IPL
escola superior
de tecnologia e gestão
instituto politécnico
de leiria

Headers, Cookies, Sessions

HTTP messages and keeping state on PHP Web Applications

Vitor Carreira & Marco Monteiro

Contributors

- Author(s):
  - Vitor Carreira (vitor.carreira@ipleiria.pt)
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

- Contributor(s):
  - Fernando Silva(Fernando.silva@ipleiria.pt)

Summary

1. Headers
2. *State*
3. *Cookies*
4. *Sessions*
5. References

1 – HEADERS

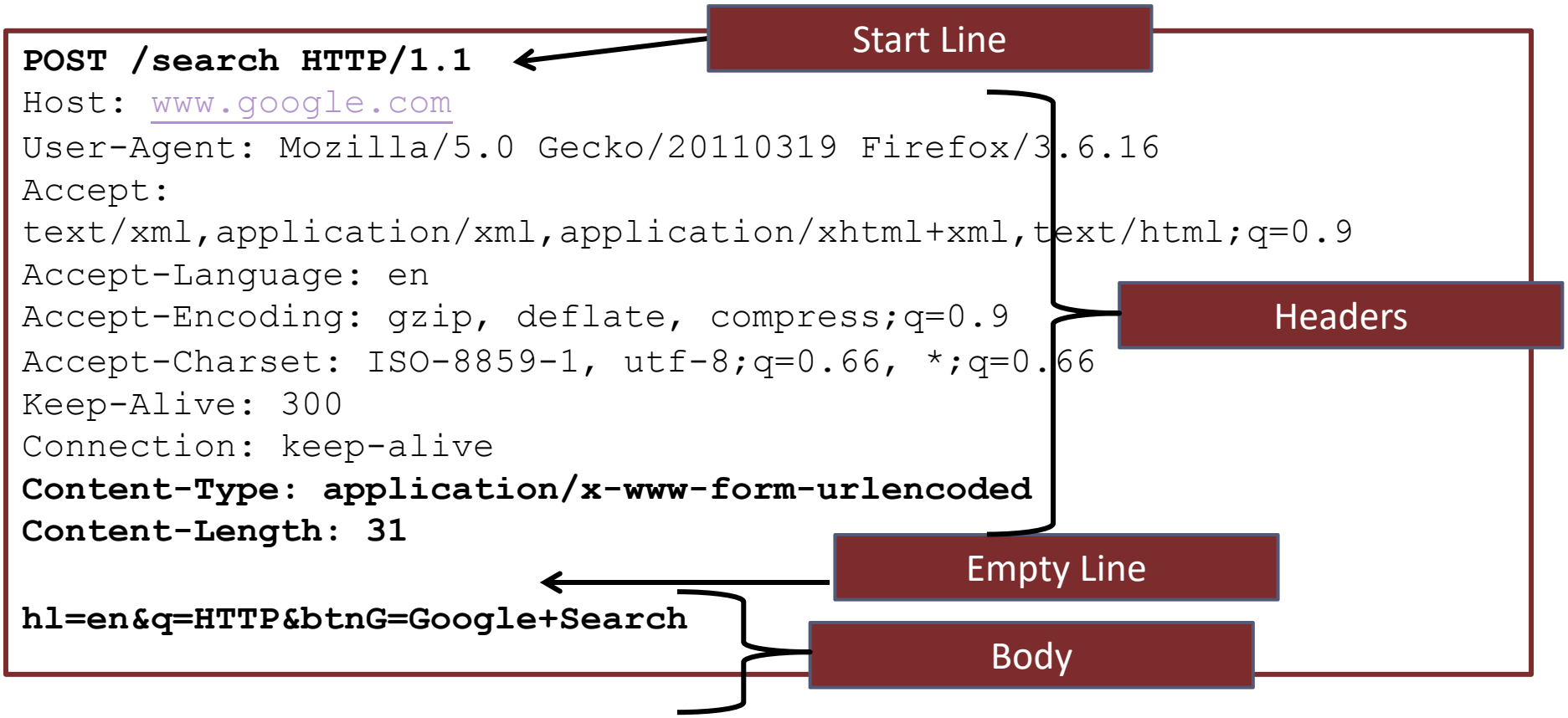
HTTP Protocol

- An HTTP transaction consists of a **request** command (sent from client to server), and a **response** result (sent from the server back to the client)

