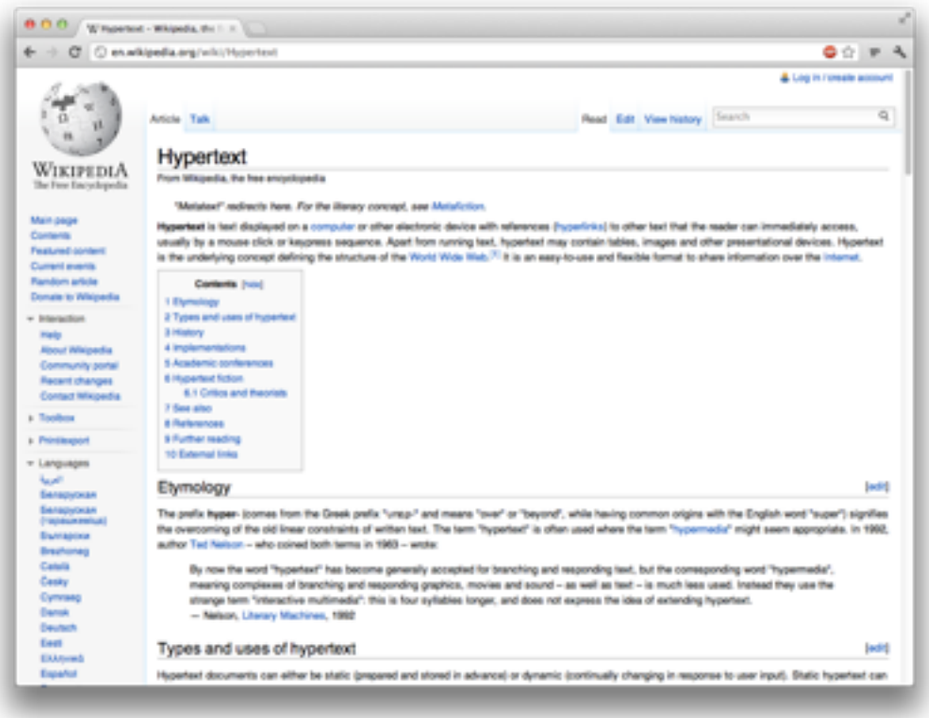
Pjax and the next generation of server-side Web frameworks

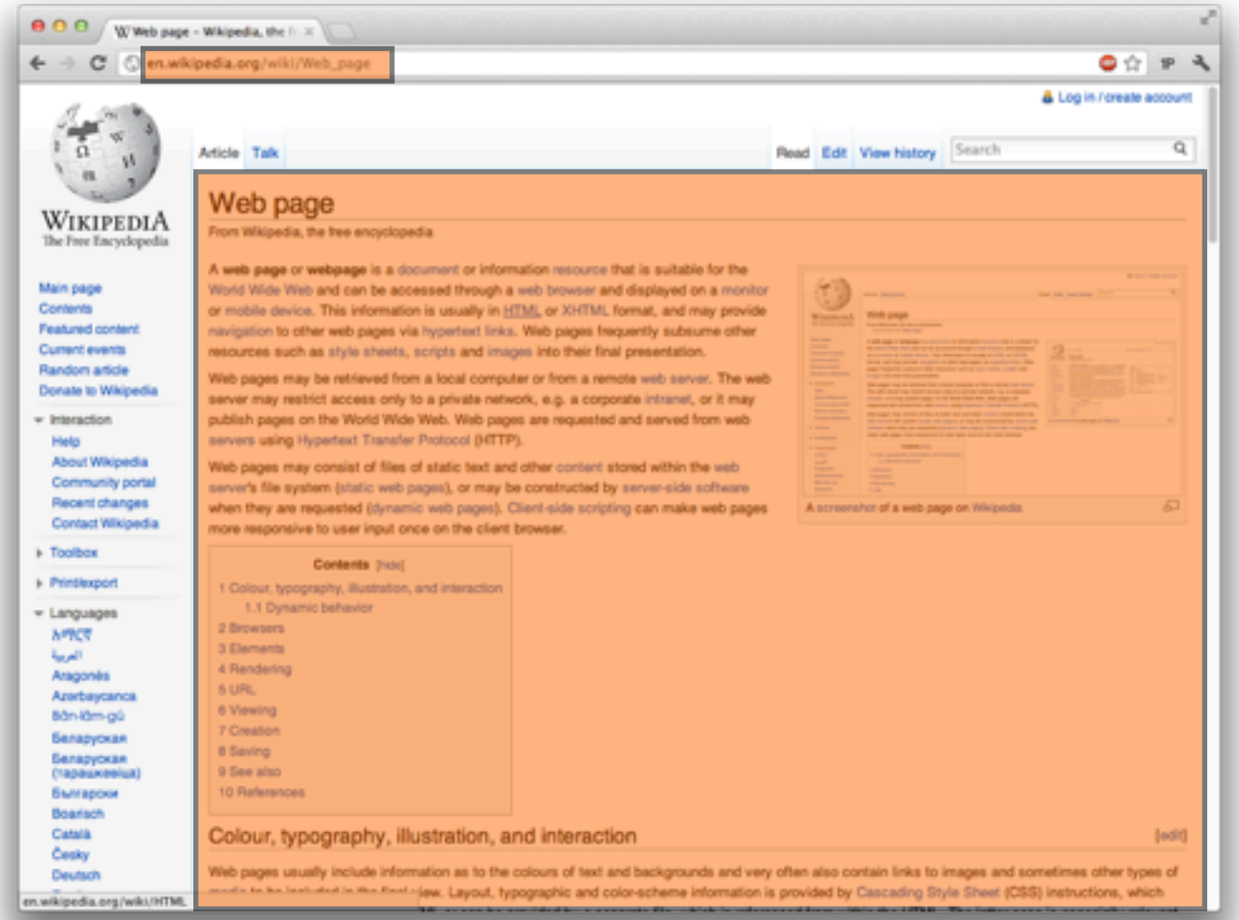
Adrian Holovaty

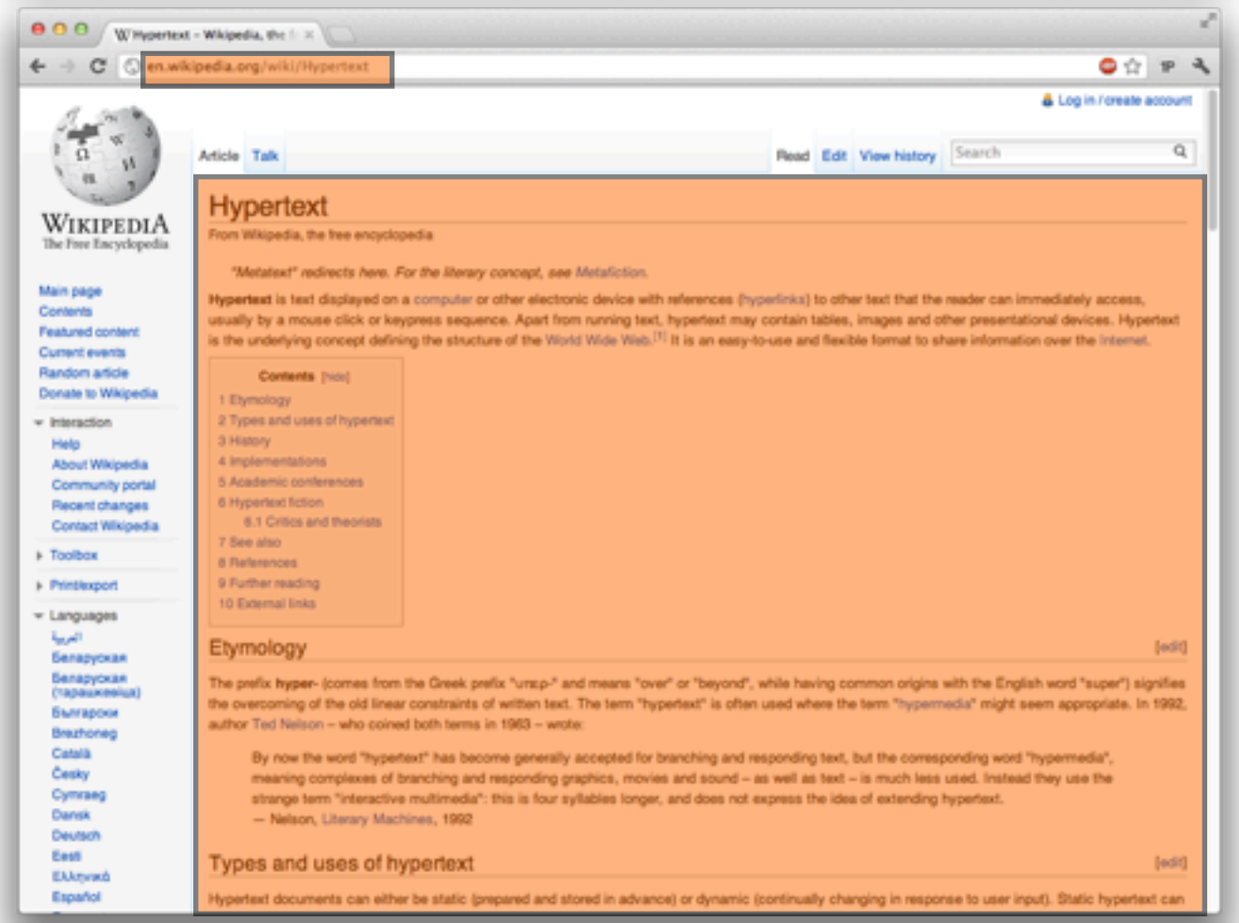
Web framework

Web framework returns full page

Web framework returns full page

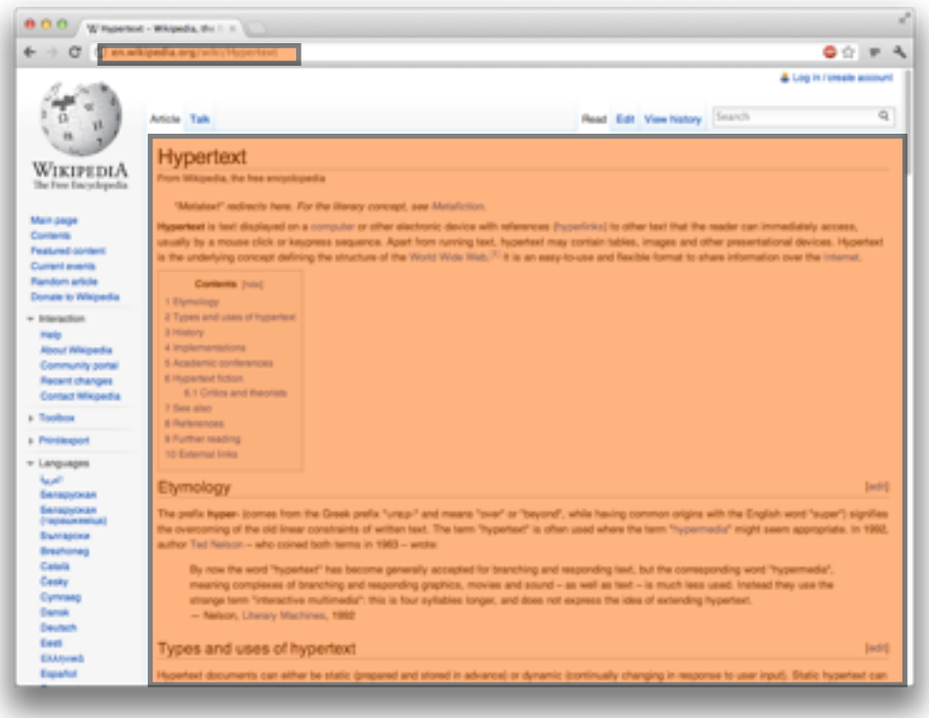



Web page - Wikipedia, the
en.wikipedia.org/wiki/Web_page
Log in / create account
Article
Talk
Read
Edit
View history
Search
WIKIPEDIA
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
Aragonès
Azərbaycanca
Bân-lâm-gú
Беларуская
Беларуская (тарашкевіца)
Български
Boarisch
Català
Česky
Deutsch
en.wikipedia.org/wiki/HTML
Web page
From Wikipedia, the free encyclopedia
A web page or webpage is a document or information resource that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a monitor or mobile device. This information is usually in HTML or XHTML format, and may provide navigation to other web pages via hypertext links. Web pages frequently subsume other resources such as style sheets, scripts and images into their final presentation.
Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).
Web pages may consist of files of static text and other content stored within the web server's file system (static web pages), or may be constructed by server-side software when they are requested (dynamic web pages). Client-side scripting can make web pages more responsive to user input once on the client browser.
A screenshot of a web page on Wikipedia
Contents [hide]
1 Colour, typography, illustration, and interaction
1.1 Dynamic behavior
2 Browsers
3 Elements
4 Rendering
5 URL
6 Viewing
7 Creation
8 Saving
9 See also
10 References
Colour, typography, illustration, and interaction
[edit]
Web pages usually include information as to the colours of text and backgrounds and very often also contain links to images and sometimes other types of
Cascading Style Sheet (CSS) instructions, which


