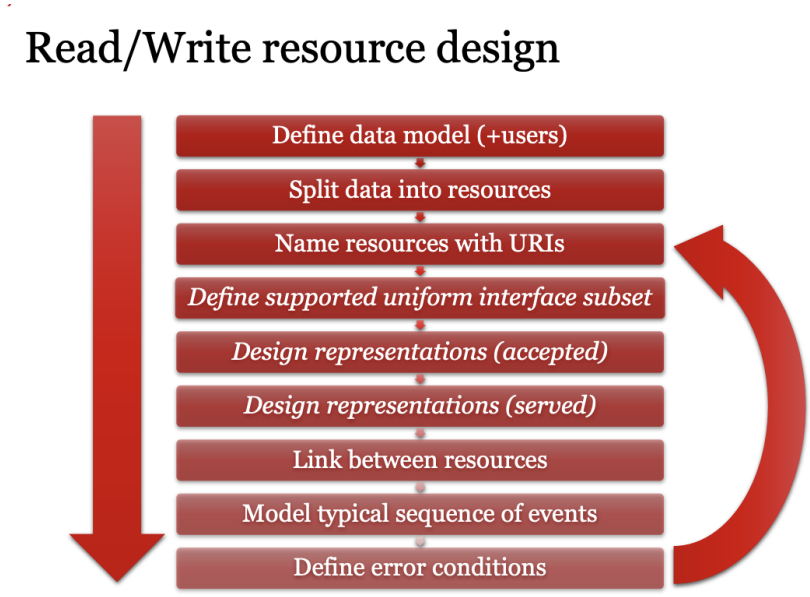
- **Which best practices apply to the design of URIs according to Masse's book?**

1. Hierarchical relationships should be represented by a /.
   http://localhost:8080/Category/Subcategory/Products
   Here a category contains a subcategory which in turn contains products.
2. Plural nouns should be used when appropriate.
   http://localhost:8080/Organization/Departments/1
   This indicates that an organization has many departments.
3. Design should improve readability. This means to use lower case letters, - instead of _, and avoid special characters.
   http://localhost:8080/House/number-of-rooms
   instead of
   http://localhost:8080/House/#ofRooms

4. File extensions should not be used (.html, .asp,...)
5. Query parameters should be used for filtering.
   http://localhost:8080/Organization/Departments?name=HR
   Here the query paramenter is *?name=HR*
6. URIs should not be used wiht CRUD operations.
   Do not use
   http://localhost:8080/Organization/get-Departments/1
   but rather
   http://localhost:8080/Organization/Departments/1 coupled with a GET request.

- **What steps should be followed for the design of a REST API?**





4 Service Oriented Architecture (SOA)

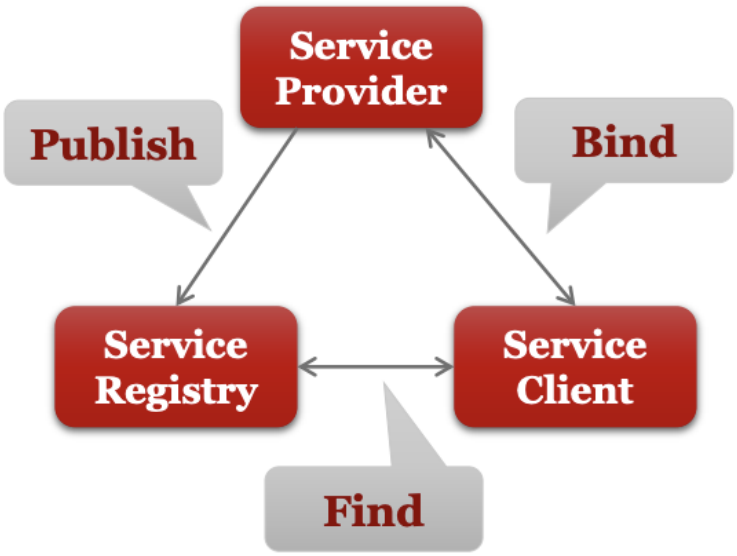
- **What are the main constituents of a service in SOA, and what is their purpose?**
  SOAs have a set of five basic concepts that make up the architectual paradigm.

  1. Services: Services are encapsulated functionality abstractions. They obscure internal implementation designs while exposing clearly defined interfaces.
  2. Infrastructure: The infrastructure of a SOA combines the services contained in an easy and flexible manner.
  3. Policies and Processes: Policies and processes need to be implented that deal with the heterogeneity, changeability and multiple ownership of large distributed systems.
  4. Interoperability: Connecting to other systems should be easily possible.
  5. Loose coupling: Dependencies between systems should be kept as low as possible. This leads to fault-tolerance, flexibility and scalability. Different levels of coupling can be present in the same system.

