Basic aspects of Web Applications
Introduction

1. Login

Hypertext Transfer Protocol

2. Mail-address and password accepted

3. Delivery of initial page that shows e-mail inbox

Server side

Logic

Data

- Comfortable user interaction?
- Form validation?
- Dynamic client-side content?

- Transfer efficiency?
- Protocol features?
- Functional range?

- Efficiency?
- Integration of existing logic?
- Simplicity of application development?
REST model

- The Representational State Transfer (REST) is a model for distributed hypermedia systems that predominantly fits to applications in the World Wide Web.
- It is based on a client/server architecture with a stateless communication protocol.
- Every message contains all necessary context information thus neither the server nor the client has to store context data.
- In contrast to Remote Procedure Call (RPC), requests in a REST system are not directed to procedures but to resources (documents) using a generic interface with standard semantics.
- Every resource has to be available through a unique identifier.
Outline

The lecture presents selected principles and technologies of web applications:

- **Evolution of web technology**
  - Short survey of technology evolution

- **Server-side aspects**
  - Architecture of a web server

- **Client-side aspects**
  - Web browser architecture
  - (X)HTML forms and XML form technology XForms

- **Communication aspects**
  - HTTP Communication details
  - Aspects of HTTP 1.1
Development

Classical Web Applications

Client tier
- Presentation Engine
- Presentation logic

Server tier
- Presentation logic
- Business logic
- Data

Current Web Applications

Presentation Engine
- Presentation logic
- Business logic
- Data

HTTP 1.0

HTTP 1.1
Web server architecture

- A web server is a software that makes resources available through an interface that is accessible by HTTP.
- Web servers that deliver static content only have to extract a requested document from the file system and pass it to the web browser.

Due to the need for low latency time and to handle high load current web servers are structured in a hierarchical way that allows parallel request processing.
On start-up the web server creates a pool of entities for client handling and opens an interface for incoming connection requests.

1. Web browser initiates communication on the basis of TCP through this interface which is handled by a connector entity.

2. The connector entity creates a client descriptor which is the endpoint of a communication path to the client and passes it to the delegation process.

3. The delegation process chooses an available pregenerated processing entity and forwards the client descriptor to it.

4. Communication with the web browser is finally realized through HTTP.
Server-side logic

- Web applications are based on dynamic content which is generated by application logic based on e.g.:
  - Common Gateway Interface (CGI) Scripts
  - Java Servlet Container
  - Application Server that provides further services such as transaction, security or directory services and thus enables development of complex logic

- The application engine may use an (X)HTML engine to combine dynamic content with (X)HTML templates to generate resulting documents that finally are delivered to the web browser
Web browser architecture

- Early web browsers only requested documents and presented them after a rendering process to the user.
- Today web browsers are extended by further technologies as especially XML support, script engines and plug-in engines.
Web browser architecture

- **User interface:** Front-end for displaying a page to the user
- **Browser engine:** Embedded component that provides a high level interface for querying and using the rendering engine
- **Rendering engine:** Performs parsing and layout for (X)HTML documents enriched with other languages such as CSS
- **Networking:** Realises HTTP communication with the server
- **XML parser:** Parses XML content
- **Script engine:** Executes scripts embedded in (X)HTML pages
- **Display backend:** Provides drawing and windowing primitives, user interface widgets and fonts (e.g. GNU Image Manipulation Program Toolkit (GTK+))
- **Data persistence:** Stores associated data (cached pages, cookies etc.)
- **Plug-in engine:** Dynamic extension point for plug-ins e.g. an XForms processor plug-in
- Further subsystems such as e.g. an integrated Extensible Stylesheet Transformation (XSLT) processor are not shown
(X)HTML forms

Please enter mail address and password.<br/>
<form method="post" action="login">
  <input type="text" name="id"/><br/>
  <input type="password" name="password"/><br/>
  <input type="submit" value="login"/>
</form>

