Web application development

Prof. Dr. Ralf Lämmel
University of Koblenz-Landau
Faculty of Computer Science
Software Languages Team
Introduction
Web application

From Wikipedia, the free encyclopedia

A web application or web app is any program that runs in a web browser. It is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a web browser to render the application.[1][2][3]

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a [[client (computing)|client]] to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.

Horde groupware is an open-source web application.
A Web application (WA) is a software system based on technologies and standards of the WWW that provides Web-specific resources such as content and services and is accessed via a user interface -- the Web browser.

Principle architecture of WA

- It is typically a Client-server architecture.
- End users access a client (in a Web browser).
- Services are implemented on a (Web) server.
- Services are materialized as documents.
- Client & server communicate via protocols over the internet.

supports rendering and interaction with HTML pages

delivers Web pages over HTTP

describe message formats and rules for exchanging messages
Client/server architecture (as opposed to “standalone”)

- Client-side scripting
- **Server-side scripting**
- Model view controller (MVC)
- Remote Method Invocation (RMI)
- Messaging
- **RESTful web services**
- Pull and push
- …

Concerns of a WA architecture

What if the client does not need a server all the time?

What if this is a native app?

What's the request?

Is that a complete view, a delta, or what?

What if this is a Web service?

What if the server is often unavailable?
Demo

A Web application for a simple human-resources management system.

http://101companies.org/wiki/Contribution:html5local

Not much internals are discussed at this point. Appearance and architecture options are discussed.
$ pwd
/Users/laemmel/github/101companies/101repo/contributions/html5local

$ ls
README.md      department.html   employeeModel.js
company.html   departmentController.js employeeView.js
company.js     departmentModel.js  head.js
companyController.js departmentView.js  index.html
companyModel.js employee.html  style.css
companyView.js  employeeController.js

$ open index.html
http://101companies.org/wiki/Contribution:html5local
Web application framework

From Wikipedia, the free encyclopedia

A web application framework (WAF) is a software framework that is designed to support the development of dynamic websites, web applications, web services and web resources. The framework aims to alleviate the overhead associated with common activities performed in web development. For example, many frameworks provide libraries for database access, templating frameworks and session management, and they often promote code reuse.[1] For a comparison of concrete web application frameworks, see Comparison of web application frameworks.

1 History
2 Types of framework architectures
   2.1 Model–view–controller (MVC)
      2.1.1 Push-based vs. pull-based
   2.2 Three-tier organization
3 Framework applications
   3.1 General-purpose website frameworks
   3.2 Discussion forums, wikis and weblogs
   3.3 Organizational portals
   3.4 Content management systems (CMS)
4 Features
   4.1 Web template system
   4.2 Caching
   4.3 Security
   4.4 Database access, mapping and configuration
   4.5 Scaffolding
   4.6 URL mapping
   4.7 Ajax
   4.8 Web services
   4.9 Web resources
5 See also
6 References

© 2016 Ralf Lämmel
http://101companies.org/wiki/Theme:Web_programming

Theme: Web programming

Headline

Demonstrations of web programming

Metadata

- Contribution: happystack  memberOf  this
- Contribution: html5ajax  memberOf  this
- Contribution: html5local  memberOf  this
- Contribution: jsf  memberOf  this
- Contribution: pyjamas  memberOf  this
- Contribution: rubyonrails  memberOf  this
- Contribution: seaside  memberOf  this
- Contribution: silverlight  memberOf  this
- Contribution: strutsAnnotation  memberOf  this
- Contribution: zend  memberOf  this

Find demos of different WAFs on 101.
What are the characteristics of Web applications and their development?
### Rich functionality

This image depicts a Gmail inbox with various emails listed. The inbox is organized with different categories and features.

- **Inbox (9,148)**: The total number of emails in the inbox.
- **Sent Mail**: The emails that have been sent.
- **Drafts (247)**: The emails that are in draft form.
- **Circles**: Contacts and groups to which emails are addressed.