Hypertext - Wikipedia, the
en.wikipedia.org/wiki/Hypertext
Log in / create account
Article
Talk
Read
Edit
View history
Search
WIKIPEDIA
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
Беларуская
Беларуская (тарашкевіца)
Български
Brezhoneg
Català
Česky
Cymraeg
Dansk
Deutsch
Eesti
Ελληνικά
Español
Hypertext
From Wikipedia, the free encyclopedia
"Metatext" redirects here. For the literary concept, see Metafiction.
Hypertext is text displayed on a computer or other electronic device with references (hyperlinks) to other text that the reader can immediately access, usually by a mouse click or keypress sequence. Apart from running text, hypertext may contain tables, images and other presentational devices. Hypertext is the underlying concept defining the structure of the World Wide Web.[1] It is an easy-to-use and flexible format to share information over the Internet.
Contents [hide]
1 Etymology
2 Types and uses of hypertext
3 History
4 Implementations
5 Academic conferences
6 Hypertext fiction
6.1 Critics and theorists
7 See also
8 References
9 Further reading
10 External links
Etymology
[edit]
The prefix hyper- (comes from the Greek prefix "ὑπερ-" and means "over" or "beyond", while having common origins with the English word "super") signifies the overcoming of the old linear constraints of written text. The term "hypertext" is often used where the term "hypermedia" might seem appropriate. In 1992, author Ted Nelson – who coined both terms in 1963 – wrote:
By now the word "hypertext" has become generally accepted for branching and responding text, but the corresponding word "hypermedia", meaning complexes of branching and responding graphics, movies and sound – as well as text – is much less used. Instead they use the strange term "interactive multimedia": this is four syllables longer, and does not express the idea of extending hypertext.
— Nelson, Literary Machines, 1992
Types and uses of hypertext
[edit]
Hypertext documents can either be static (prepared and stored in advance) or dynamic (continually changing in response to user input). Static hypertext can

Web framework
returns full page

Web framework
returns *diff*
and changes URL



Bing and Google Agree: Slow Pages Lose Users

by Brady Forrest | @brady | Comments: 22 | 23 June 2009

Today representatives of Google Search and Microsoft's Bing teams, Jake Brutlag and Eric Schurman respectively, presented the results of user performance tests at today's Velocity Conference. The talk was entitled The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search. These are long-term tests were designed to see what aspects of performance are most important. To know how to improve their sites both Bing and Google need to know what tweaks to page load perceptions and realities help or hurt the user experience. This is one of the first performance tests that has actual data (and is not strictly anecdotal). The numbers may seem small, but they if you are dealing in millions/billions they add up quickly.

Here are Brutlag's and Schurman's final points:

- "Speed matters" is not just lip service
- Delays under half a second impact business metrics
- The cost of delay increases over time and persists
- Use progressive rendering
- Number of bytes in response is less important than what they are and when they are sent

PJAX

(P = pushstate)

Launchpad | Basecamp Classic Projects | Highrise Contacts | Backpack Organizer | Campfire Chat | **Basecamp: msnbc**

Sign out

Support

Basecamp Projects Calendar Daily Progress Everyone Me

Jump to a project, person, or search...

EveryBlock General

Seattle postcard

Edit

Delete...

Posted by Sandor Weisz 23 hours ago

How's this look, Marina?