- Forms are the essential method of sending user input within a web application via HTTP to the web server
- Regular (X)HTML forms have some fundamental drawbacks:
  - No possibility for client-side input validation
  - Primitive data representation (strings)
  - Dynamic form elements (e.g. direct user feedback) not possible
- Script technologies such as e.g. JavaScript can be used to eliminate some of these shortcomings but out of the perspective of the form they are an external and not an integrated solution
XForms

- Client side XML form technology developed by the W3C that can be used for (dynamic) XHTML forms
- Basic idea is the separation of data and presentation
- Concept of traditional (X)HTML forms is split into three parts: model, user interface, instance data
- Namespace http://www.w3.org/2002/xforms
- Needs client-side XForms processor for execution

[Diagram]

- XHTML file
  - XForms Model
    - describes form data, constraints upon that data, and submissions
  - XForms User Interface
    - serves as a point of user interaction deployed to display and input data
  - (XML) Instance Data
    - internal tree representation of the values of a form that might be submitted as XML data
    - Submit
XForms

<head>
  ...
</head>

<xf:model>
  <xf:instance xmlns="">
    <login>
      <id/>
      <password/>
    </login>
  </xf:instance>
  <xf:submission id="form1" action="login" method="post"/>
</xf:model>

<xf:input ref="id" incremental="true">
  <xf:label>Address<br/></xf:label>
</xf:input>
<br/>

<xf:secret ref="password">
  <xf:label>Password<br/></xf:label>
</xf:secret>
<br/>

<xf:submit submission="form1">
  <xf:label>Submit</xf:label>
  <xf:output value="id">
    <xf:label>Mail address: </xf:label>
  </xf:output>
</xf:submit>

Submitted via HTTP POST to the resource '/login'

<?xml version="1.0" encoding="UTF-8"?>
<login xmlns:xf="http://www.w3.org/2002/xforms">
  <id>me@webmailer.ws</id>
  <password>pass123</password>
</login>
XForms

• Beside predefined form elements XForms specifies:
  – **Functions**: Used for specific client-side logic (calculations, validations, …)
  – **Actions**: Handle responses to special events
  – **Properties**: Can be bound to XForms data
• For validating form input XML Schema can be used

```html
<html xmlns:xf="http://www.w3.org/2002/xforms"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xf:model>…
    <person xmlns="">
      <firstname xsi:type="xsd:string"/>
      <lastname xsi:type="xsd:string"/>
      <born xsi:type="xsd:date"/>
    </person>
    <bind nodeset="person/firstname" required="true()"/>
  </xf:model>…
  <input ref="firstname">
    <message level="ephemeral" event="DOMFocusIn">
      Please input value
    </message>
  </input>
</html>
```

**XMLSchema definitions**

**Property**: Element firstname is required

**Action**: If the user sets the focus on the input field (event=DOMFocusIn) a hint message is displayed (achieved by level=ephemeral)
XForms

- To make pages that use XForms visible in web browsers without an XForms processor a server side transformation to regular (X)HTML forms is necessary
- Client should be classified for deciding which format is delivered by the response
- If any XForms properties, actions or functions were used they may be emulated by JavaScript
HTTP

- The Hypertext Transfer Protocol (version 1.1 RFC 2616) realises communication on top of TCP by the exchange of messages in a request-response manner.
- Every message is divided into a header and a body.
  
  ![HTTP Header](image)
  ![HTTP Body](image)
  Divided into several sub-headers

- The header specifies the operation that should be performed on the addressed resource and includes parameters (passed as key-value pairs).
- Resources are addressed by the request through Uniform Resource Identifiers (URI) which in this context are simple strings that identify a resource via name, location or any other characteristic:
  - Request-URI = "*" | absolute URI | absolute Path

  - Used if no resource is associated with the request
  - Path to the resource on the server combined with the server address e.g. http://www.serv.com/index.html
  - Absolute path to the resource on the server e.g. /index.html
HTTP 1.1

- In contrast to version 1.0 default behaviour of HTTP 1.1 defines establishment of a persistent connection ("connection: keep-alive") that makes communication more efficient.
HTTP 1.1 pipelining

- Pipelining is a technique that makes it possible to send further requests to the server without the need to wait for the response of previous requests.
- Responses are transmitted in the order in which the associated requests were sent.
- Not supported by all common browsers.