### Emails in the Inbox

<table>
<thead>
<tr>
<th>Subject</th>
<th>Sender</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re: Debeka-Seminar am 14.9.7.9.</td>
<td><a href="mailto:rlaemmeli@gmail.com">rlaemmeli@gmail.com</a></td>
<td>Sep 3</td>
</tr>
<tr>
<td>Wie geht es Dir? - Ganztagesprogramm für L</td>
<td></td>
<td>Aug 31</td>
</tr>
<tr>
<td>NoSQL - plan - for the Debeka course. The ic</td>
<td></td>
<td>Aug 29</td>
</tr>
<tr>
<td>My meeting with Thomas tomorrow - plan - w</td>
<td></td>
<td>Aug 29</td>
</tr>
<tr>
<td>NoSQL - on the Debeka course &gt; materials. l</td>
<td></td>
<td>Aug 29</td>
</tr>
<tr>
<td>Re: Fwd: Fwd: Sponsoring der Sommerschul</td>
<td></td>
<td>Jul 13</td>
</tr>
<tr>
<td>Debeka als Sponsor des akademischen IT E</td>
<td></td>
<td>Jul 10</td>
</tr>
<tr>
<td>Sponsoring der Sommerschule Software Tec</td>
<td></td>
<td>Jul 4</td>
</tr>
<tr>
<td>Fwd: Unterlagen Selbstevaluation Ul</td>
<td></td>
<td>May 14</td>
</tr>
<tr>
<td>Debeka-Semianare - 2012/5/3 Jürgen Ebert</td>
<td></td>
<td>May 3</td>
</tr>
<tr>
<td>Termine für die Debeka-Seminar -</td>
<td></td>
<td>May 3</td>
</tr>
<tr>
<td>You've just been ousted as the mayor of Deb</td>
<td></td>
<td>Apr 12</td>
</tr>
</tbody>
</table>
### Wählen Sie eine Verbindung

#### Ihre Hinfahrtmöglichkeiten

<table>
<thead>
<tr>
<th>Bahnhof/Haltepunkt</th>
<th>Datum</th>
<th>Zeiten</th>
<th>Dauer</th>
<th>Umst.</th>
<th>Produkte</th>
<th>Preis für alle Reisenden*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frankfurt(M) Flughafen Fernbf</td>
<td>Do, 06.09.12</td>
<td>an 15:35</td>
<td>1:51</td>
<td>1</td>
<td>IC, BUS</td>
<td>26,00 EUR (Teilpreis)</td>
</tr>
<tr>
<td>Metternich Trifter Weg B258, Koblenz</td>
<td>Do, 06.09.12</td>
<td>ab 13:58</td>
<td></td>
<td></td>
<td></td>
<td>Normalpreis</td>
</tr>
</tbody>
</table>

#### Weitere Informationen

- Hauptbahnhof (ZOB), Koblenz
- Fußweg 7 Min.
- Bus 350
  - Bus Richtung: Obertor, Mayen, Rhein-Mosel-Bus

---

© 2016 Ralf Lämmel
Secure operation
Responsive user interface
Support for different browsers

http://www.unixmen.com/browser-war-heats-up-again-as-chrome-unseats-ie-for-may-2012/
public class HelloWorldApplication {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Recommended resources: ‘HTML5 for the Java developer’
http://www.youtube.com/watch?v=I-GD4ro9fh4
http://www.youtube.com/watch?v=2msljs0Qcls
Anything else you want to emphasize as a characteristic?
Basics of web application development
The HTTP protocol

$ curl www.google.com/

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.de/?gfe_rd=cr&amp;ei=KtEQV62kHK2o8we0xI-YCA">here</A>.
</BODY></HTML>
Static web pages


Hello World!

<html>
<head>
<title>Hello World WebApp</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
Dynamic web pages


We use the Perl language for what it matters.

Computation of HTML markup

clockhost/helloWorld.cgi

```perl
#!/usr/bin/perl
print "Content-type: text/html\n\n"
print <<HTML;
<html>
<head>
<title>Hello World WebApp</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
exit;
```
**CGI - Common Gateway Interface**

http://www.citycat.ru/doc/CGI/overview/overview.html

![Diagram](image)
Access to server-side resources

Perl code with file access on server side

```perl
#!/usr/bin/perl
open FILE, "demo.txt";
my $lines = <FILE>;
print "Content-type: text/html\n\n";
print <<HTML;
<html>
<head>
<title>Spying WebApp</title>
</head>
<body>
<p>
HTML
print $lines;
print <<HTML;
</p>
</body>
</html>
HTML
exit;
```

localhost/showFile.cgi
HTTP request parameters

```perl
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);

# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET") {
    $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($key, $val) = split(/=/, $pair);
    $val =~ tr/+/%/;
    $val =~ s/%(/pack("C", hex($1))/e;
    %FORM{$key} = $val;
}
$name = %FORM{name};

print "Content-type:text/html\n\n\n";
pod print "<html>";
pod print "<head>";
pod print "<title>HelloName WebApp</title>";
pod print "</head>";
pod print "<body>";
pod print "<h2>Hello $name!</h2>";
pod print "</body>";
pod print "</html>";
1;
```
HTML forms

http://en.wikipedia.org/wiki/Form_(web)

```html
<FORM action="helloName.cgi" method="GET">
Name: <input type="text" name="name">
<br>
<input type="submit" value="Submit">
</FORM>
```
Code embedded into HTML

localhost/helloWorld.php

```html
<html>
<head>
  <title>HelloWorld WebApp</title>
</head>
<body>
  <?php echo '<p>Hello World!</p>'; ?>
</body>
</html>
```