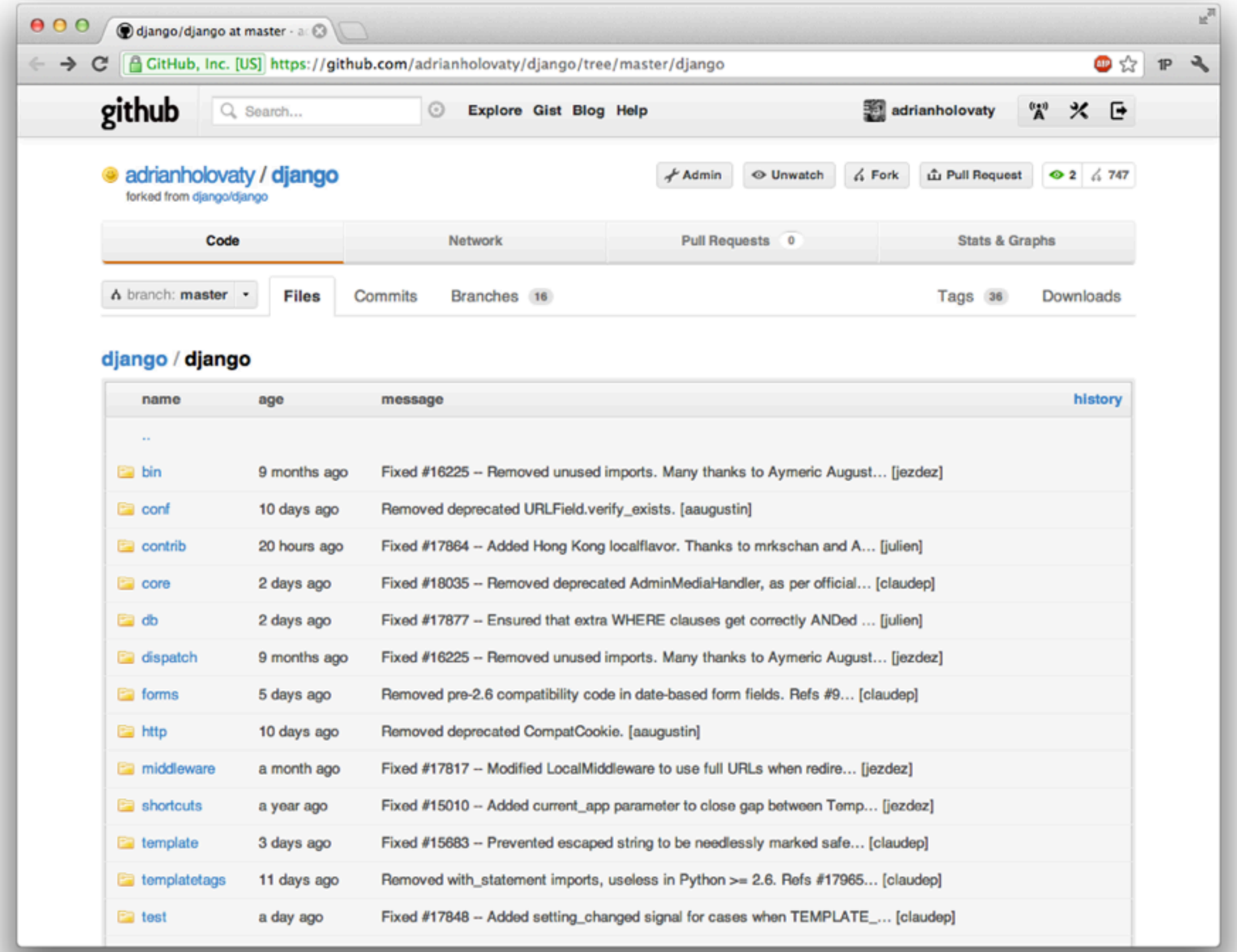
Seattle.pdf

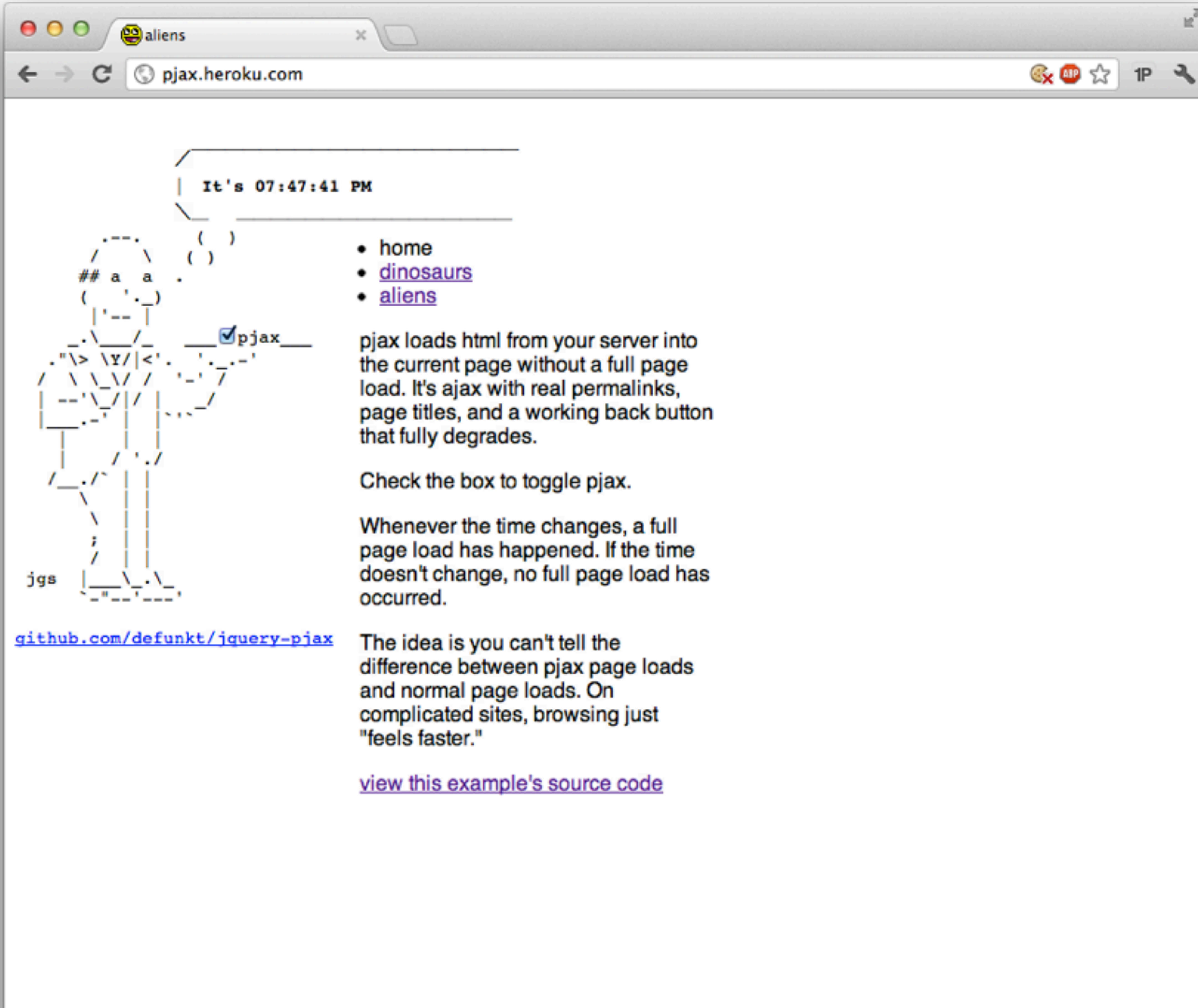
Discussion

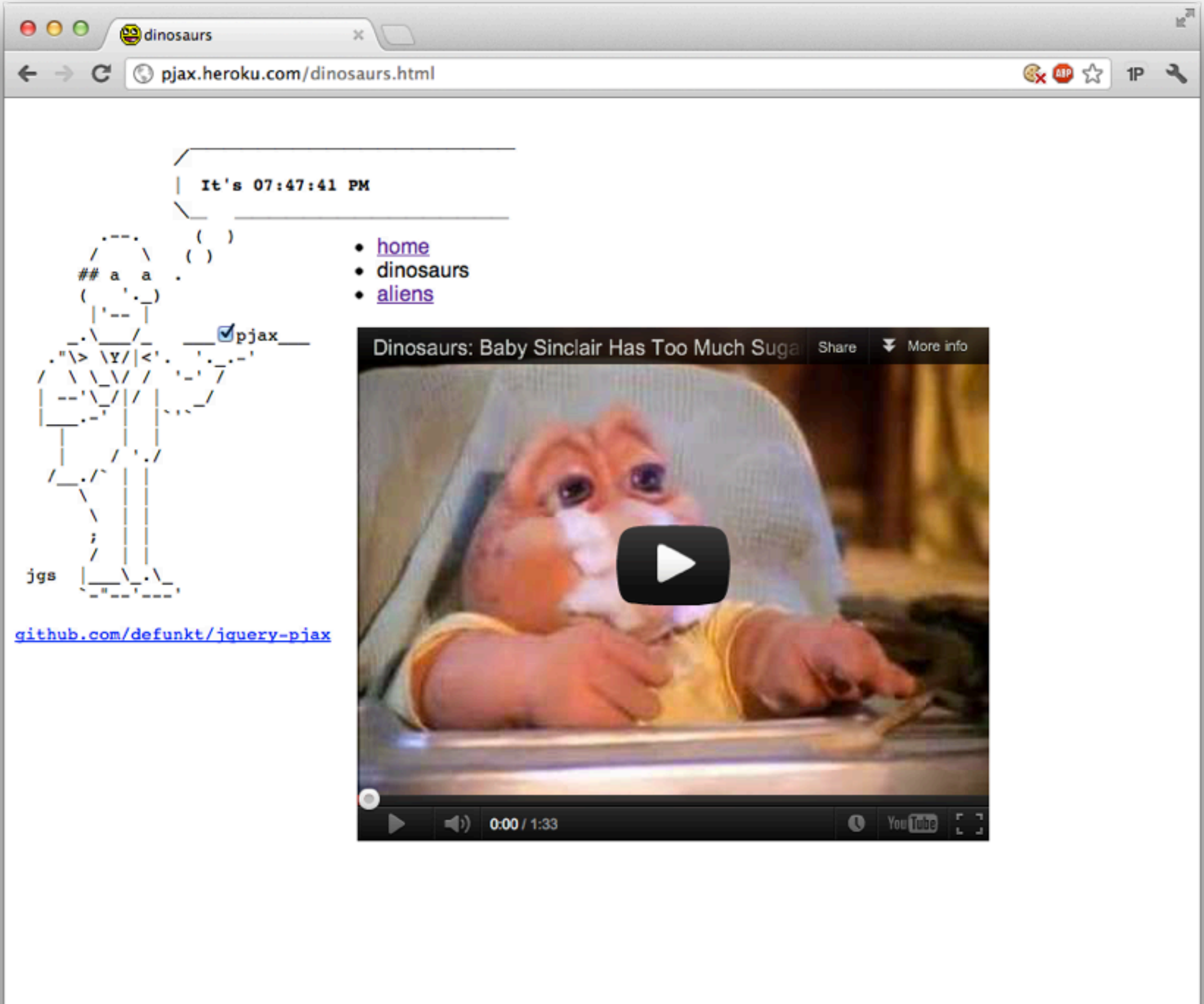
Add a comment...

History

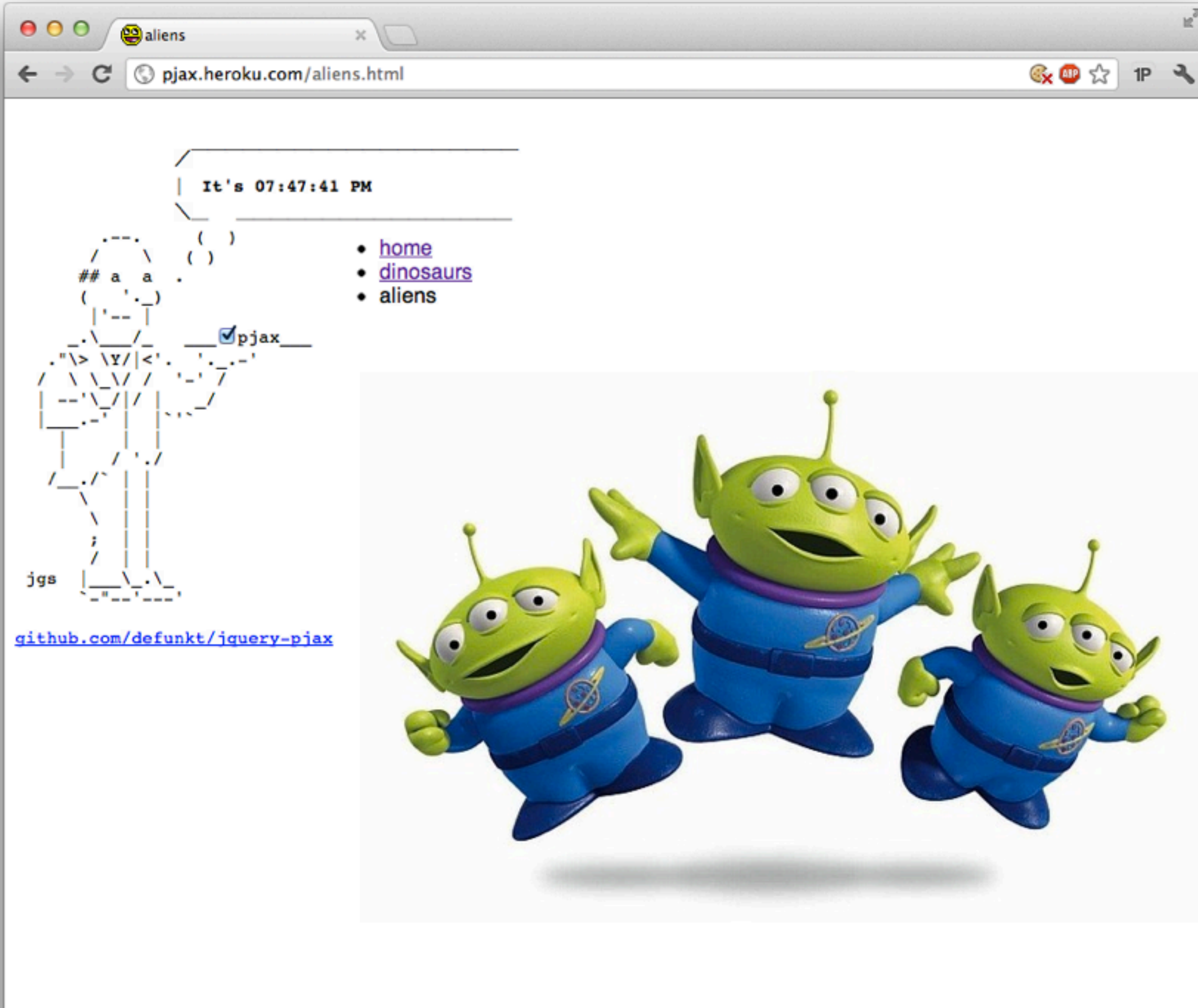
Apr 9 at 10:20am: Sandor W. posted the message (Becca Martin and Marina Gordon were notified)


django/django at master
GitHub, Inc. [US] https://github.com/adrianholovaty/django/tree/master/django
github
Search...
Explore Gist Blog Help
adrianholovaty
adrianholovaty / django
forked from django/django
Admin Unwatch Fork Pull Request 2 747
Code Network Pull Requests 0 Stats & Graphs
branch: master
Files Commits Branches 16 Tags 36 Downloads
django / django
name age message history
..
bin 9 months ago Fixed #16225 -- Removed unused imports. Many thanks to Aymeric August... [jezdez]
conf 10 days ago Removed deprecated URLField.verify_exists. [aaugustin]
contrib 20 hours ago Fixed #17864 -- Added Hong Kong localflavor. Thanks to mrkschan and A... [julien]
core 2 days ago Fixed #18035 -- Removed deprecated AdminMediaHandler, as per official... [claudep]
db 2 days ago Fixed #17877 -- Ensured that extra WHERE clauses get correctly ANDed ... [julien]
dispatch 9 months ago Fixed #16225 -- Removed unused imports. Many thanks to Aymeric August... [jezdez]
forms 5 days ago Removed pre-2.6 compatibility code in date-based form fields. Refs #9... [claudep]
http 10 days ago Removed deprecated CompatCookie. [aaugustin]
middleware a month ago Fixed #17817 -- Modified LocalMiddleware to use full URLs when redire... [jezdez]
shortcuts a year ago Fixed #15010 -- Added current_app parameter to close gap between Temp... [jezdez]
template 3 days ago Fixed #15683 -- Prevented escaped string to be needlessly marked safe... [claudep]
templatetags 11 days ago Removed with_statement imports, useless in Python >= 2.6. Refs #17965... [claudep]
test a day ago Fixed #17848 -- Added setting_changed signal for cases when TEMPLATE_... [claudep]


aliens
pjax.heroku.com
It's 07:47:41 PM
pjax
jgs
github.com/defunkt/jquery-pjax
home
dinosaurs
aliens
pjax loads html from your server into the current page without a full page load. It's ajax with real permalinks, page titles, and a working back button that fully degrades.
Check the box to toggle pjax.
Whenever the time changes, a full page load has happened. If the time doesn't change, no full page load has occurred.
The idea is you can't tell the difference between pjax page loads and normal page loads. On complicated sites, browsing just "feels faster."
view this example's source code



dinosaurs
pjax.heroku.com/dinosaurs.html
It's 07:47:41 PM
pjax
jgs
github.com/defunkt/jquery-pjax
home
dinosaurs
aliens
Dinosaurs: Baby Sinclair Has Too Much Suga
Share
More info
0:00 / 1:33


aliens
pjax.heroku.com/aliens.html
It's 07:47:41 PM
pjax
jgs
home
dinosaurs
aliens
github.com/defunkt/jquery-pjax

defunkt/jquery-pjax

GitHub, Inc. [US] https://github.com/defunkt/jquery-pjax

github Search... Explore Gist Blog Help adrianholovaty 14

defunkt / jquery-pjax

Watch Fork 3,748 184

Code | Network | Pull Requests 5 | Issues 11 | Stats & Graphs

pushState + ajax = pjax — Read more

http://pjax.heroku.com

Clone in Mac | ZIP | HTTP | Git Read-Only | https://github.com/defunkt/jquery-pjax.git | Read-Only access

branch: master | Files | Commits | Branches 4 | Tags | Downloads

Latest commit to the **master** branch

Merge pull request #112 from hazzik/patch-1

josh authored 6 days ago commit c040777774

jquery-pjax /

name	age	message	history
test	6 days ago	Add X-PJAX-Container header [josh]	
.travis.yml	2 months ago	Silence server logs [josh]	
LICENSE	a year ago	pjax [defunkt]	
README.md	a month ago	Update README.md [oscardelben]	
jquery.pjax.js	6 days ago	Merge pull request #112 from hazzik/patch-1 [josh]	