  Now services of a SOA have two main components.

1. Interface: This is the part visible to the external world. It is the means to access all functionality of the system and should therefore be simple to understand and self-explanatory.
2. Implementation: This is the backend of a service. It realizes specific service interfaces. It can use multiple languages or platforms and can even use other services to implement the functionality.

- **What are the main interaction roles in the SOA triangle, and what operations are they supporting?**



The SOA triangle has three components.

1. Provider: This is the organization that owns the service and implements all business logic behind it. It also hosts the service and controls access to it.
   It publishes the description/registration of the service and binds itself to a client via an invocation.
2. Registry: This is a searchable registry in which services are registered and described.
   It allows users to find suitable services and learn how to properly use them.
3. Client: Clients can be organizations that need a certain functionality to be satisfied or another application or service using the service.
   Clients can discover services through the registry and find informa-

tion about them. Also they bind themselves to the provider by using a service.

5 Architectural Concerns

- **What are the functionalities encapsulated in each of the application layers in the respective pattern by Fowler?**
  According to Fowler there are three distinct application layers.

  1. Presentation layer: This layer is completely focused on the presentation of resources. Its tasks are rendering data, reacting to events and processing sessions via communication/ conversation logic.
  2. Application Logic Layer: This layer implements the functions offered by the presentation layer. It also contains the logic used by the application.
  3. Resource Layer: The logic to access data needed by the application logic is embedded in this layer. It consists of databases, files, content, queues and other support functions.

- **What is the relation between layers and tiers? Which architectural decision is important for this decision?**
  Layers describe functionality and logic that is contained inside of a conceptual block of the application. These are split into the above mentioned three layers. The layers can be seen as functional organization units.
  Tiers are an accumulation of the layers. If we take the two-tier model with client and server as an example the question appears on how much functionality should be on the client side and how much on the server side. Tiers can therefore be seen as physical organization units.
  We differentiate between thick and thin clients.

  1. Thin client: Thin clients have little to no presentation logic. They essentially act as dumb terminals to the program.
  2. Thick client: Thick clients host some of the presentation logic and sometimes more logic than that. Depending on the used model a thick client can be responsible for the presentation layer and part of or all of the application layer. The resource layer stays on the server side.

- **Where does the application split lie in the case of remote presentation/distributed/remote data applications? Name examples of such types of applications**

  1. Remote presentation: In these types of applications the client is responsible for the presentation layer, while the server is operating the application and resource layer. This split can be found in single page applications (SPA) or multi-device web applications.

2. Distributed applications: In these types of applications the client is responsible for the presentation layer, while the server operates the resource layer. The application layer is partly done by the client and partly by the server. This often means that some processing is delegated from the client to a funciton on the server. This can also be seen in SPAs and many current web applications.
3. Remote data applications: In these types of applications the client is responsible for the presentation layer and all of the application layer, the server is responsible for the resource layer. Data is accessed by high-level interfaces to resources. This type of application provides location transparency to resources. This architecture is often seen in mobile/ platform games.

6 Web Technologies

- **What is the difference between text elements with semantics and those without in HTML? Which ones are recommended to be used, and why?**
  Semantic elements are elements that clearly describe their meaning to the browser and developer. Non-semantic elements are ambiguous about what meaning they have.

  Examples of non-semantic elements: <*div*> and <*span*> - Tells nothing about its content.

  Examples of semantic elements: <*form*>, <*table*> and <*article*> - Clearly defines its content.

  Generally it is recommended to use semantic elements if possible. Non-semantic elements should only be used if no meaning is attached to them, mostly for CSS decorating.

- **How was embedded video and audio handled up to HTML5? How are they handled by HTML5? What are the implications of this mechanism?**
  Before HTML5 non-native web technologies (e.g. Flash) had to be used to handle embedded video and audio on a web page. HTML5 introduced the <*audio*> and <*video*> elements. Both of the use of the default *control* attribute and the use of a JS API is possible. This breaks the compliance to SGML (Standard Generalized Markup Language) since now HTML enforces control in addition to markup.

- **What are the types of HTML form validation available, and when are they to be used?**
  Since data provided by users should never be trusted it is important to validate content of forms before using them in the application. There are

two types of form validation server and client-side validation. The built in validation possibilities of HTMl are all client side validation.
When an element is valid, the following things are true:

- The element matches the :valid CSS pseudo-class, which lets you apply a specific style to valid elements.
- If the user tries to send the data, the browser will submit the form, provided there is nothing else stopping it from doing so (e.g., JavaScript).