HTML with code

We use the PHP language for what it matters.
Server-side Scripting
Local Computer
http://www.example.com/login.php

Internet

Web server
The web server is processing the request
PHP interpreter
HARD DISK
MySQL db

What is scripting?

- Embed simple code in HTML pages.
- The HTML pages use the code to decide what elements and data to display.
- Existing APIs can be invoked to compute information for inclusion in the web page.

Server-side scripting is a technique used in website design which involves embedding scripts in an HTML source code which results in a user's (client's) request to the server website being handled by a script running server-side before the server responds to the client's request.

Key components of Server-side scripting

- **Web Server** -- makes available documents and handles requests.
- **Server-side processor** -- interprets scripts and generates HTML
Web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver Web content that can be accessed through the Internet.

Apache web server

- Apache http server project
- http://httpd.apache.org
- Apache foundation started to support the web server project, but eventually extended to a multitude of other projects.
Configuration of Apache to work with PHP:

Configure directory `DocumentRoot`

This is the directory from which the Web server will serve the documents (html, php, etc). That is, by default, requests are resolved to this directory.

Configuration entry:

`DocumentRoot "/Library/WebServer/Documents"`
What happens when we open a PHP file in the browser?

Non-processed HTML is shown unless PHP is enabled. 

`php5_module` should be enabled to handle `*.php`

LoadModule php5_module libexec/apache2/libphp5.so
Remember --
Code embedded into HTML

localhost/helloWorld.php

<html>
<head>
  <title>HelloWorld WebApp</title>
</head>
<body>
<?php echo '<p>Hello World!</p>'; ?>
</body>
</html>

HTML with PHP
PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. It is one of the first developed server-side scripting languages to be embedded into an HTML source document rather than calling an external file to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page.

PHP Flow

HTML is substituted for PHP code

Browser

Internet

Web Server

PHP Pre-processor

Raw .php source

Request for .php file

.php source file
Adding interaction

Welcome Andrei!
You are 27 years old.

<html>
<body>
Welcome Andrei!<br/>
You are 27 years old.
</body>
</html>
Underlying source

**intro.html**

```html
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

**welcome.php**

```html
<html>
<body>
Welcome <?php echo $_POST["fname"];?>!<br />
You are <?php echo $_POST["age"];?> years old.
</body>
</html>
```
<?php

$lines = preg_split('~\s*\[\r\n\]+\s*~', file_get_contents('prod.txt'));

foreach($lines as $i => $line) {
    $pairs = explode(';', $line);
    foreach($pairs as $pair) {
        list($column, $value) = explode('=', $pair, 2);
        $columns[$column] = true;
        $rows[$i][$column] = $value;
    }
}
$columns = array_keys($columns);
echo '<table><thead><tr>';
foreach($columns as $column) {
    echo '<th>'.$column.'</th>';
}
echo '</tr></thead><tbody>';
foreach($rows as $row) {
    echo '<tr>';
    foreach($columns as $column) {
        echo '<td>'.$row[$column].'</td>.viewmodel/leaf.html';
    }
    echo '</tr>.viewmodel/leaf.html';
}
echo '</tbody></table>.viewmodel/leaf.html';
?>

Scripting example: Compute HTML table from CSV file

difficult to read and difficult to maintain
DEMO?

http://101companies.org/wiki/Contribution:php

- Show the implementation in action.
- Explain the architecture of the implementation.
- Show details of server-side scripting.
- Cover accidental complexity such as DB access.
Model-View-Controller (MVC) in Web Application Development
MVC - a classic definition

• The *Model* is the application object
• The *View* is its screen presentation
• The *Controller* defines the way the user interface reacts to user input
Model–View–Controller (MVC) is a computer software design pattern that separates the representation of information from the user's interaction with it. The model consists of application data and business rules, and the controller mediates input, converting it to commands for the model or view. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible.

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
The Model-View-Controller Architecture

here: server side

1. Browser sends request
2. Controller interacts with model
3. Controller invokes view
4. View renders next browser screen

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
The (Ruby on) Rails MVC

2. Routing finds Store controller
3. Controller interacts with model
4. Controller invokes view
5. View renders next browser screen
Model

- Maps to a table in a database. By convention, a model named Company will map to the database table companies, and the model will have a filename company.rb within app/models folder.