One. Functionally obtrusive, loading the href with ajax into data-pjax:

```
<a href='/explore' data-pjax='#main'>Explore</a>
```

```
$('a[data-pjax]').pjax()
```

https://github.com/defunkt/jquery-pjax

Two. Slightly obtrusive, passing a container and binding an error handler:

```
<a href='/explore' class='js-pjax'>Explore</a>
```

```
$('.js-pjax').pjax('#main')

$('#main').live('pjax:error', function(e, xhr, err) {
  $('.error').text('Something went wrong: ' + err)
})
```

https://github.com/defunkt/jquery-pjax

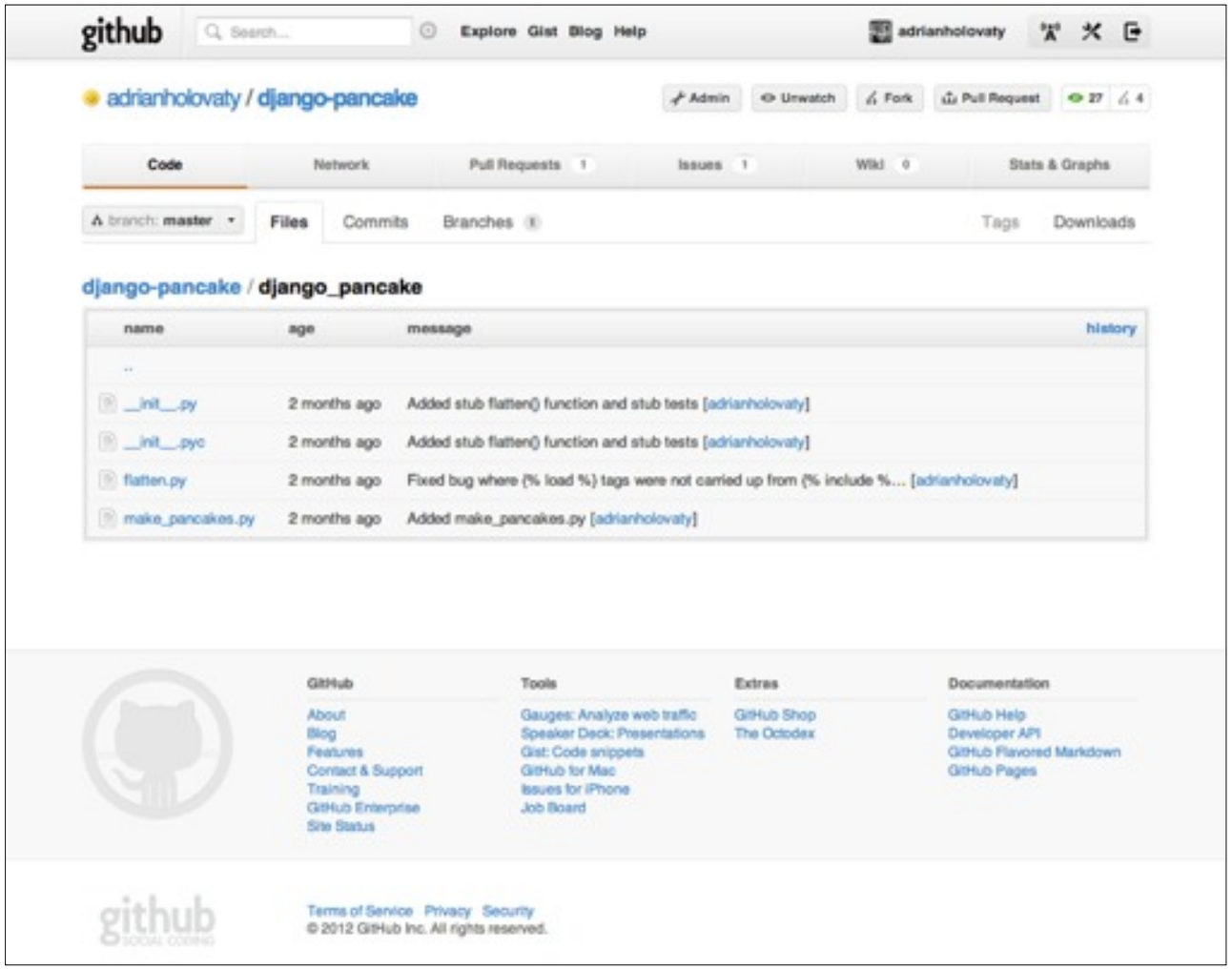
Three. Unobtrusive, showing a 'loading' spinner:

```
<div id='main'>
  <div class='loader' style='display:none'><img src='spin.gif'></div>
  <div class='tabs'>
    <a href='/explore'>Explore</a>
    <a href='/help'>Help</a>
  </div>
</div>
```

```
$('a').pjax('#main').live('click', function(){
  $(this).showLoader()
})
```

https://github.com/defunkt/jquery-pjax

```
def my_view(request):
    if request.META.get('HTTP_X_PJAX'):
        template = 'content-pjax.html'
    else:
        template = 'content.html'

    # Get data...

    return render_to_response(template, context)
```



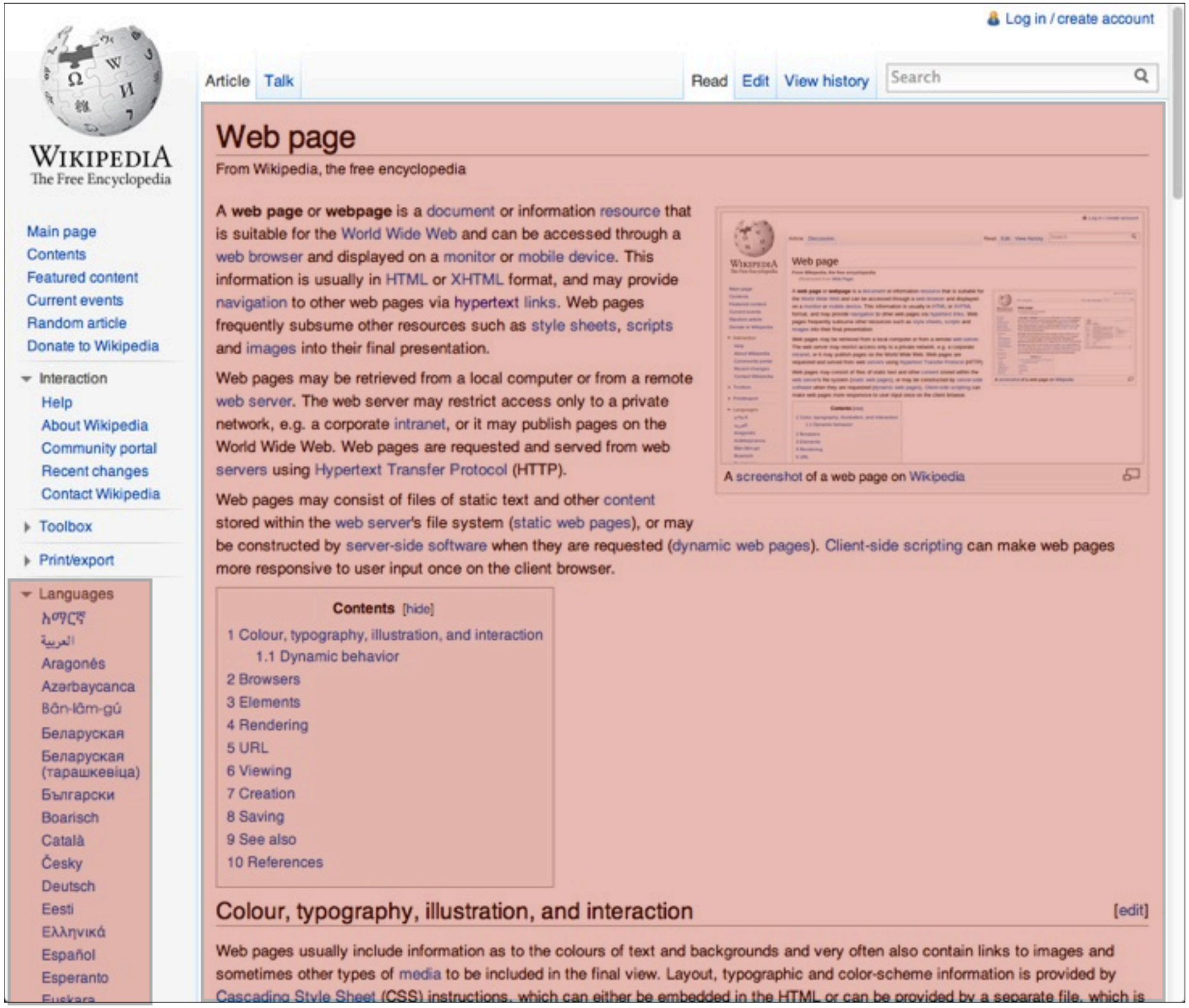
vs.

name	age	message	history
..			
__init__.py	2 months ago	Added stub flatten() function and stub tests [adrianholovaty]	
__init__.pyc	2 months ago	Added stub flatten() function and stub tests [adrianholovaty]	
flatten.py	2 months ago	Fixed bug where {% load %} tags were not carried up from {% include %}... [adrianholovaty]	
make_pancakes.py	2 months ago	Added make_pancakes.py [adrianholovaty]	

```
from djpjax import pjax


@pjax()
def my_view(request):
    # Get data...
    template = 'content.html'
    return render_to_response(template, context)
```