- An HTTP message has the following format:
  - **Start line** - Request or Response line
  - **HTTP headers** (optional)
  - **Empty line**
  - **Body** - Request / Response content

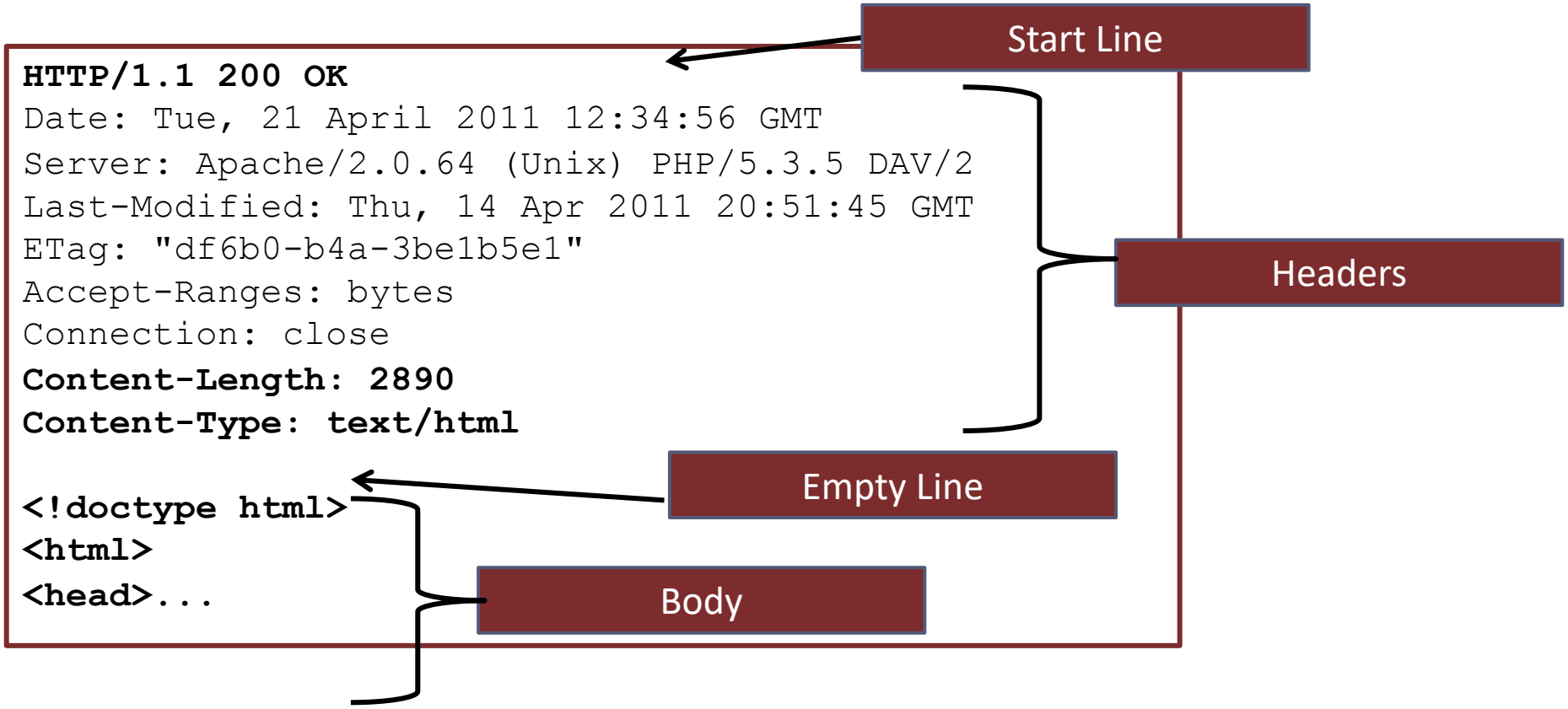
HTTP Request

- Request line format: <Request method (GET, POST, etc)> <URI> <HTTP version>
- Body is only present for POST or PUT requests