<table>
<thead>
<tr>
<th></th>
<th>HTTP/1.0</th>
<th>HTTP/1.1 Persistent</th>
<th>HTTP/1.1 Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packets from client to server</td>
<td>226</td>
<td>70</td>
<td>25</td>
</tr>
<tr>
<td>Packets from server to client</td>
<td>271</td>
<td>153</td>
<td>58</td>
</tr>
<tr>
<td>Total number of packets</td>
<td>497</td>
<td>223</td>
<td><strong>83</strong></td>
</tr>
</tbody>
</table>

Multiple HTTP messages can be sent with the same TCP segment.

Taken from http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html
## HTTP 1.1 request message

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Line</td>
<td>Specifies the <strong>method</strong> to be performed, the associated web resource and the HTTP version. Example: <code>GET /news/index.html HTTP/1.1</code></td>
</tr>
<tr>
<td>General Header</td>
<td>Gives general information such as the type of connection (e.g. close, keep-alive) or the date of request generation.</td>
</tr>
<tr>
<td>Request Header</td>
<td>Allows to pass information about the request and about the client to the server such as e.g. the accepted content encoding.</td>
</tr>
<tr>
<td>Entity Header</td>
<td>Delivers meta information about the body content (if available) such as e.g. the content length.</td>
</tr>
<tr>
<td>CRLF</td>
<td>Content is separated by new line (CRLF).</td>
</tr>
<tr>
<td>Message body</td>
<td>Represents the content (if available) of the request such as e.g. parameters passed to the server.</td>
</tr>
</tbody>
</table>

Order of header content elements is irrelevant.
HTTP 1.1 response message

- **Status Line**: Specifies the HTTP protocol version and a status code of request processing in numeric and textual representation; Example: HTTP/1.1 200 OK
- **General Header**: Gives general information such as the type of connection (e.g. close, keep-alive) or the date of response generation
- **Response Header**: Provides additional information about the response such as e.g. product data of the web server that produced the response
- **Entity Header**: Delivers meta information about the body content such as e.g. the content length
- **CRLF**: Content is separated by new line (CRLF)
- **Message body**: Represents the content (e.g. HTML-data of a requested web page)

Order of header content elements is irrelevant.
**HTTP 1.1 methods**

- Methods specify the resource by passing its absolute or relative Uniform Resource Identifier (URI)
- If no resource is needed an asterisk is passed
- HTTP 1.1 defines an expandable set of methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Requests for delivery of the specified resource</td>
</tr>
<tr>
<td>HEAD</td>
<td>Identical to GET except the message body of the response is empty thus the client only receives meta information of the header; can e.g. be used to determine the type of a file or its length without the need to receive the file</td>
</tr>
<tr>
<td>POST</td>
<td>Submits the message body content to the specified resource</td>
</tr>
<tr>
<td>PUT*</td>
<td>Instructs the server to make the passed content available under the specified URL</td>
</tr>
<tr>
<td>DELETE*</td>
<td>Deletes the specified resource</td>
</tr>
<tr>
<td>OPTIONS*</td>
<td>Requests communication options from server such as e.g. the methods that are supported by the server</td>
</tr>
<tr>
<td>TRACE*</td>
<td>Echoes back the send request; can be used for diagnostic purposes</td>
</tr>
<tr>
<td>CONNECT*</td>
<td>Reserved name for use by proxy servers</td>
</tr>
</tbody>
</table>

* Not defined by HTTP 1.0
HTTP 1.1 status codes

- Numeric representation of status codes consists out of a three digit value
- The first digit describes the category of the status