```ruby
class Company < ActiveRecord::Base
  validates :name, :presence => true
  has_many :departments
end

class Department < ActiveRecord::Base
  belongs_to :company
  belongs_to :department
  has_many :departments
  has_many :employees
end

class Employee < ActiveRecord::Base
  belongs_to :department
end
```
Controller

- Responds to external requests from the web server to the application, and responds to the external request by determining which view file to render.
Controller (II)

- Handles *people-friendly* URLs extremely well.
- Manages *caching*, which can give applications orders-of-magnitude performance boosts.
- Manages *sessions*, giving users the impression of ongoing interaction with our applications.
In the default configuration of Rails is an `erb` file. It is typically converted to output HTML at run-time.

Ruby embedded instead of PHP
Ruby

Ruby is a dynamic, reflective, general-purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. It was also influenced by Eiffel and Lisp.

Philosophy ( Rails )

- **DRY** – “Don’t Repeat Yourself” – suggests that writing the same code over and over again is a bad thing.

- **Convention Over Configuration** – means that Django makes assumptions about what you want to do and how you’re going to do it, rather than requiring you to specify every little thing through endless configuration files.

- **REST is the best pattern** for web applications – organizing your application around resources and standard HTTP verbs is the fastest way to go.

Rails **enforces** the use of MVC as a pattern.
Example

101 implementation: rubyonrails

- Look into internals this time.
- Consider details of MVC in this implementation.
- Understand bits of Ruby on Rails philosophy.
Companies::Application.routes.draw do
  resources :employees http://exampe.com/employees/
  resources :departments do
    resources :departments http://exampe.com/departments/
    resources :employees http://exampe.com/departments/:id/employees
  end
  resources :companies do
    resources :departments http://exampe.com/companies/:id/departments
  end
  get "home/index" http://exampe.com/ (HTTP GET only)
rake routes

GET /employees(.:format) {:action=>"index", :controller=>"employees"}
POST /employees(.:format) {:action=>"create", :controller=>"employees"}
GET /employees/new(.:format) {:action=>"new", :controller=>"employees"}
GET /employees/:id/edit(.:format) {:action=>"edit", :controller=>"employees"}
GET /employees/:id(.:format) {:action=>"show", :controller=>"employees"}
PUT /employees/:id(.:format) {:action=>"update", :controller=>"employees"}
DELETE /employees/:id(.:format) {:action=>"destroy", :controller=>"employees"}
departments_controller.rb

# GET /departments/1
# GET /departments/1.json
def show
  @department = Department.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render :json => @department }
  end
end
<div class="headline"><h2>101companies Ruby on Rails Web App</h2></div>
<div class="content">
<p id="notice"><%= notice %></p>
<div class="attr">
<p>
<b>Name:</b>
<br />
<%= @department.name %>
</p>
</div>
<hr>
<div class="attr">
<p>
<b>Manager:</b>
<br />
<% @department.employees.each do |employee| %>
<p>
<%= if (employee.isManager?)
    link_to employee.name, employee_path(employee)
  end
%>
</p>
<% end %>
</div>

.....
JavaScript and AJAX
JavaScript (sometimes abbreviated JS) is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.


Standardized JavaScript = ECMAScript

http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf
JavaScript is a very important language :-)
Warning / Disclaimer

• JS might look very unnatural at first.
• It is essentially LISP with C-like syntax:
  ‣ Very powerful
  ‣ Very flexible
  ‣ Complicated due to some language design decisions

We will compare JS with Java, erratically.
A JavaScript joke

```javascript
[] + {}
"[object Object]"

{} + []
0
```
Some ‘good’ part of JS:

```javascript
var add = function (a, b) {
    return a + b;
}
var y = add(2,3)
```

Define and apply a function.

Some ‘bad’ part of JS:

```
[] + {} 
"[object Object]" 
{} + [] 
0
```
The notion of prototype

Object.create = function (o) {
  var F = function () {};
  F.prototype = o;
}

meganalysis = {
  "name": "Meganalysis"
};
meganalysis2 = Object.create(meganalysis);

Everything is an Object, like in Smalltalk :-)

meganalysis2 = Object.create(meganalysis);