When an element is invalid, the following things are true:

- The element matches the :invalid CSS pseudo-class, and sometimes other UI pseudo-classes (e.g., :out-of-range) depending on the error, which lets you apply a specific style to invalid elements.
- If the user tries to send the data, the browser will block the form and display an error message.

The following HTML validations are available:

1. *required* Specifies whether a form field needs to be filled in before the form can be submitted. This should be used for things like e-mail address/ user name when logging into a system.
2. *minlength/ maxlength* Specifies the minimum and maximum length of textual data (strings). This should be used when only a certain amount of space is allocated for the form field.
3. *min/ max* Specifies the minimum and maximum values of numerical input types. This should be used if a field can only be between a range of numbers (e.g. birth year can not lay in the future).
4. *type* Specifies whether the data needs to be a number, an email address, or some other specific preset type. This should be used when a certain format is required.
5. *pattern* Specifies a regular expression that defines a pattern the entered data needs to follow. This could be used for a phone number (e.g. country-code;actual phone number).

- **How is the cascade mechanism used to resolve conflicts between CSS rules?**
  If an element is affected by multiple rules in a CSS sheet and both have equal specificity, then the last one defined (at the bottom of the document) will apply to the element.
  For example: Here the heading is blue not red.


```
h1 {
    color: red;
}
h1 {
    color: blue;
}
```

```
<h1>This is my heading.</h1>
```

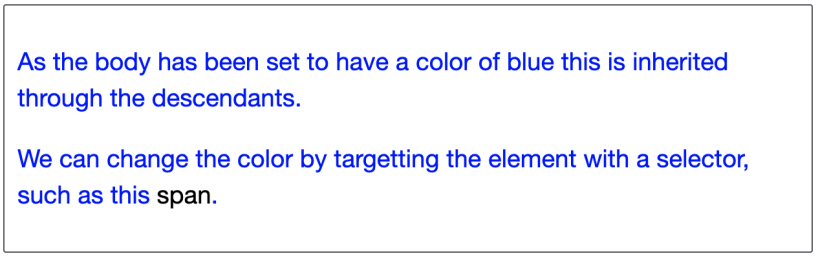
- **What is the order of specificity in for CSS rules?**
  If multiple rules, with different selectors, could apply to the same element then specificity is one way the browser decides which rule to apply to the element. The more specific selector will be applied.
  An element selector is less specific — it will select all elements of that type that appear on a page — so will get a lower score.
  A class selector is more specific — it will select only the elements on a page that have a specific class attribute value — so will get a higher score.
  An id selector (indicated as $\#id\{\}$) only applies to one element, since ids are unique. It has the highest specificity score.
  For example: The class selector is more specific than the header selector, so the header will be red.


```
.main-heading {
    color: red;
}

h1 {
    color: blue;
}
```

```
<h1 class="main-heading">This is my heading.</h1>
```

- **What is the inheritance in CSS?**
  Inheritance also needs to be understood in this context — some CSS prop-

erty values set on parent elements are inherited by their child elements, and some aren't.
For example, if you set a color and font-family on an element, every element inside it will also be styled with that color and font, unless you've applied different color and font values directly to them.



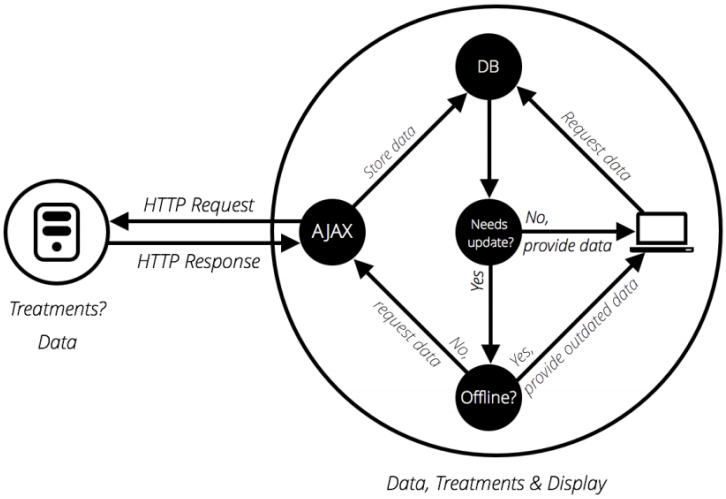

```
body {
    color: blue;
}

span {
    color: black;
}
```

```
<p>As the body has been set to have a color of blue this is inherited
through the descendants.</p>
<p>We can change the color by targetting the element with a selector,
such as this <span>span</span>.</p>
```

- **Which technologies are typically associated with the AJAX Web app model? What is their purpose?**
  Asynchronous JavaScript and XML (AJAX) applications allow non-blocking interaction with the back-end for client side applications. It agrgegates multiple technologies.