HTTP Response

- Response line format: <HTTP version supported> <Status code> <Status message>
- Content of the response contains the requested URI (HTML, CSS, binary file, etc)



HTTP Headers

- HTTP headers are used to pass additional information
- Header syntax: **<Name>:<Value>**
- There are four types of HTTP headers:
  - **General headers** - relative to the message being transmitted (e.g. Cache-Control, Connection, etc…)
  - **Entity headers** - relative to the content being transmitted (e.g. Content-Type, Content-Length, Content-Location, etc…)
  - **Request headers** - additional information about the request (e.g. Accept, Accept-Charset, Cookie, etc…)
  - **Response headers** - additional information about the response (Accept-Ranges, Location, Set-Cookie, Content-Disposition, etc…)

HTTP Headers

- If an HTTP message doesn’t have an empty body, the following headers SHOULD be present:
  - **Content-Type** - MIME type of the resource being transmitted (text/html, text/css, image/png, application/zip, etc…)
  - **Content-Length** - specifies the length of the body section of the HTTP message in bytes

PHP and HTTP Headers


```
header(string name_value [,bool replace [,int http_response_code]])
```

- The header function can be used to set/add additional HTTP headers to the PHP response
- The function MUST be called **before** sending **any output** to the client
  - The following are accountable as output:
  - Any kind of text (including whitespace) outside the php tag
  - Any PHP function that outputs text (echo, print, printf, print_r, etc…)

Headers: Redirect

- **Location** - used to achieve protocol-level redirection
  - According to the HTTP specification the value must be an absolute URL

```
<?php
  header('Location: http://www.site.com/newpage.php');
  exit(0);
?>
```

Headers: Refresh

- **Refresh** – allows to refresh a page automatically after some delay
  - Delay is given in seconds
  - Refreshes current page after 10 seconds:

```
<?php
  header('Refresh: 10; ');
?>
<html> . . .
```

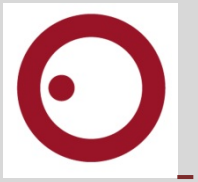
  - After 8 seconds, redirects to another page:

```
<?php
  header('Refresh: 8; url= http://www.a.com/new.php');
?>
<html> . . .
```

Headers: Content Disposition

- **Content-Disposition** - the most common use of this header is to force a filename for a file that should be saved rather than rendered

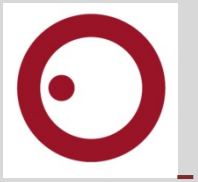
```
<?php
  header('Content-Type: application/pdf');
  // Indicates that user should be prompted to download
  // the file and that the pre-filled filename should be example.pdf

  header('Content-Disposition: attachment; filename="example.pdf"');

  header('Content-Transfer-Encoding: binary');

  $size = filesize('original.pdf');
  header("Content-Length: $size");

 // Outputs the file
  readfile('original.pdf');
  exit(0);
```



2 – STATE

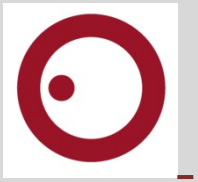
Extra class:

Just for informational purpose (not required for classes or evaluation)



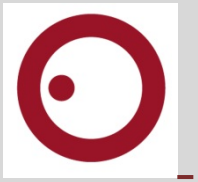
State and HTTP

- HTTP is a stateless protocol
- There is no built-in way of maintaining state between two transactions
- There is no automatic link or association between subsequent requests from the same user
- Example of why the state is important:
  - When implementing a Shopping Cart feature, the state (items in the shopping cart) needs to be maintained across requests.