```javascript

```

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
The Method Invocation Pattern

var company = {
    total: 1000,
    increment: function(val) {
        this.total += val;
    }
}

company.increment(100);
console.log(company.total);

'This'/local scope: "company" object

company - object
total - public property
increment - public method

Think in Java: no classes???
The Function Invocation Pattern

```
add = function (a,b) {
    console.log(this);
    return a+b;
}
x = add(2,3);
```

‘this’/local scope: a global object

JS runs in the web browser. The global object is **Window**.
Constructor Invocation Pattern

// Create a constructor function for employees.
var Employee = function (name) {
    this.name = name;
};

// Give all employees a public method.
Employee.prototype.get_name = function () {
    return this.name;
};

// Make an instance of Employee.
var ralf = new Employee('Ralf');

name = ralf.get_name();
console.log(name);

Think in Java:
constructor invocation
var Employee = function (name) {
    this.name = name;
};
Employee.prototype.get_name = function () {
    return this.name;
};
var ralf = new Employee('Ralf');
ralf.name = "Andrei"
name = ralf.get_name();

Q: What is the value of the name?
A: Think in Java: We need to ‘hide’ properties.
```javascript
var ralf = (function () {
    var name = "Ralf";
    return {
        getName: function () {
            return name;
        }
    };
})();
```

```
name: function () { return name; }  
```

```
getName: function () { return name; }  
```

```
username: "Ralf"
```

"name" is hidden
Inheritance

Think in Java: toString is overridden.

var Person = function (name) {
    this.name = name;
    this.isHuman = true;
};
var Employee = function (name) {
    this.name = name;
};
Person.prototype.isHuman = function(){
    return this.isHuman;
};
Person.prototype.toString = function(){
    return '['+this.name+']';
};

// Here's where the inheritance occurs
Employee.prototype = new Person();

// Otherwise instances of Employee would have a constructor of Person
Employee.prototype.constructor = Employee;

Employee.prototype.toString = function(){
    return '['+this.name+']';
};
JS is not the ‘best’ OO language. Why should I care?

Because it’s the language in the Web browser:

Client-side scripting
Front-end development
Interactive web applications

} = JavaScript
HTML Document
Object Model

<html>
<head>
  <title>My title</title>
</head>
<body>
  <a href="#">My Link</a>
  <h1>My header</h1>
</body>
</html>
HTML DOM Event Handling

http://jsfiddle.net/DrGigabit/aQctY/1/
```html
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My header</h1>
    <button id="createButton">Click me</button>
  </body>
</html>
```

```javascript
var button = document.getElementById("createButton");
button.addEventListener("click", function() {
  alert("Click!");
}, false);
```

asynchronously = interactive UI
Function as arguments (callbacks)

// Define a function on two number args and a function arg.
function randomBlock(arg1, arg2, callback) {
  // Generate a random number between arg1 and arg2.
  var rnd = Math.ceil(Math.random() * (arg2 - arg1) + arg1);
  // Pass the result to the function argument.
  callback(rnd);
}

// Apply randomBlock to an anonymous function.
randomBlock(5, 15, function(arg) {
  // This anonymous function will be applied later.
  console.log("Callback called with arg = "+ arg);
});
Motivating scenario: Asynchronous input/output

Make a request *synchronously*

```javascript
request = prepare_the_request(...);
response = send_request_synchronously(request);
zzzzZZZZZZzzz <-- Waiting time
display(response);
```

Make a request *asynchronously*

```javascript
request = prepare_the_request(....);
send_request_asynchronously(request, function (response) {
    display(response); <-- When ready
});
doSomethingElse();
```
jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.
**jQuery**

```javascript
var button = $( '#createButton' );
button.click(function(){
    alert('clicked');
});
```

**plain JS**

```javascript
var button = document.getElementById( 'createButton' );
button.addEventListener( 'click', function() {
    alert( 'Click!' );
}, false );
```

```javascript
$( '#createButton' ) === document.getElementById( 'createButton' );
```
Another DOM Manipulation

h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>

+ 
$(
'.inner').append('<p>Test</p>');

= 

<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    Hello
    <p>Test</p>
  </div>
  <div class="inner">
    Goodbye
    <p>Test</p>
  </div>
</div>
Asynchronous JavaScript and XML (AJAX)

Motivation

We know how to do client-side programming in JavaScript. How do we interact with the server?
What’s AJAX?

• AJAX = Asynchronous JavaScript and XML
• Make asynchronous requests to the server.
• Receive response eventually through callback.
• Support based on XMLHttpRequest object.
• ‘No page refresh’
AJAX example: *loading company data from the server*