  1. HTML HTML is used for definition of the content.
  2. CSS CSS is used for styling and decorating of the content.
  3. DOM & JS DOM is used in combination with JS for dynamic content display.
  4. XML XML lets clients exchange application data with the server. Today XML is often replaced by JSON.

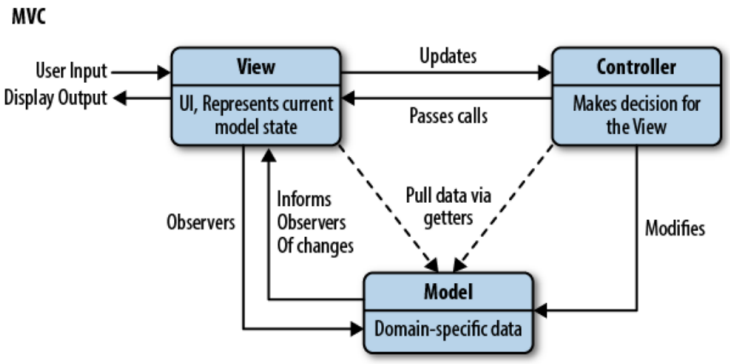
  The model uses web APIs as a proxy for more efficient data requests.



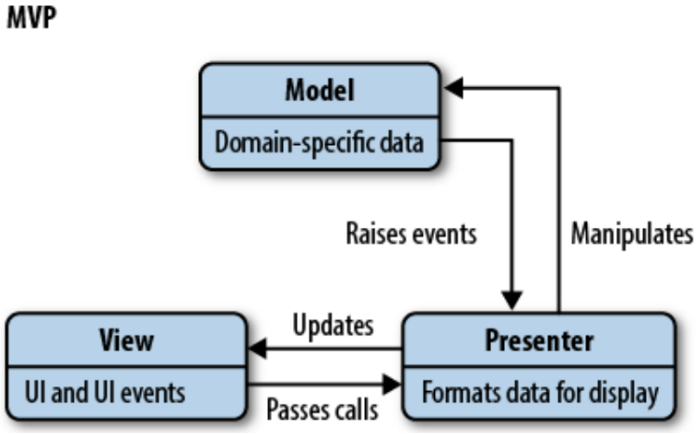
- **Under which conditions would you recommend the use of a SPA-style application?**
  Opposed to regular web applications, which removed most of the client side behaviour, single page applications (SPA) are heavily client side dynamic web pages. Content is loaded from static HTML and JS is used to run the application. Its main upside compared to regular web applications is the rich user interface possibilities. Familiarity with JavaScript/ TypeScript is required. So if an application with many requirements to the user interface is to be developed and the team has a sufficient background is JS/TS then I would recommend a SPA.

- **How is the MVX pattern defined for JavaScript-using Web applications, where X= C, P, VM?**

  1. Model View Controller (MVC): This pattern uses HTML templates for the view as HTML markup to avoid overgenerating elements. The view builds and maintains DOM elements.
     The strict sepperation of M V and C allows for parallel development, easy testing and easier maintenance.

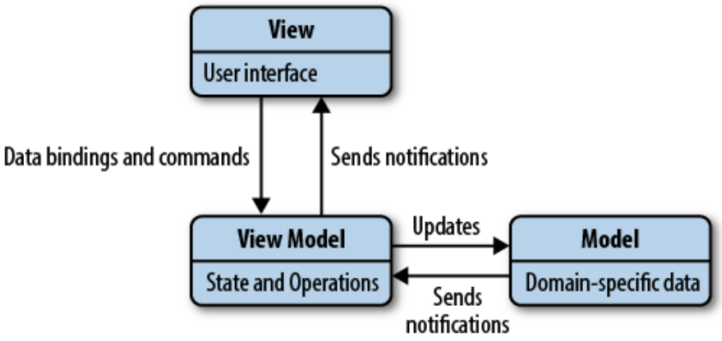


2. Model View Presenter (MVP): In this model the controller is replaced by the presenter. Decoupling of the VP is achieved by communication through interfaces. This model is most commonly implemented with little to no logic in the view, so all presentation logic is pushed to the presenter. It improves testability and modularity.



3. Model View ViewModel (MVVM): In this model the ViewModel encapsulates the the business logic with the View. It deals with the formatting of data. The ViewModel can be seen as a specialized controller doing data conversion. Validation is actually done by the models. The model facilitates the separation of the development of the graphical user interface from the development of the business logic or back-end logic so that the view is not dependent on any specific model platform

MVVM


View
User interface
Data bindings and commands
Sends notifications
View Model
State and Operations
Updates
Sends
notifications
Model
Domain-specific data