https://github.com/jacobian/django-pjax

Problem #1: Multiple content boxes



Problem #2: Non-visible changes

Problem #3: Context-switching

Python

JavaScript

SQL

Problem #4: Computers should do this

Two pages, one system

The Solution

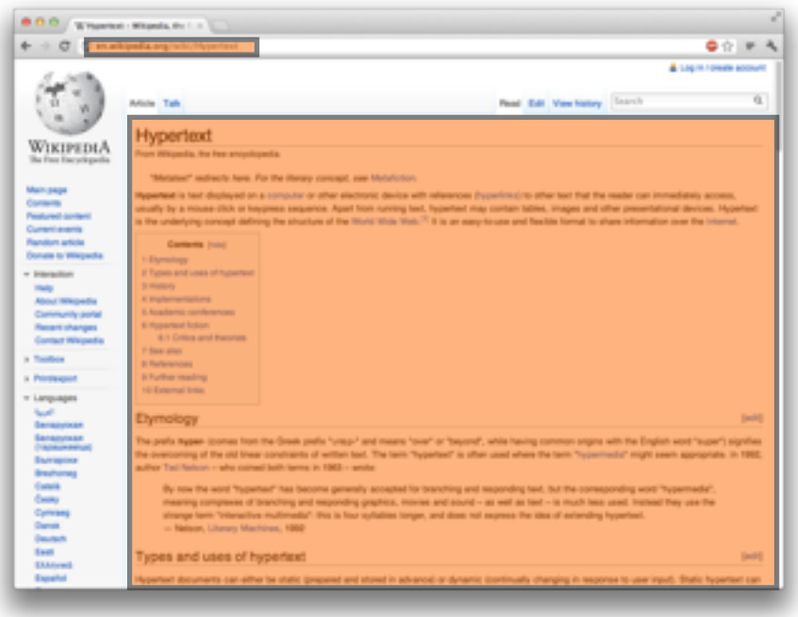
Auto-pjax

tinyurl.com/django-pjax

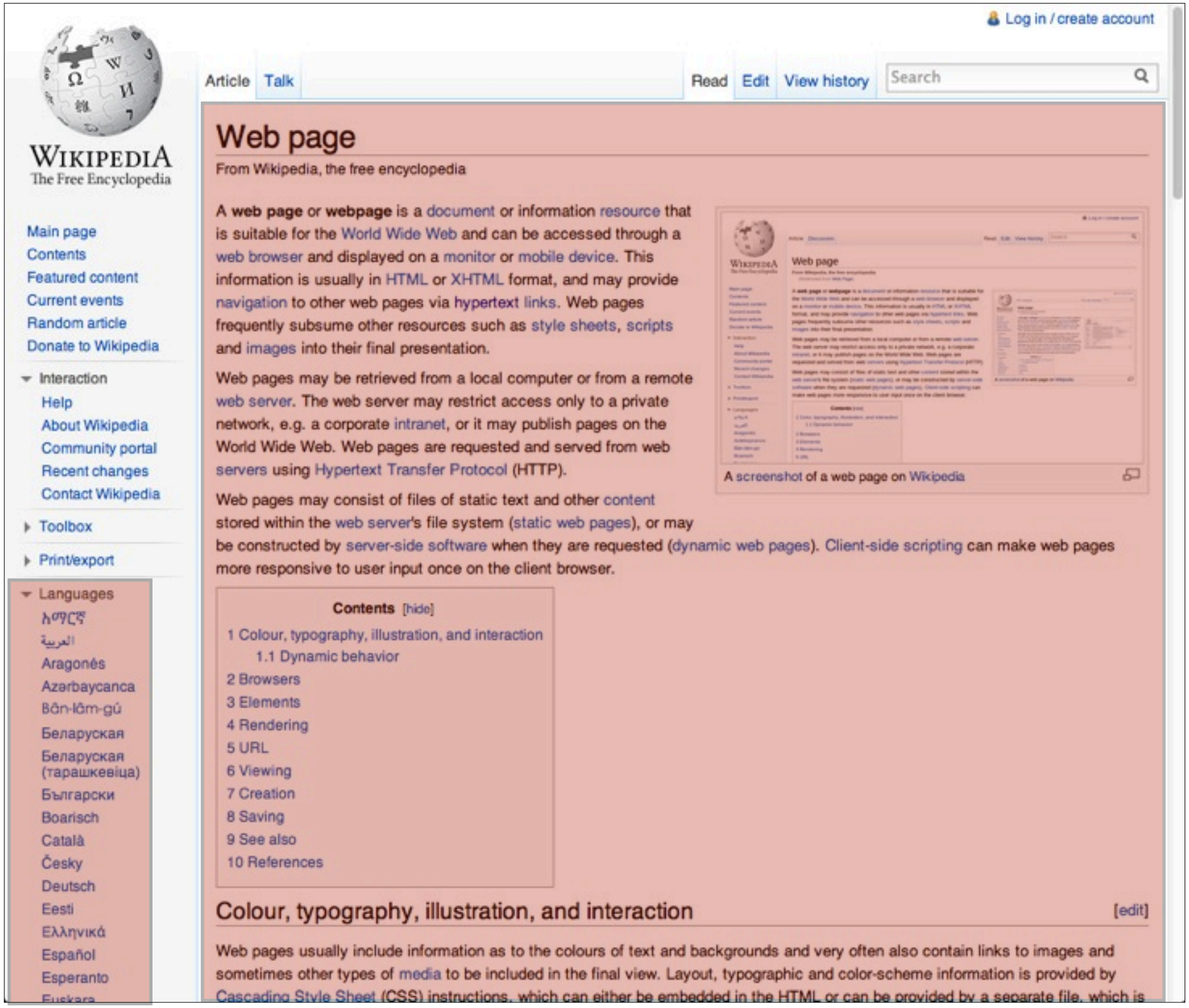
Goal #1: Auto diffs via Ajax

Web framework returns full page

Web framework returns *diff*



Goal #2: Support multiple content boxes

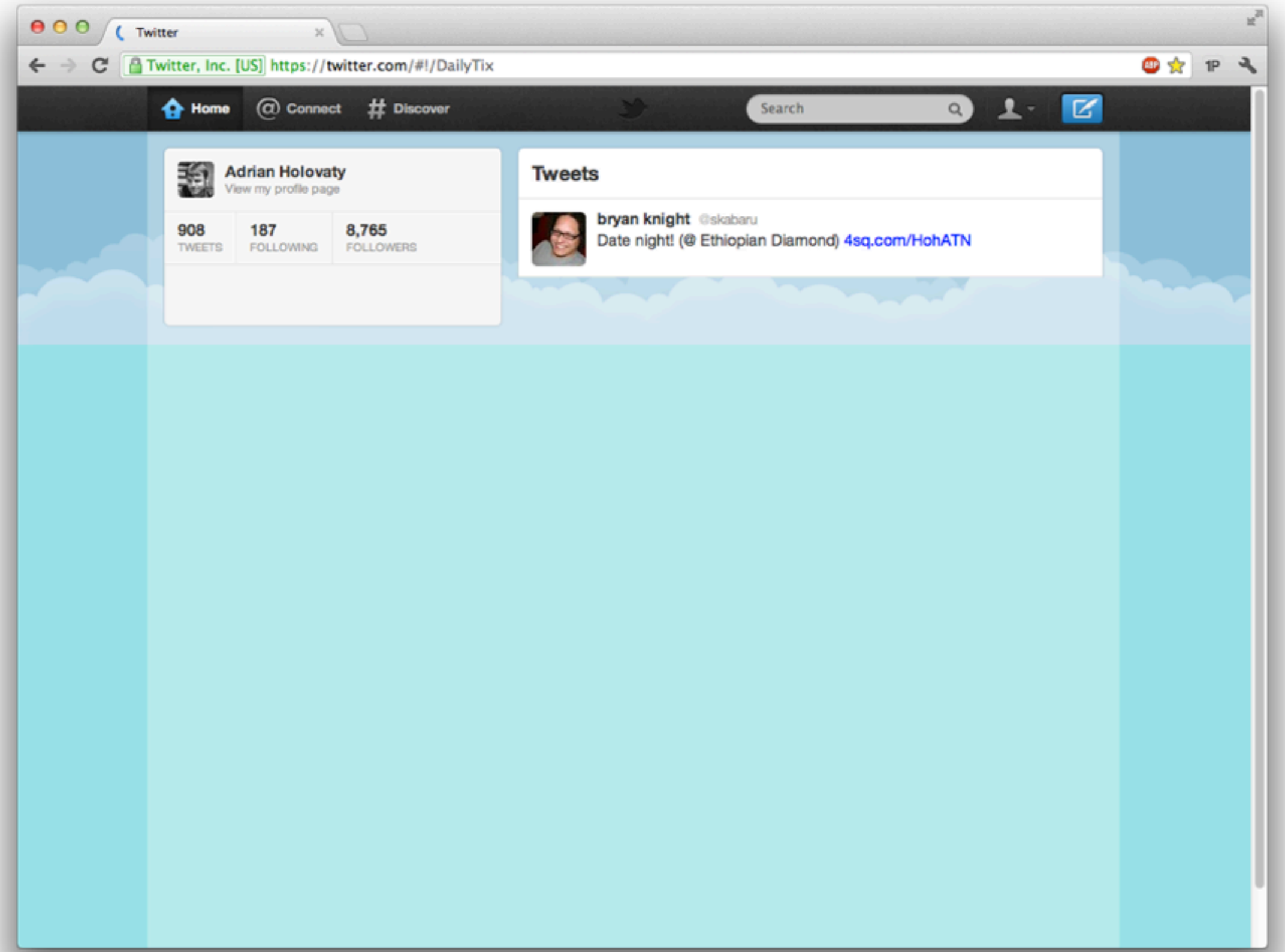


Goal #3: Support non-visible changes (JS!)

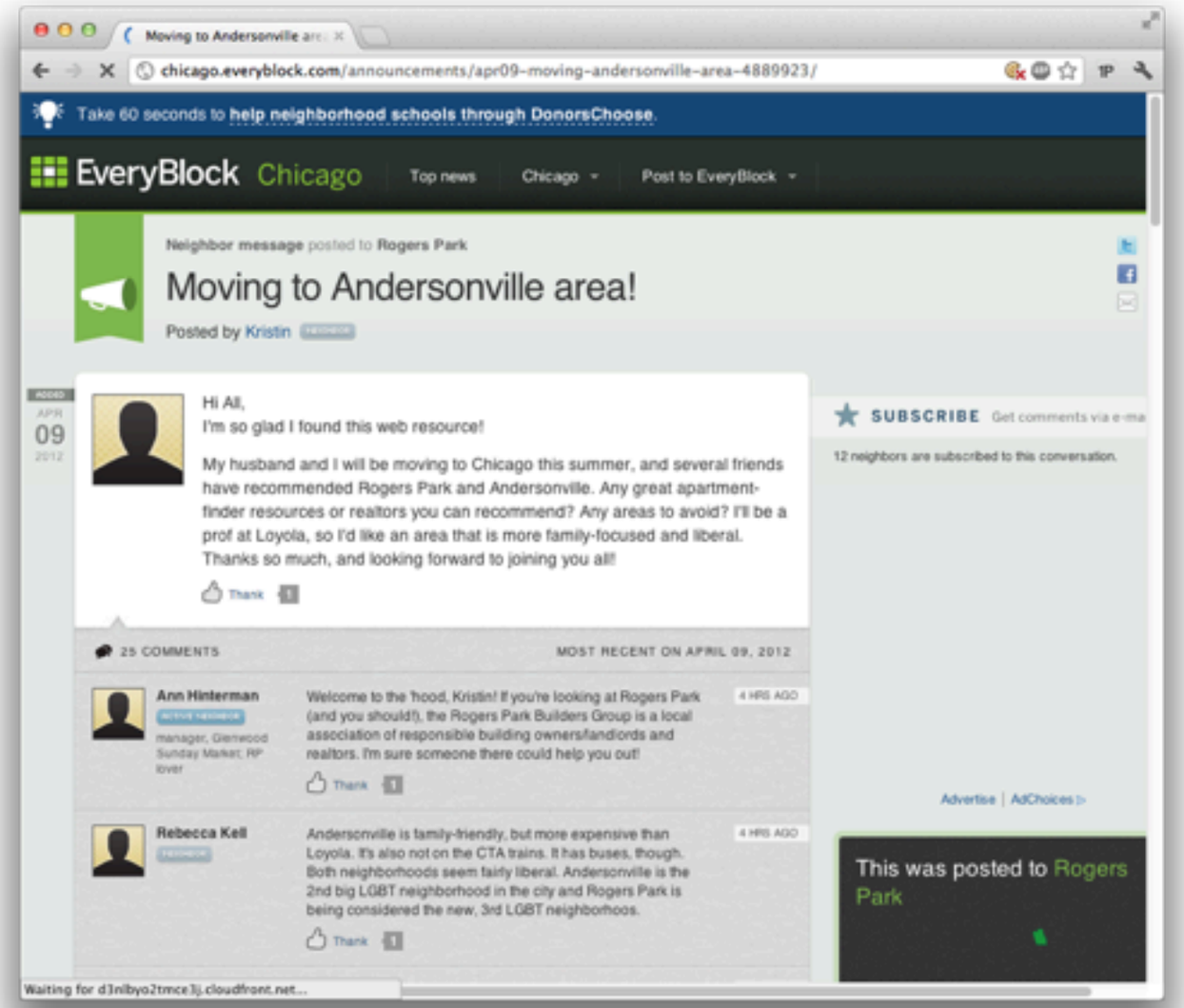
Goal #4: Respect URLs and permalinks

twitter.com/#!/adrianholovaty/status/18947108

Goal #5: Don't require JS on initial page load



Goal #6: Bail if needed

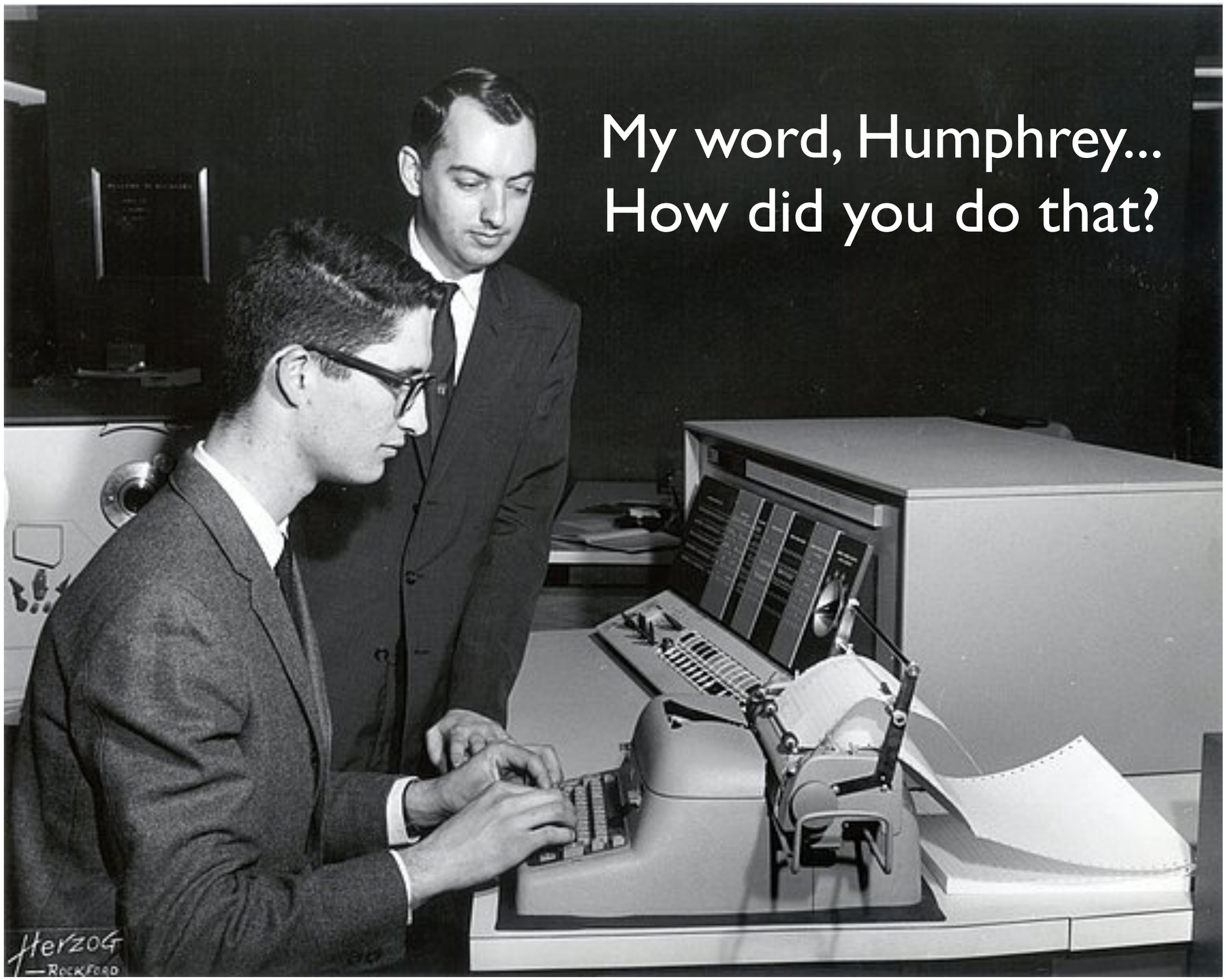


Goal #7: Security

Any application-specific data that's revealed should be safe for public consumption.