```javascript
var company = {};

company.response;

company.loadData = function() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'company.xml', true);
    xhr.onload = function(e) {
        if (this.status == 200) {
            company.response = xhr.responseXML;
            controller.loadInner();
        }
    };
    xhr.send();
}
```

- Prepare request object
- Point to resource
- Register response handler
- Send actual request

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
DEMO

101implementation:html5XMLHttpRequest

Show XHR (XmlHttpRequest) in a 101implementation.
Cascading Style Sheets (CSS)
What is CSS?

- Cascading Style Sheets (CSS) is a language for specifying how documents are presented to users.

- A document is a collection of information that is structured using a markup language.
Without CSS

<head>
<title>CSS Example</title>
</head>
<body>
<H1>Very important</H1>
42
<H2>Less important</H2>
88
</body>
With CSS

<head>
<title>CSS Example</title>
<style type="text/css">
  H1 { font-size: x-large; color: red }
  H2 { font-size: large; color: blue }
</style>
</head>
<body>
<H1>Very important</H1>
42
<H2>Less important</H2>
88
</body>
Why do we need CSS?

• Provides a powerful and flexible way to control the details of displaying web documents.

• HTML is more concerned about the content; CSS is used to impose a particular style on the document.

• Named ‘cascading style sheets’ because they can be defined at three different levels to specify the style of a document:

  Inline (per element), document level, external.
DEMO

Vary CSS for the shown HTML markup.
http://jsfiddle.net/rlaemmel/eVbY7/
Format of style rules

Remember HTML DOM?

selector { property1: value1; property2: value2 }

H1 { font-size: x-large; color: red }
Rules with an *id* selector

CSS

```
name { text-indent: 3em }
```

HTML

```
<p id="name">Andrei</p>
```
Rules with an \textit{class} selector

\textbf{CSS}

```
.center {text-align:center;}
```

\textbf{HTML}

```
<p class="center">Some text</p>
```

Specify a class of style which can be choosen.
Rules with ‘class’ in HTML5

http://jsfiddle.net/rlaemmel/MgPxj/3/
Layout in HTML5

http://jsfiddle.net/DrGigabit/ZbV3F/2/

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
Web application development with **Django**

We use the Polls app as the running example.

Find the code here:  
https://github.com/rlaemmel/mysite

A deployment *MAY* be available here:  
http://rlaemmel.pythonanywhere.com/polls/
Python Web frameworks

- Django
- web2py
- Flask
- Bottle
A low-level view on the Polls app

A database

The model

The view

Python code, HTML, and templates

Templates for views
Issues

• What is the schema underlying the database image?
• What are the roles of the different python scripts?
• How do code and database relate to each other?
• What technologies are used by the app?
• Aren’t the HTML files using non-HTML constructs?
Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- What technologies are used by the app?
- Aren’t the HTML files using non-HTML constructs?
A command to request the DB schema

~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py sql polls

Response by Django

```
BEGIN;
CREATE TABLE "polls_poll" (  
"id" integer NOT NULL PRIMARY KEY,
"question" varchar(200) NOT NULL,
"pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (  
"id" integer NOT NULL PRIMARY KEY,
"poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
"choice" varchar(200) NOT NULL,
"votes" integer NOT NULL
)
;
COMMIT;
```
Command line language for Django administration

• `python manage.py sql polls`
• `python manage.py syncdb`
• (There exist more such administrative commands.)
Issues

- What is the schema underlying the database image?
- **What are the roles of the different python scripts?**
- How do code and database relate to each other?
- What technologies are used by the app?
- Aren’t the HTML files using non-HTML constructs?
Concepts behind the many Python scripts

- mysite
  - __init__.py  Initialization
  - manage.py  Administration
- polls
  - __init__.py  Initialization
  - admin.py  View
  - models.py  Model
  - tests.py  Testing
  - views.py  View
- settings.py  Configuration
- urls.py  Router (Routing)
Concepts

- **Model**: the data / business logics part MVC
- **View**: the user interface part of MVC
- **Router**: a form of controller (part of MVC)
- **Configuration**: configuration of a component or a system
- **Initialization**: initialization of a component or a system
- **Administration**: administration of a system
- **Testing**: test of an artifact or a system
The *model*

```python
from django.db import models
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __unicode__(self):
        return self.question
    def was_published_today(self):
        return self.pub_date.date() == datetime.datetime.date().today()
    was_published_today.short_description = 'Published today?'