State and HTTP

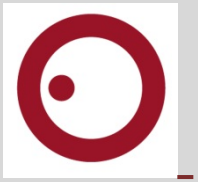
- How to keep state across requests?
  - Client-side: **cookies**
  - Server-side: **sessions**



3 – COOKIES

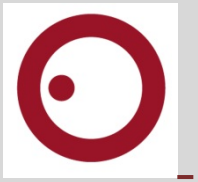
Extra class:

Just for informational purpose (not required for classes or evaluation)



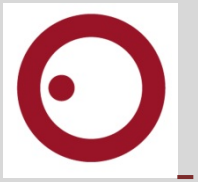
Cookies

- Cookie is a kind of variable (name-value pair) sent by the server on each request and it is stored on the client's web browser
- When the browser fetches a web page, it sends along with the request all cookies stored for the page's domain/path
- Cookies have attributes for:
  - domain and path - defines the **cookie scope**
  - expiration date - tells the browser when to delete the cookie
  - security - restricts the cookie's usage (secure connections only, http protocol only)



Cookies

- Limitations:
  - Browser's limits:
    - A per domain cookie limit that allows a single domain to only store x cookies before the oldest gets erased
    - A global cookie limit which erases the oldest cookies when the limit is reached
  - 4KB of maximum storage for each cookie (for maximum compatibility)
  - Users can delete or disable cookies



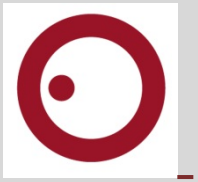
PHP Cookies

- Function `setcookie()` - sends a cookie to the client.

  This function must be called before any output is sent to the client

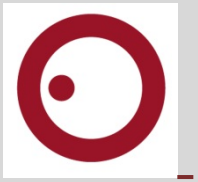
  setcookie <=> header("Set-Cookie: ..."))

- The superglobal associative array `$_COOKIE` keeps track of the cookies sent by the client



PHP Cookies


```
<?php
  $counter = $_COOKIE['counter'] ?? 0;
  $counter++;
  // set a cookie called counter. Cookie expires after 300s
  setcookie('counter', $counter, time() + 300);
?>
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>PHP: Cookies</title>
</head>
<body>
  <h1>Welcome. This is your visit #<?= $counter ?></h1>
</body>
</html>
```



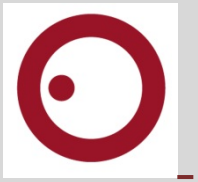
PHP Cookies – HTTP Messages

- First request headers (client => server)

```
Host: 127.0.0.1:8888
User-Agent: Mozilla/5.0 Gecko/20110319 Firefox/3.6.16
...
```

- First reply (response) headers (server => client)

```
...
Date: Tue, 12 Apr 2011 19:31:04 GMT
Server: Apache/2.0.64 (Unix) PHP/5.3.5 DAV/2
X-Powered-By: PHP/5.3.5
Set-Cookie: counter=1; expires= Tue,13-Mar-2018 19:36:05 GMT
Content-Length: 335
Content-Type: text/html
...
```



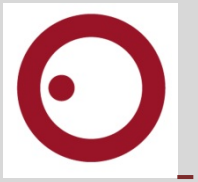
PHP Cookies – HTTP Messages

- Second request headers (client => server)

```
Host: 127.0.0.1:8888
User-Agent: Mozilla/5.0 Gecko/20110319 Firefox/3.6.16
Cookie: counter=1
...
```

- Second reply (response) headers (server => client)

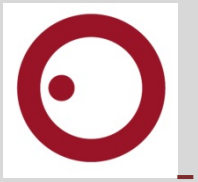
```
...
Date: Tue, 12 Apr 2011 19:31:04 GMT
Server: Apache/2.0.64 (Unix) PHP/5.3.5 DAV/2
X-Powered-By: PHP/5.3.5
Set-Cookie: counter=2; expires=Tue,13-Mar-2018 19:42:25 GMT
Content-Length: 335
Content-Type: text/html
...
```