Goal #8: Favor correctness, win devs' trust

Goal #9: Amaze



Why pass HTML instead of structured data?

Structured data

```
{"title": "Web page",
"content": "A web page or webpage is a document or...",
"last_edited": "2012-03-25"}
```

- Then you need client-side templating.
- Optimize for server-side development, because you have to do it anyway. And it's more fun. :-)

Example

Log in / create account

WIKIPEDIA
The Free Encyclopedia

Article | Talk | Read | Edit | View history | Search

Web page

From Wikipedia, the free encyclopedia

A **web page** or **webpage** is a document or information resource that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a monitor or mobile device. This information is usually in HTML or XHTML format, and may provide navigation to other web pages via hypertext links. Web pages frequently subsume other resources such as style sheets, scripts and images into their final presentation.

Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).

A screenshot of a web page on Wikipedia

Web pages may consist of files of static text and other content stored within the web server's file system (static web pages), or may be constructed by server-side software when they are requested (dynamic web pages). Client-side scripting can make web pages more responsive to user input once on the client browser.

Contents [hide]

Colour, typography, illustration, and interaction [edit]

Web pages usually include information as to the colours of text and backgrounds and very often also contain links to images and sometimes other types of media to be included in the final view. Layout, typographic and color-scheme information is provided by Cascading Style Sheet (CSS) instructions, which can either be embedded in the HTML or can be provided by a separate file, which is

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

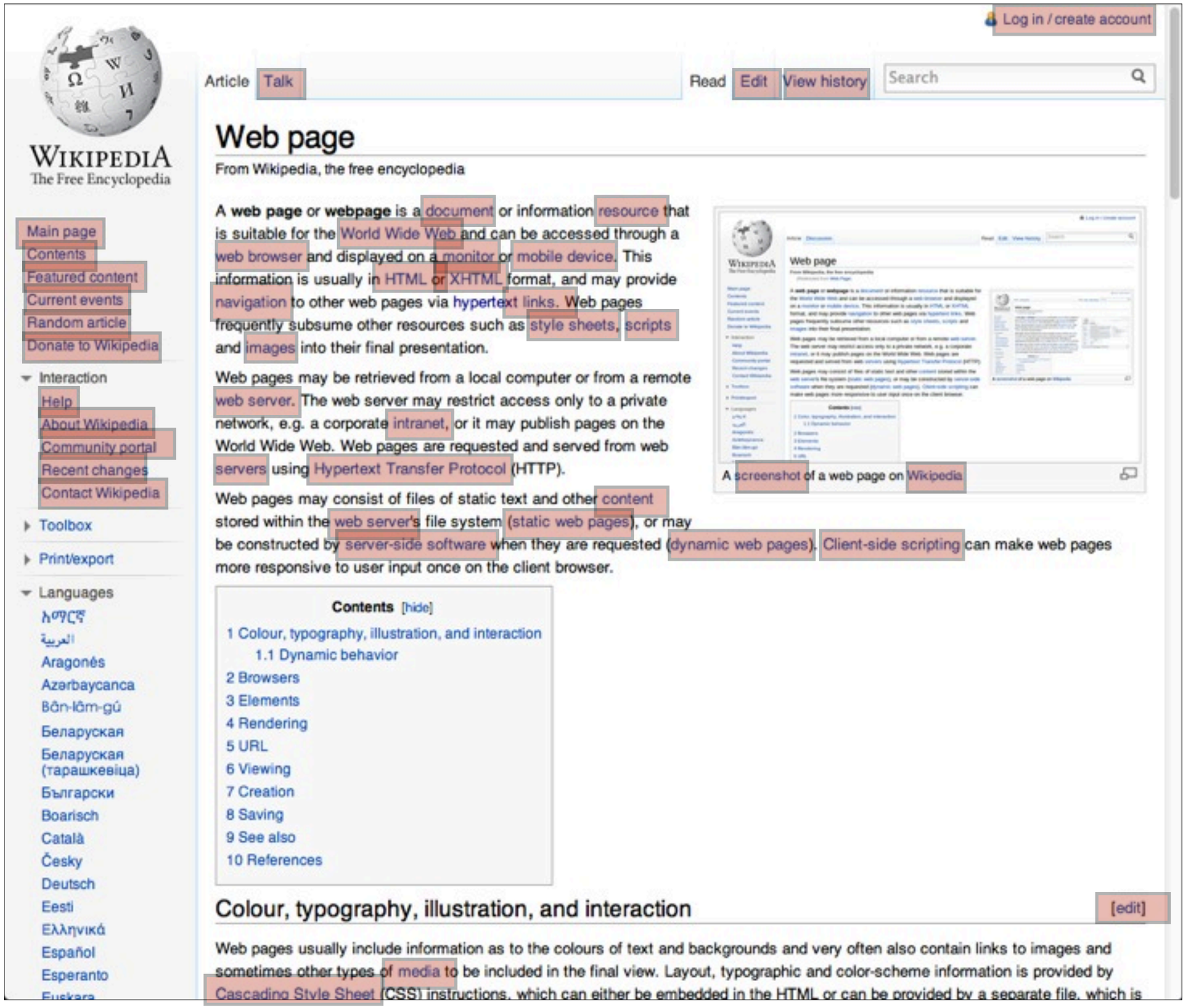
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
አማርኛ
العربية
Aragonés
Azərbaycanca
Bân-lâm-gú
Беларуская
Беларуская (тарашкевіца)
Български
Boarisch
Català
Česky
Deutsch
Eesti
Ελληνικά
Español
Esperanto
Euskara

1. Find all on-site links



2. Set onclick

Load URL with Ajax:

Pass X-PJAX header

Pass current URL

When response comes in:

Replace diffs (MAGIC!)

On error:

Load next page manually

3. Django middleware

If request has X-PJAX header:

Calculate diff (MAGIC!)

Return JSON of diff

Overview


Web page - Wikipedia, the
en.wikipedia.org/wiki/Web_page
Log in / create account
Article
Talk
Read
Edit
View history
Search
WIKIPEDIA
The Free Encyclopedia
Web page
From Wikipedia, the free encyclopedia
A web page or webpage is a document or information resource that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a monitor or mobile device. This information is usually in HTML or XHTML format, and may provide navigation to other web pages via hypertext links. Web pages frequently subsume other resources such as style sheets, scripts and images into their final presentation.
Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).
Web pages may consist of files of static text and other content stored within the web server's file system (static web pages), or may be constructed by server-side software when they are requested (dynamic web pages). Client-side scripting can make web pages more responsive to user input once on the client browser.
Contents [hide]
1 Colour, typography, illustration, and interaction
1.1 Dynamic behavior
2 Browsers
3 Elements
4 Rendering
5 URL
6 Viewing
7 Creation
8 Saving
9 See also
10 References
Colour, typography, illustration, and interaction [edit]
Web pages usually include information as to the colours of text and backgrounds and very often also contain links to images and sometimes other types of
Cascading Style Sheet (CSS)

Overview

Click link

Trigger Ajax request

Overview

Click link

Trigger Ajax request

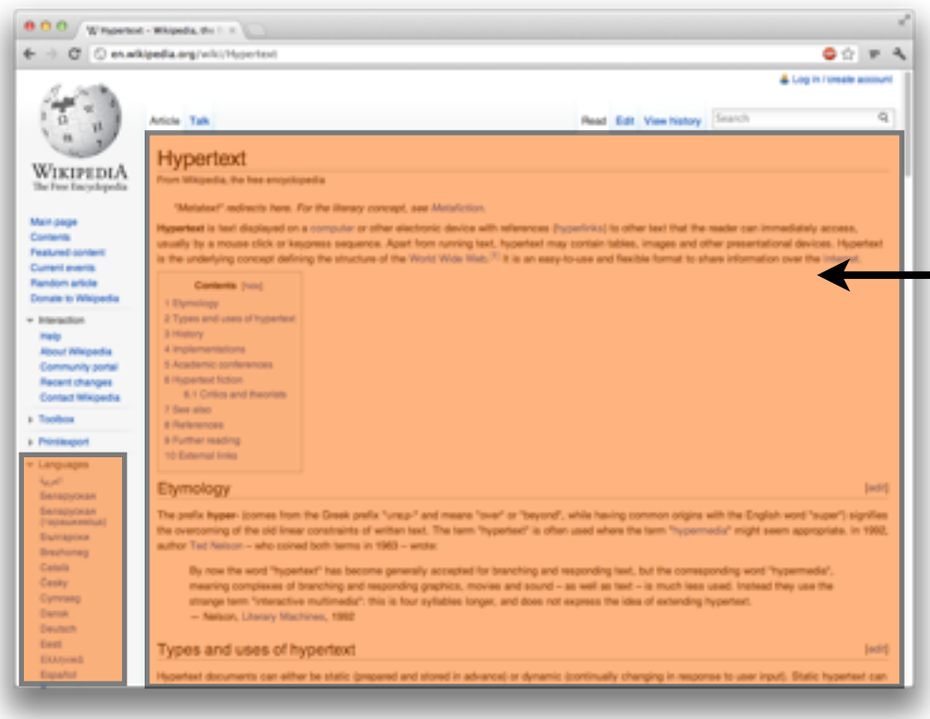
Server says: “These bits changed”

Overview

Click link

Trigger Ajax request

Server says: “These bits changed”



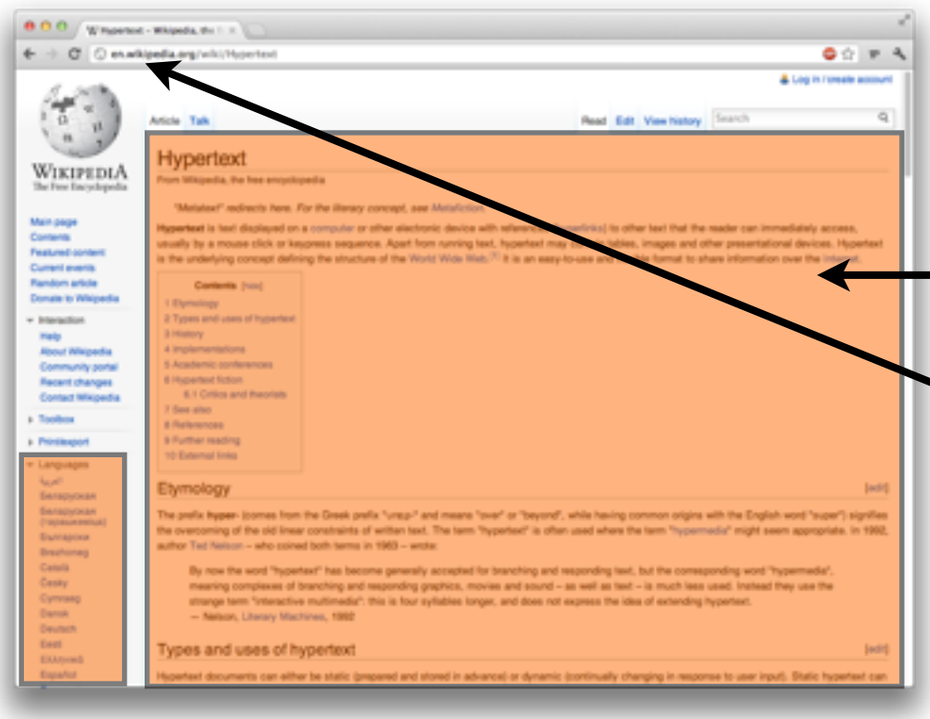
JavaScript changes the page in place

Overview

Click link

Trigger Ajax request

Server says: “These bits changed”



JavaScript changes the page in place

URL gets updated with pushstate

Two bits of MAGIC

- Server: Which bits are different?
- Client: How do we update the HTML document?