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

Don’t bother about details: these are Python (Django) classes for the business data of the Polls app.
The view for „end users“

```python
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    t = loader.get_template('polls/index.html')
    c = Context({
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(t.render(c))

def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/detail.html', {'poll': p},
                              context_instance=RequestContext(request))

def results(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/results.html', {'poll': p})
```

Don’t bother about details, but a typical view loads or saves data, and renders data as HTML via a template.
The view for „admins“ according to Django

```python
class ChoiceInline(admin.TabularInline):
    # Another more spacious option
    # class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question', 'pub_date', 'was_published_today')
    list_filter = ['pub_date']
    search_fields = ['question']
    date_hierarchy = 'pub_date'

admin.site.register(Poll, PollAdmin)
```

These views are standardized by Django: they allow us to do basic data management for polls and choices.
Routing
(A router maps URLs to views)

```python
from django.conf.urls.defaults import patterns, include, url
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^polls/$', 'mysite.polls.views.index'),
    url(r'^polls/(?P<poll_id>\d+)/$', 'mysite.polls.views.detail'),
    url(r'^polls/(?P<poll_id>\d+)/results/$', 'mysite.polls.views.results'),
    url(r'^polls/(?P<poll_id>\d+)/vote/$', 'mysite.polls.views.vote'),
    url(r'^admin/', include(admin.site.urls)),
)
```

Regular expression for URLs with parameters

Python function for a particular view
Issues

• What is the schema underlying the database image?
• What are the roles of the different python scripts?
• **How do code and database relate to each other?**
• What technologies are used by the app?
• Aren’t the HTML files using non-HTML constructs?
How do code (model) and database relate to each other?

```sql
CREATE TABLE "polls_poll" ( ...
);
CREATE TABLE "polls_choice" ( ...
)
```
Request of database sync via CLI

~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py syncdb

Creating tables ...
Creating table auth_permission

... 
Creating table django_admin_log
Creating table polls_poll
Creating table polls_choice

You just installed Django's auth system, which means you don't have any superusers defined. Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'rlaemmel'): rlaemmel

... 
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.

Response by Django

This is basically just informative text produced by the admin functionality to report on database changes.
Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- **What technologies are used by the app?**
- Aren’t the HTML files using non-HTML constructs?
Referenced python modules
(We use “megamodeling” notation here.)

- Runtime < Technology
- Template processor < Technology
- Protocol < Concept
- PythonRuntime : Runtime
- os : Library
- datetime : Library
- Django.db : Library
- Django.test : Library
- Django.template : Template processor
- Django.http : Library
- Database access : Concept
- Testing : Concept
- Template processing : Concept

- HTTP : Protocol
- webapp uses os
- webapp uses datetime
- webapp uses Django.db
- webapp uses Django.test
- webapp uses Django.template
- webapp uses Django.http
- datetime partOf PythonRuntime
- os partOf PythonRuntime
- Django.db facilitates Database access
- Django.test facilitates Testing
- Django.http facilitates HTTP

This naming convention introduces **parts**.
Issues

• What is the schema underlying the database image?
• What are the roles of the different python scripts?
• How do code and database relate to each other?
• What technologies are used by the app?

• Aren’t the HTML files using non-HTML constructs?
The template for the *index* view

```html
{% if latest_poll_list %}
  <ul>
    {% for poll in latest_poll_list %}
      <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
  </ul>
{% else %}
  <p>No polls are available.</p>
{% endif %}
```
The template for the *detail* view

```html
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="/polls/{{ poll.id }}/vote/" method="post">
{% csrf_token %}
{% for choice in poll.choice_set.all %}
  <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
  <label for="choice{{ forloop.counter }}">{{ choice.choice }}</label>
<br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```
The template for the results view