4 – SESSIONS

Extra class:

Just for informational purpose (not required for classes or evaluation)

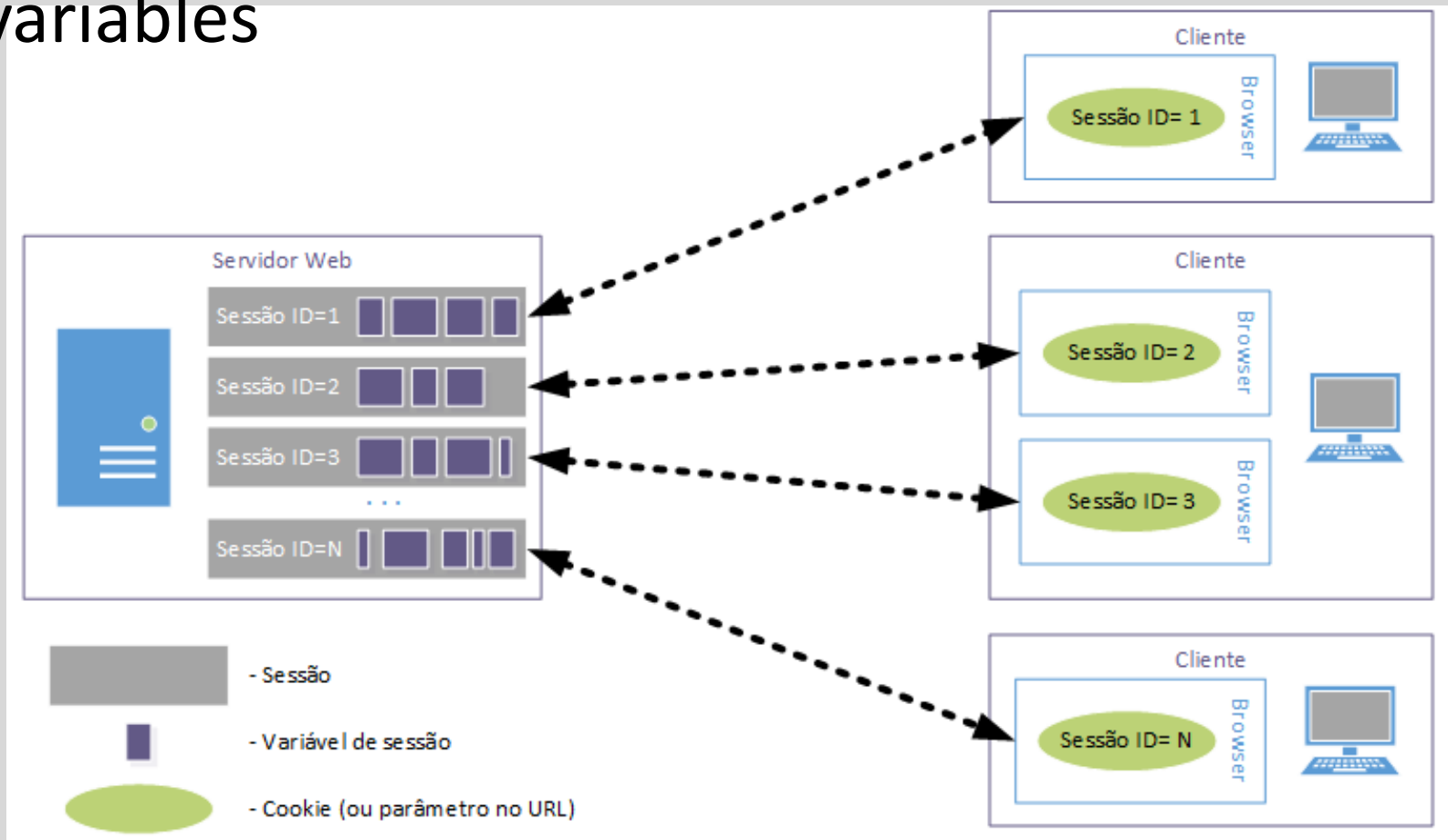


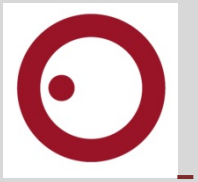
Sessions

- Session control allows a web server to track a user during a single session on a website
- The session ID is generated by the server and stored on the client side for the lifetime of a session. It can be either stored on a user's computer in a cookie or passed along through URLs
- The session ID acts as a key to register particular variables called session variables
- The **session variables are stored at the server** (flat file, database or in memory) and available to all pages of the Web Application
- A session has an implicit timeout, after which it is destroyed
- The session ID is the only information visible at the client side

Sessions

- Each session has its own set of session variables
- Al pages of the site have access to the session variables

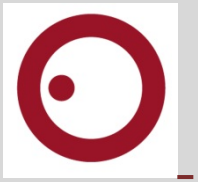




PHP Sessions

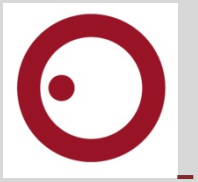
- Sessions in PHP are represent by a unique session ID (cryptographically 32 digit hexadecimal random number)
- The basic steps of using sessions are:
  - Starting a session
  - Registering (set) session variables
  - Using (read) session variables
  - Deregistering variables and destroying the session

  - *Note: these steps don't necessarily occur in the same script*



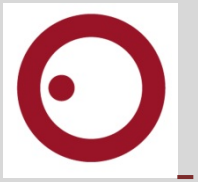
PHP Sessions

- Starting a session
  - Function `session_start()` - creates a session or resumes the current one
    - This function must be called before any output is sent to the client
- Registering (set) and using (read) session variables
  - Setting or reading values from the superglobal associative array: `$_SESSION`



PHP Sessions

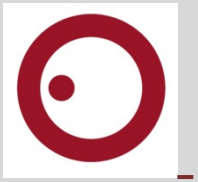

```php
<?php
  session_start();
  $counter = $_SESSION['counter'] ?? 0;
  $counter++;
  $_SESSION['counter'] = $counter;
?>
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>PHP: Cookies</title>
</head>
<body>
  <h1>Welcome. This is your visit #<?= $counter ?></h1>
</body>
</html>
```