Django template inheritance

base.html

```
<html>
<body>
<h1>{% block h1 %}{% endblock %}</h1>

<div id="content">
{% block content %}{% endblock %}
</div>
```

Django template inheritance

base.html

```
<html>
<body>
<h1>{% block h1 %}{% endblock %}</h1>

<div id="content">
{% block content %}{% endblock %}
</div>
```

child1.html

```
{% extends "base.html" %}

{% block h1 %}
First child
{% endblock %}

{% block content %}
<p>Welcome to the first child!</p>
{% endblock %}
```

Django template inheritance

base.html

```
<html>
<body>
<h1>{% block h1 %}{% endblock %}</h1>

<div id="content">
{% block content %}{% endblock %}
</div>
```

child1.html

```
{% extends "base.html" %}

{% block h1 %}
First child
{% endblock %}

{% block content %}
<p>Welcome to the first child!</p>
{% endblock %}
```

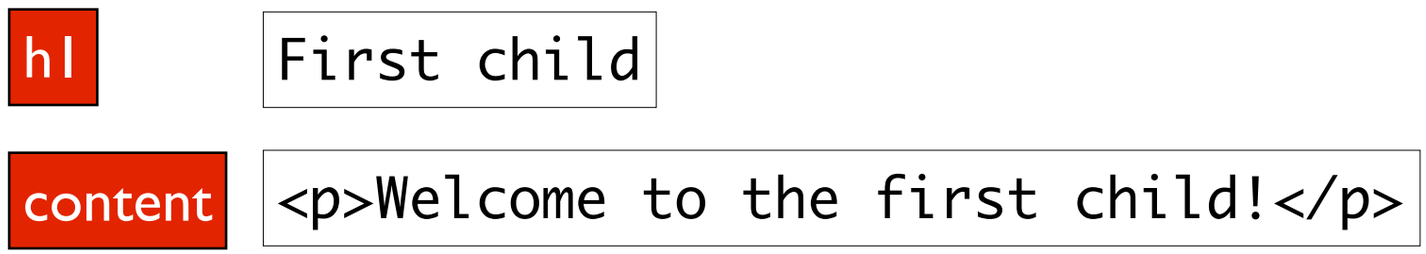
child2.html

```
{% extends "base.html" %}

{% block h1 %}
Second child
{% endblock %}

{% block content %}
<p>Welcome to the second kid!</p>
{% endblock %}
```

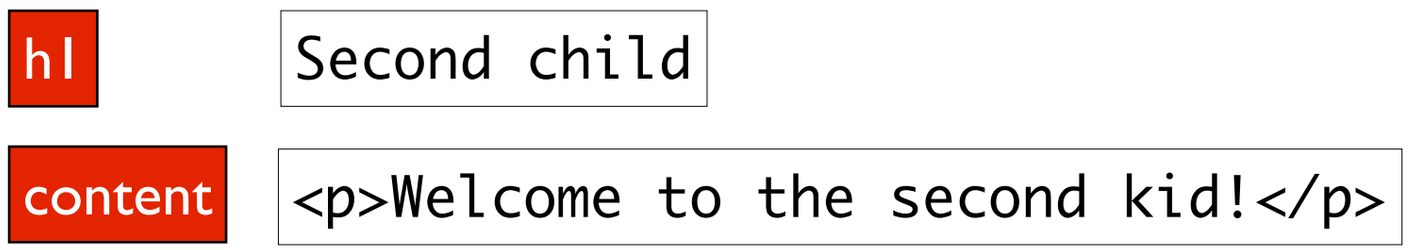
Django template inheritance

base.html

child1.html



child2.html



Basic diff using template inheritance

http://example.com/child1/

```
<html><body><h1>
```

```
h1
```

```
First child
```

```
</h1><div id="content">
```

```
content
```

```
<p>Welcome to the first child!</p>
```

```
</div>
```

http://example.com/child2/

```
{
"block:h1": "First child",
"block:content": "<p>Welcome to the second kid!</p>"
}
```

Hooking into the HTML: Option 1

http://example.com/child1/

Hooking into the HTML: Option 1

http://example.com/child1/

```
<html><body><h1>
<div id="dj_block_h1">
h1  First child
</div>
</h1><div id="content">
<div id="dj_block_content">
content  <p>Welcome to the first child!</p>
</div>
</div>
```

```
// data = from ajax.
$('#dj_block_h1').html(data['block:h1']);
$('#dj_block_content').html(data['block:content']);
```

Putting it all together

Template
(as written)

```
<html><body>
<h1>{% block h1 %}{% endblock %}</h1>
<div id="content">
{% block content %}{% endblock %}
</div>
```

Putting it all together

Template (as written)

```
<html><body>
<h1>{% block h1 %}{% endblock %}</h1>
<div id="content">
{% block content %}{% endblock %}
</div>
```

Rendered HTML

```
<html><body>
<h1><div id="dj_block_h1">First child</div></h1>
<div id="content"><div id="dj_block_content">
<p>Welcome to the first child!</p>
</div></div>
```

h1	First child	content	Welcome to the first child!

Putting it all together

Template (as written)

```
<html><body>
<h1>{% block h1 %}{% endblock %}</h1>
<div id="content">
{% block content %}{% endblock %}
</div>
```

Rendered HTML

```
<html><body>
<h1><div id="dj_block_h1">First child</div></h1>
<div id="content"><div id="dj_block_content">
<p>Welcome to the first child!</p>
</div></div>
```

h1	First child	content	Welcome to the first child!

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Putting it all together

Template (as written)

```
<html><body>
<h1>{% block h1 %}{% endblock %}</h1>
<div id="content">
{% block content %}{% endblock %}
</div>
```

Rendered HTML

```
<html><body>
<h1><div id="dj_block_h1">First child</div></h1>
<div id="content"><div id="dj_block_content">
<p>Welcome to the first child!</p>
</div></div>
```

h1	First child	content	Welcome to the first child!

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Ajax response

```
{"dj_block_h1": "Second child",
"dj_block_content": "<p>Welcome to the second kid!</p>"}
```

Putting it all together

Template (as written)

```
<html><body>
<h1>{% block h1 %}{% endblock %}</h1>
<div id="content">
{% block content %}{% endblock %}
</div>
```

Rendered HTML

```
<html><body>
<h1><div id="dj_block_h1">First child</div></h1>
<div id="content"><div id="dj_block_content">
<p>Welcome to the first child!</p>
</div></div>
```

h1	First child	content	Welcome to the first child!

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Ajax response

```
{"dj_block_h1": "Second child",
"dj_block_content": "<p>Welcome to the second kid!</p>"}
```

JS callback

```
$('#dj_block_h1').html(data['dj_block_h1']);
$('#dj_block_content').html(data['dj_block_content']);
history.pushState(null, document.title, "/page2/");
```

Template variables

```
def my_view(request):
    return render_to_response('test.html', {
        'name': request.GET.get('name'),
    })
```

test.html

```
<html>
<body>
<h1>Hello, {{ name }}!</h1>
```

Template variables

```
<html>
<body>
<h1>Hello, {{ name }}!</h1>
```

```
<html><body><h1>Hello,
<span id="dj_var_name">
name
</span>
!</div>
```

```
// data = from ajax.
$('#dj_var_name').html(data['dj_var_name']);
```

Then it starts to get tricky...

```
<html>
<head>
  <title>{{ title }}</title>
</head>
```

Can't put `<span id="dj_var_title">` within <title>

Insertion JavaScript needs to special-case this:

```
document.title = 'New title';
```

How does the framework know this is a title??

How does the framework know this is a title?

```
<html>
<head>
  <title>{{ title title }}</title>
</head>
```

Option 1: Provide hints in the template.

Requires extension of template syntax.
Requires manual work by template authors. :-(

How does the framework know this is a title?

```
<html>
<head>
  <title>{{ title }}</title>
</head>
```

Option 2: Parse HTML.

Requires SUPER-MEGA-ALL-IN-ONE HTML, JavaScript, CSS, Django template language parser.

How does the framework know this is a title?

```
<html>
<head>
  <title>{{ title }}</title>
</head>
```

Option 3: ALWAYS change the <title>.

This is what jquery-pjax does.

Partial titles

```
<html>
<head>
  <title>{{ title }} | EveryBlock.com</title>
</head>
```

Can't put `<span id="dj_var_title">` within <title>

document.title call now needs extra information

```
document.title = 'New title' + ' | EveryBlock.com ';
```

A good argument for always changing the <title>.

Snippets in HTML tags

```
<body class="{{ body_class }}">
  <p {{ p_attrs }}>
     {{ content }}
  </p>
</body>
```

Can't put `<span id="dj_var_body_class">` within attribute

How does the framework know these are within tags/attributes?

Snippets in HTML tags

```
<body class="{{ body_class }}">
  <p {{ p_attrs }}>
     {{ content }}
  </p>
</body>
```

Option 1: Search parents until you find a sane container.

Snippets in HTML tags

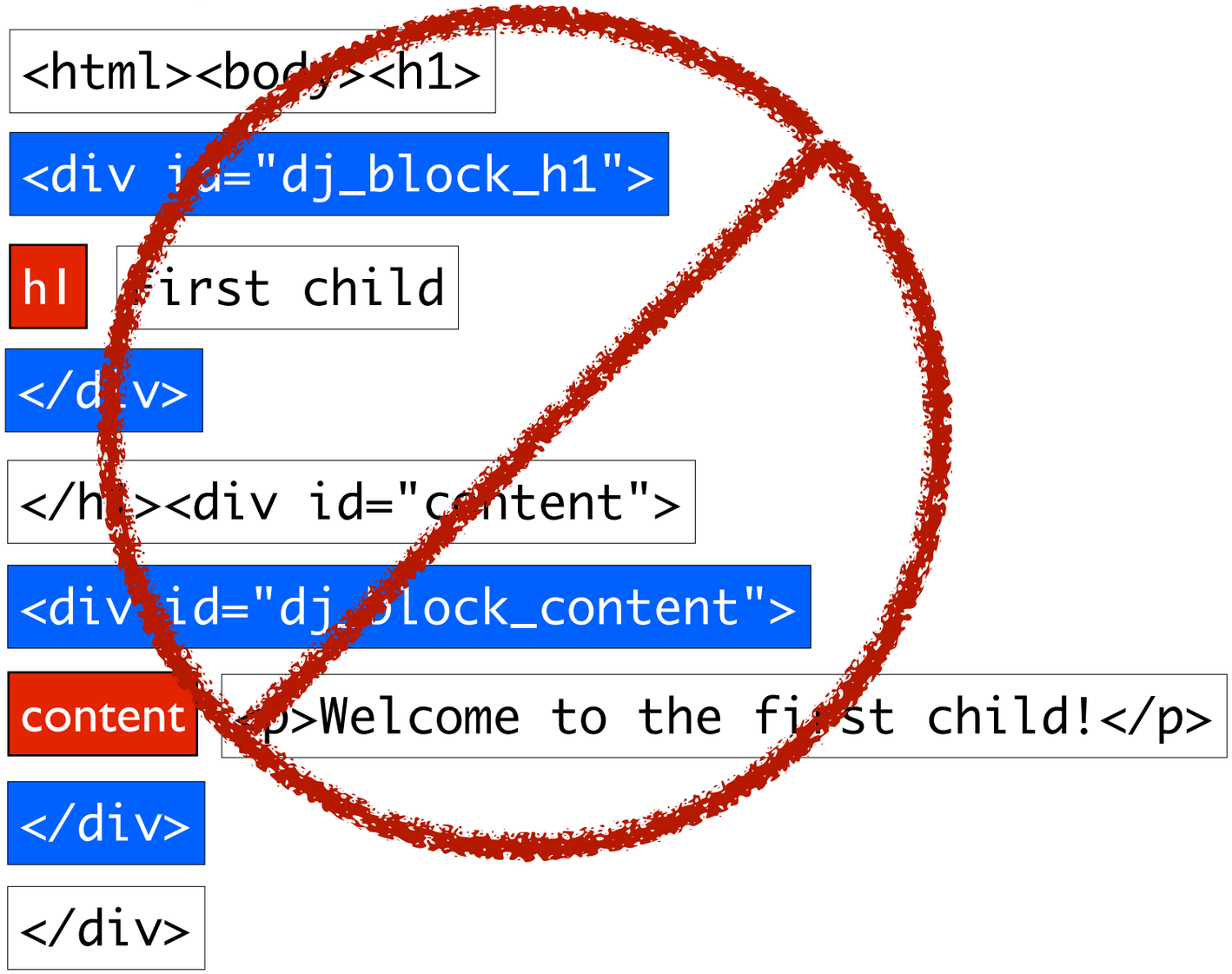
```
<body class="{{ body_class }}">
  <p {{ p_attrs }}>
     {{ content }}
  </p>
</body>
```