```html
<h1>{{ poll.question }}</h1>

<ul>
{% for choice in poll.choice_set.all %}
  <li>{{ choice.choice }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="/polls/{{ poll.id }}">Vote again?</a>
```
A language for *templates*

* We designate a language **Django.Templ ⊃ HTML**.
* Django.Templ offers extra constructs like this:
  * Python expressions {{ … }} evaluating to HTML
  * Loops over Python data to return HTML
RESTful web services

(Use HTTP+REST in a client/server architecture.)
Aspects of a RESTful web service API

- Standard HTTP methods: GET, PUT, DELETE, POST
- A base URI such as http://example.com/resources/
- Links to reference resources such as http://.../resources/1
- An internet media type such as JSON or XML
- Constraints
  - „Client-server“: data on server, UI on client, …
  - „Stateless“: no client context being stored on server
  - „Cacheable“: responses define themselves as cacheable

REST is an architectural style — not a protocol
Resources and Resource Identifiers

Examples of identifiers

- The key abstraction of information in REST is a resource.
- Each resource has a resource identifier.
  - http://example.com/customers/1234
  - http://example.com/products/4554
  - http://example.com/processes/salary-increase-234
Hypertext Transfer Protocol


- **GET** Request representation for resource
- **HEAD** Like GET but without response body
- **PUT** Upload representation for resource
- **POST** Submit data for resource
- **OPTIONS** Query for available methods
- **CONNECT** Facilitate SSL-encrypted communication
- **DELETE** Delete specified resource
- **TRACE** Return request as it arrived at server
- **PATCH** Partial modification of resource
RESTful Web Service HTTP methods

- **Collection** URI, such as http://example.com/companies/
- **GET**: List the URIs and perhaps other details of the collection's members
- **PUT**: Replace the entire collection with another collection.
- **POST**: Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.
- **DELETE**: Delete the entire collection.
RESTful Web Service HTTP methods

- **Element** URI, such as http://example.com/companies/32
- **GET:** Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
- **PUT:** Replace the addressed member of the collection, or if it doesn't exist, **create** it.
- **POST:** Treat the addressed member as a collection in its own right and **create** a new entry in it.
- **DELETE:** **Delete** the addressed member of the collection.
An API for the Polls app

- index: Return all polls
  
  ```json
  {"1": "What's the coolest language?“, 
  "2": "Where do you go tomorrow?"}
  ```

- detail: Return choices with numbers of votes for a poll
  
  ```json
  {"1": {"votes": 5, "choice": „Cobol"}, 
  "2": {"votes": 90, "choice": "Python"}, …}
  ```

- vote: Count a vote towards a choice of a poll

---

https://github.com/rlaemmel/mysite/blob/master/polls/api.py
urlpatterns = patterns(''
    #
    # An experiment with a RESTful API
    #
    url(r'^api/polls/$', 'mysite.polls.api.index'),
    url(r'^api/polls/(\d+)/$', 'mysite.polls.api.detail'),
    url(r'^api/vote/$', 'mysite.polls.api.vote'),
    #
    # What follows is from the basic webapp
    #
    ...)

© 2016 Ralf Lämmel
# Return all polls in JSON

def index(request):
    poll_list = Poll.objects.all()
    poll_dict = dict()
    for p in poll_list:
        poll_dict[p.id] = p.question
    content = json.dumps(poll_dict)
    return http.HttpResponse(content, content_type='application/json')
# Return choices with numbers of votes for a poll in JSON

def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    choice_dict = dict()
    for c in p.choice_set.all():
        choice_dict[c.id] = dict()
        choice_dict[c.id]["choice"] = c.choice
        choice_dict[c.id]["votes"] = c.votes
    content = json.dumps(choice_dict)
    return http.HttpResponse(content,
        content_type='application/json')
# Count a vote towards a choice of a poll; return votes count in JSON

def vote(request):
    dict_content = json.loads(request.raw_post_data)
    poll_id = int(dict_content['poll'])
    choice_id = int(dict_content['choice'])
    try:
        p = Poll.objects.get(pk=poll_id)
        c = p.choice_set.get(pk=choice_id)
    except (KeyError, Choice.DoesNotExist):
        raise Http404()
    c.votes += 1
    c.save()
    return http.HttpResponse(json.dumps(c.votes),
                              content_type='application/json')
A client using the API of the Django-based Polls app at https://github.com/rlaemmel/mysite

There are these scripts for accessing the API:

- index.py: list all polls
- detail.py: list details for a given poll
- vote.py: vote on a given choice of a given poll
- bot.py: continuously vote to maintain a given choice of a given poll as the most popular one

This example is used in a software development lecture to explain basics of REST.
Voter fraud

$ python detail.py 1
{5: {u'votes': 1, u'choice': u'Nowhere'}, 6: {u'votes': 10, u'choice': u'Microsoft'}, 7: {u'votes': 11, u'choice': u'Google'}}

$ python bot.py 1 2
Starting bot for poll 1 and choice 2 ...
The question for poll ID 1 is "What's the coolest language?". The choice for choice ID 2 is "Python" with current popularity 90. Popularity challenged by "Haskell" with current popularity 91. Vote!
The resulting popularity of "Python" is 91. Popularity challenged by "Haskell" with current popularity 91. Vote!
The resulting popularity of "Python" is 92. Popularity maintained.
Popularity maintained.
Popularity maintained.
^C
Finishing
The End