PHP Sessions

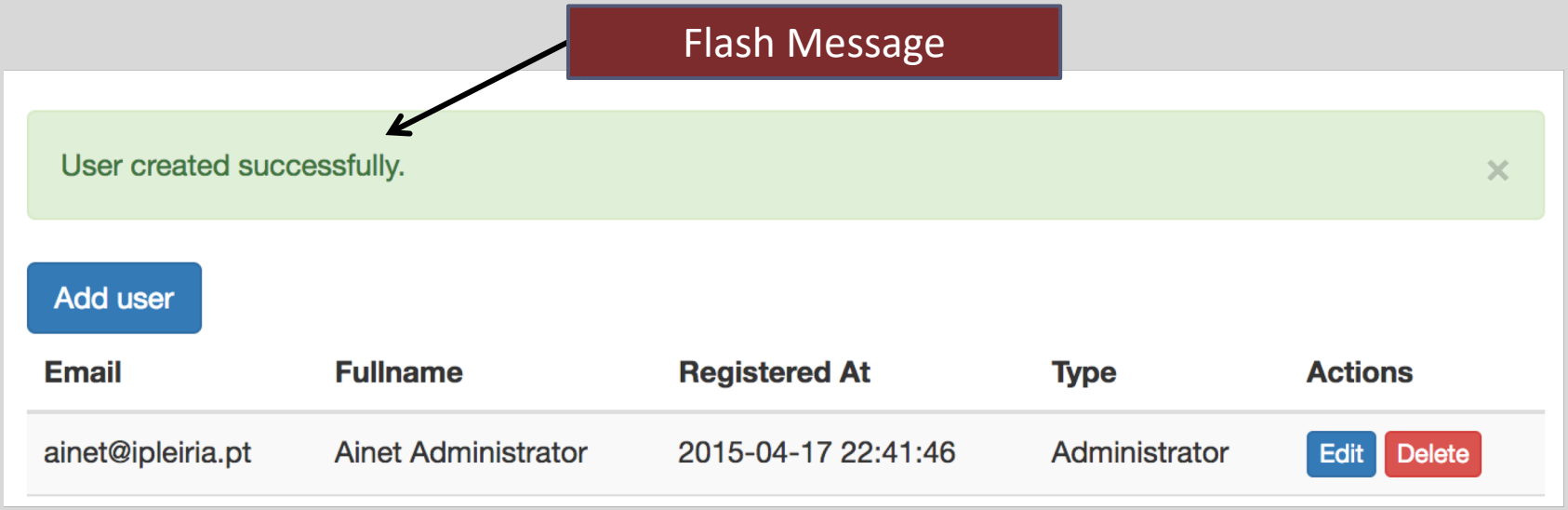
- Deregistering session variables
  - Destroy each session variable individually by calling `unset($_SESSION["var_name"])`
- Destroy the session
  - Destroy all session variable with `$_SESSION = array()`
  - At the end invalidate the session id by calling: `session_destroy()`

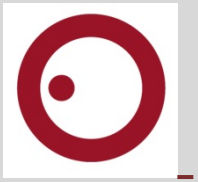
```
<?php
  // To destroy a session:
  session_start();
  $_SESSION = array();
  session_destroy();
?>
```



PHP Sessions: “Flash Messages”

- Flash messages: short-lived status messages
  - Only available for the next request
    - Usually the message is “emitted” on one page and “received” on another page
  - Typically is used to show users that an action was performed successfully or has failed.

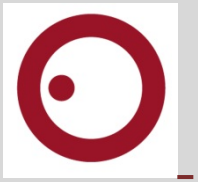




PHP Sessions: “Flash Messages”

- File where “flash message” is emitted
  - Usually after executing an operation
  - “Flash message” is stored on the Session

```
<?php
  session_start();
  // OPERATION
  $_SESSION['flashmessage'] = "Operation OK!";
  header("Location: otherfile.php");
```

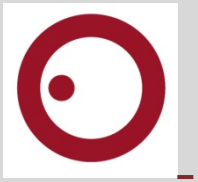


PHP Sessions: “Flash Messages”

- File that receives “flash message”
  - Reads the “flash message” from the Session
  - Deregister (removes) the “flashmessage” from the Session – guarantees message is only used once

otherfile.php:

```
<?php
  session_start();
  $flashmsg = $_SESSION['flashmessage'] ?? "";
  unset($_SESSION['flashmessage']);
  . . .
  if (!empty($flashmsg)) {
      echo $flashmsg;
  }
```



PHP Session

- Simple shopping cart example:
  - Check the provided demos

5 – REFERENCES

References

- Official (PHP)
  - http://www.php.net/
    - Cookies: http://www.php.net/manual/en/features.cookies.php
    - Sessions: http://www.php.net/manual/en/book.session.php
- PSR - PHP Standard Recommendations
  - https://www.php-fig.org
  - https://www.php-fig.org/psr/psr-4/
- Flash Messages
  - https://laravel.com/docs/5.6/session
- PHP and MySQL Web Development (4th Edition)
  - Luke Welling and Laura Thomson, Addison-Wesley 2009
- PHP Objects, Patterns, and Practice (2nd Edition)
  - Matt Zandstra, APress 2008
- Object Oriented PHP Concepts Techniques and Code
  - Peter Lavin, No Starch Press 2006