Option 2: Parse HTML and provide selectors for diffs.

```
// data = from ajax.
$('body').attr('class', data['body_class']);
$('body > p:nth-child(1)').attr(data['p_attrs']);
$('body > p:nth-child(1)').html(data['content']);
```

Hooking into the HTML: Option 1

http://example.com/child1/



Hooking into the HTML: Option 2

http://example.com/child1/

```
<html><body><h1>
```

h1 First child

```
</h1><div id="content">
```

content `<p>Welcome to the first child!</p>`

```
</div>
```

http://example.com/child2/

```
{
"h1": "Second child",
"#content": "<p>Welcome to the second kid!</p>"
}
```

Hooking into the HTML: Option 2

Instead of sending data...

```
{
"h1": "Second child",
"#content": "<p>Welcome to the second kid!</p>"
}
```

...send instructions:

```
[
"$('h1').html('Second child')",
"$(#content).html('<p>Welcome to the second kid!</p>')"
]
```

Evil HTML snippets

```
<div{{ rest_of_div }}
</div>
```

rest_of_div

```
 class="foo">
```

Evil HTML snippets

```
<div{{ rest_of_div }}
</div>
```

rest_of_div

```
 class="foo">
```

Solution: Don’t do that.

Insertion types

Block

```
<div>{{ foo }}</div>
```

Insertion types

Block

```
<div>{{ foo }}</div>
```

Inline

```
<span>{{ foo }}</span>
```

Insertion types

Block	`<div>{{ foo }}</div>`
Inline	`<span>{{ foo }}</span>`
Text	`<pre>{{ foo }}</pre>`

Insertion types

Block	`<div>{{ foo }}</div>`
Inline	`<span>{{ foo }}</span>`
Text	`<pre>{{ foo }}</pre>`
Title	`<title>{{ foo }}</title>`

Insertion types

Block	`<div>{{ foo }}</div>`
Inline	`<span>{{ foo }}</span>`
Text	`<pre>{{ foo }}</pre>`
Title	`<title>{{ foo }}</title>`
Titlebit	`<title>{{ foo }} | Example.com</title>`

Insertion types

Block

```
<div>{{ foo }}</div>
```

Inline

```
<span>{{ foo }}</span>
```

Text

```
<pre>{{ foo }}</pre>
```

Title

```
<title>{{ foo }}</title>
```

Titlebit

```
<title>{{ foo }} | Example.com</title>
```

CSS

```
<style type="text/css">
{{ foo }}
</style>
```

Insertion types

Block

```
<div>{{ foo }}</div>
```

Inline

```
<span>{{ foo }}</span>
```

Text

```
<pre>{{ foo }}</pre>
```

Title

```
<title>{{ foo }}</title>
```

Titlebit

```
<title>{{ foo }} | Example.com</title>
```

CSS

```
<style type="text/css">
{{ foo }}
</style>
```

Dull

```
<head>
{{ foo }}<!-- meta tags -->
</head>
```

Insertion types

Block

```
<div>{{ foo }}</div>
```

Inline

```
<span>{{ foo }}</span>
```

Text

```
<pre>{{ foo }}</pre>
```

Title

```
<title>{{ foo }}</title>
```

Titlebit

```
<title>{{ foo }} | Example.com</title>
```

CSS

```
<style type="text/css">
{{ foo }}
</style>
```

Dull

```
<head>
{{ foo }}<!-- meta tags -->
</head>
```

Critical

```
<script>
window.onload=function() {
{% block onloadjs %}{% endblock %}
}
</script>
```

Template logic

```
<body>
{% if some_condition %}
  Some condition is true!
{% else %}
  It's false.
{% endif %}
</body>
```

Template logic

```
<body>
{% if some_condition %}
  Some condition is true!
{% else %}
  It's false.
{% endif %}
</body>
```

Treat the whole block as a changeable thing.

Nested template logic

```
<body>
{% if condition %}
  Condition is true!
  {% if sub_condition %}
    And sub-condition!
  {% else %}
    But not sub-condition.
  {% endif %}
{% else %}
  Condition is false.
{% endif %}
</body>
```

Treat each block as a changeable thing.

Template logic in crafty places

```
<body class="{% if foo %}foo{% else %}bar{% endif %}">
```

Removal of markup

```
<body>
{% if condition %}
  Condition is true!
  {% if sub_condition %}
    And sub-condition!
  {% else %}
    But not sub-condition.
  {% endif %}
{% else %}
  Condition is false.
{% endif %}
</body>
```

condition=True
sub_condition=False

```
<body>
Condition is true!
But not sub-condition.
</body>
```

condition=True
sub_condition=True

```
<body>
Condition is true!
And sub-condition!
</body>
```

condition=False
sub_condition=True

```
<body>
Condition is false.
</body>
```

Putting it all together

Template
(as written)

```
<html>
<head><title>{{ title }}</title></head>
<body>
<h1>Welcome, {{ name }}!</h1>
</body>
```

Putting it all together

Template (as written)

```
<html>
<head><title>{{ title }}</title></head>
<body>
<h1>Welcome, {{ name }}!</h1>
</body>
```

Rendered HTML

```
<html>
<head><title>Title here</title></head>
<body>
<h1>Welcome, Adrian!</h1>
</body>
```

title `Title here` name `Adrian`

Putting it all together

Template (as written)

```
<html>
<head><title>{{ title }}</title></head>
<body>
<h1>Welcome, {{ name }}!</h1>
</body>
```

Rendered HTML

```
<html>
<head><title>Title here</title></head>
<body>
<h1>Welcome, Adrian!</h1>
</body>
```

title `Title here` name `Adrian`

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Putting it all together

Template (as written)

```
<html>
<head><title>{{ title }}</title></head>
<body>
<h1>Welcome, {{ name }}!</h1>
</body>
```

Rendered HTML

```
<html>
<head><title>Title here</title></head>
<body>
<h1>Welcome, Adrian!</h1>
</body>
```

title `Title here` name `Adrian`

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Ajax response

```
["document.title='Second page'",
"$('h1').html('Welcome, Bob!')"]
```

Putting it all together

Template (as written)

```
<html>
<head><title>{{ title }}</title></head>
<body>
<h1>Welcome, {{ name }}!</h1>
</body>
```

Rendered HTML

```
<html>
<head><title>Title here</title></head>
<body>
<h1>Welcome, Adrian!</h1>
</body>
```

title | Title here | name | Adrian

Ajax request

```
GET /page2/
X-pjax: /page1/
```

Ajax response

```
["document.title='Second page'",
"$('h1').html('Welcome, Bob!')"]
```

JS callback

```
document.title='Second page';
$('h1').html('Welcome, Bob!');
history.pushState(null, document.title, "/page2");
```

Security, part 1

Internal variable names

```
<html>
<body>

<div id="content">
Hello, {{ stupid_username }}
</div>
```

Security, part 2

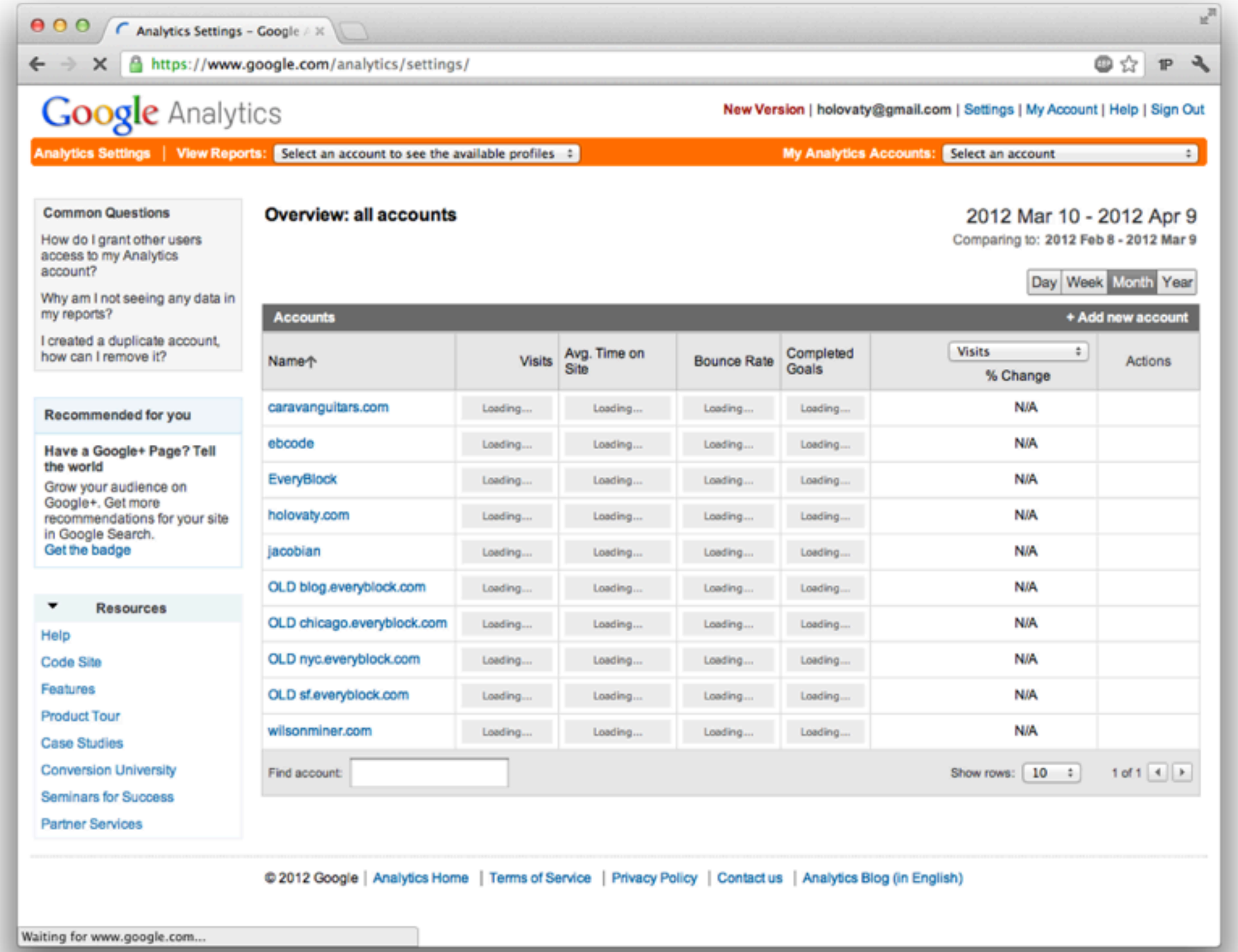
Auto-escaping

```
<html>
<body>

<div id="content">
Hello, {{ name }}
</div>
```

name | `<script>alert('hacked you');</script>`

Bonus!

Deferred loading



Deferred loading

```
<html>
<body>
<h1>{% block h1 %}{% endblock %}</h1>

<div id="content">
{% defer_load %}
{% block content %}{% endblock %}
{% end_defer_load %}
</div>
```

Template scanner

- Scan all templates in the codebase.
- Figure out which combinations are good/bad.
- Use that data to inform auto-pjax.

“This sounds like a horrible amount of work.”

adrian@holovaty.com
tinyurl.com/django-pjax