<table>
<thead>
<tr>
<th>Code</th>
<th>Status category</th>
<th>Description</th>
<th>Example : Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Informational</td>
<td>Request received, continuing processing</td>
<td>100 : Client should continue sending the request (e.g. extensive request)</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>The action was successfully received, understood and accepted</td>
<td>200 : Request has succeeded</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>Further action must be taken in order to complete the request</td>
<td>301 : Requested resource has been moved permanently to new URI</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>The request contains bad syntax or cannot be fulfilled</td>
<td>404 : Server has not found anything matching the Request-URI</td>
</tr>
<tr>
<td></td>
<td>(Request error)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>The server failed to fulfil an apparently valid request</td>
<td>501 : Server does not support the functionality required to fulfil the request</td>
</tr>
</tbody>
</table>
HTTP 1.1 communication details

```
HEAD /folder/indx.html HTTP/1.1
Host: www.webmailer.ws
Connection: close
Accept-Encoding: gzip
Accept: text/xml,application/xml,text/html,*/*
Accept-Language: en-gb
User-Agent: Mozilla/5.0

HTTP Status Code: HTTP/1.1 404 Not Found
Server: Apache/1.3.19 (Unix) PHP/4.3.3
Content-Type: text/html
Content-Length: 0
Date: Fri, 12 Jan 2007 14:25:15 GMT
Connection: close
```
HTTP 1.1 communication details

POST /cgi/login HTTP/1.1\r\nHost: www.webmailer.ws\r\nConnection: Keep-Alive\r\nAccept-Encoding: gzip\r\nAccept: text/xml,application/xml,text/html,*/*\r\nAccept-Language: en-gb\r\nUser-Agent: Mozilla/5.0\r\nContent-type: application/x-www-form-urlencoded\r\nContent-length: 41 \r\n\r
login=me@webmailer.ws&password=pass123

HTTP/1.1 200 OK\r\nDate: Fri, 12 Jan 2007 12:26:16 GMT\r\nServer: Apache/1.3.19 (Unix) PHP/4.3.3\r\nLast-Modified: Wed, 26 Oct 2005 14:38:41 GMT\r\nContent-Encoding: gzip\r\nContent-Length: 1922\r\nContent-Type: text/html\r\nKeep-Alive: timeout=5, max=1024\r\nConnection: Keep-Alive\r\n\r
............}yw......9....n....5.{d[I.z.....{{{|.X..K....
.....o. })..dmv...K(.

Gzip encoded HTML content
HTTP download of associated resources

After receiving the document it is analyzed for associated resources that have to be downloaded for presentation purpose.

Web browser

```html
<html>
<head>
<title>Example.com</title>
<link rel="stylesheet" href="main.css">
</head>
<body>
<img src="banner.jpg" />
<h1 class="center">Welcome!</h1>
<br/>
...
```

Web server

GET `/index.html` HTTP/1.1
HTTP/1.1 200 OK
Document delivery

GET `/main.css` HTTP/1.1
HTTP/1.1 200 OK

GET `/banner.jpg` HTTP/1.1
HTTP/1.1 200 OK
HTTP user agent detection

- HTTP header information makes adaptation of delivered content to the needs of user agents possible
- Example for user-agent strings:
  - “Mozilla (compatible; MSIE; Windows NT; .NET CLR; .NET CLR)”
  - “Mozilla [en] (PalmOS; U; WebPro)”
  - “Mozilla (compatible; MSIE; Windows CE; PPC; 240x320)”
- Analogous with e.g. content language adaptation (“Accept-Language”)
Session handling

- A session context establishes a state for web applications
- During a session the browser can store information about a user in a special data structure
- For identification purpose the web server sends a session id to the web browser which is included into later requests thus allowing the server to associate the right session data structure with the originator of a request
- Famous technique for session id exchange is the use of cookies which are small pieces of data stored on client-side
Conclusion

Web browser

- JavaScript engine
- XForms processor
- Rendering engine
- Browser engine
- ...

Web server

- Establish TCP connection
- GET XHTML document
- GET associated CSS file
- Close TCP connection

Processing entity

- HTTP Header
- HTTP Body

Connector entity

- Client handling is delegated to

Server side logic

Processing entity

Processing entity
References

Links at W3C:

XForms home http://www.w3.org/MarkUp/Forms/

RFCs:


Details about the REST architecture:


Current development on HTTP carried out by Google:

SPDY project http://dev.chromium.org/spdy