Using PHP in a Web Application

Internet Applications, ID1354

Contents

Cookies
HTTP Sessions
HTTP Parameters
Application Scope and File Handling
To Identify a List Item
Architecture

Section

- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket
- This is solved with cookies.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket
- This is solved with cookies.
- A cookie is a name/value pair passed between browser and server in the HTTP header.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket
- This is solved with cookies.
- A cookie is a name/value pair passed between browser and server in the HTTP header.
- A cookie is only passed to the server from which it originated.

To Set a Cookie

- Cookies are set with the **`setcookie`** function. Since cookies are sent as HTTP headers, this function must be called before any output is generated.

To Set a Cookie

- Cookies are set with the **`setcookie`** function. Since cookies are sent as HTTP headers, this function must be called before any output is generated.

```
setcookie (string $name, string $value,
           int $expire = 0, string $path,
           string $domain, bool $secure = false,
           bool $httponly = false)
```

To Set a Cookie

- Cookies are set with the **`setcookie`** function. Since cookies are sent as HTTP headers, this function must be called before any output is generated.

```
setcookie (string $name, string $value,
           int $expire = 0, string $path,
           string $domain, bool $secure = false,
           bool $httponly = false)
```

- **`name`** and **`value`** is the cookie's name/value pair.

To Set a Cookie

- Cookies are set with the **setcookie** function. Since cookies are sent as HTTP headers, this function must be called before any output is generated.

```
setcookie (string $name, string $value,
           int $expire = 0, string $path,
           string $domain, bool $secure = false,
           bool $httponly = false)
```

- **name** and **value** is the cookie's name/value pair.
- **expire** tells the instant in time when the cookie expires. **time()** returns the current time, so **time()+60*60*24*30** sets the cookie to expire in 30 days.

To Retrieve a Cookie

- Cookies are retrieved using the **`$_COOKIE`** superglobal, which is an array containing all cookies included in the current request.

To Retrieve a Cookie

- Cookies are retrieved using the **$_COOKIE** superglobal, which is an array containing all cookies included in the current request.
- The following statement retrieves all cookies with the name **userid**.

```
$_COOKIE["userid"];
```

To Retrieve a Cookie

- Cookies are retrieved using the **$_COOKIE** superglobal, which is an array containing all cookies included in the current request.
- The following statement retrieves all cookies with the name **userid**.

```
$_COOKIE["userid"];
```

- The **isset** function can be used to check if a cookie is set.

```
if (!isset($_COOKIE["userid"])) {
    echo '<a href="login.php">log in</a>';
}
```

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.
- If many servers set the same third party cookie, the third party server can track the user's surfing.

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.
- If many servers set the same third party cookie, the third party server can track the user's surfing.
- Typically used for marketing.

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.
- If many servers set the same third party cookie, the third party server can track the user's surfing.
- Typically used for marketing.
- There are many other ways, beside cookies, to identify a user for tracking purposes.
  - Flash, Silverlight and HTML5 storages
  - HTML5 canvas painting
  - content of caches and cache tags like Last-Modified or ETag
  - social networks
  - fingerprinting mechanisms like supported ciphersuites, DNS content, HTTP headers, plugins and fonts, clock drift, CPU and GPU benchmarks, network level information, user behavior

The EU Cookie Law

A person shall not store or gain access to information stored, in the terminal equipment of a subscriber or user unless the requirements of paragraph (2) are met.

(2) The requirements are that the subscriber or user of that terminal equipment

1. is provided with clear and comprehensive information about the purposes of the storage of, or access to, that information; and
2. has given his or her consent.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
  - Not relevant here.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
  - Not relevant here.
- The cookie is strictly necessary for the provision of an information society service requested by the subscriber or user.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
  - Not relevant here.
- The cookie is strictly necessary for the provision of an information society service requested by the subscriber or user.
  - Applies to authentication and shopping baskets?

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.
- It is not mandatory in any way to obey the users preferences.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.
- It is not mandatory in any way to obey the users preferences.
- Must be implemented by server developer.

Question 1

Section

- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

Sessions

- A session is the time span during which a particular browser interacts with a particular server.

Sessions

- A session is the time span during which a particular browser interacts with a particular server.
- For session tracking, PHP creates and maintains a session tracking id (Unique ID, UID), for each visitor and stores variables based on this UID.

Sessions

- A session is the time span during which a particular browser interacts with a particular server.
- For session tracking, PHP creates and maintains a session tracking id (Unique ID, UID), for each visitor and stores variables based on this UID.
- The UID is stored on the client, for example in a cookie or as part of URLs, and included in each request to the server.

Sessions

- A session is the time span during which a particular browser interacts with a particular server.
- For session tracking, PHP creates and maintains a session tracking id (Unique ID, UID), for each visitor and stores variables based on this UID.
- The UID is stored on the client, for example in a cookie or as part of URLs, and included in each request to the server.
- The only way to terminate a session is to manually unset all data related to the session in the server-side code.

Sessions

- A session is the time span during which a particular browser interacts with a particular server.
- For session tracking, PHP creates and maintains a session tracking id (Unique ID, UID), for each visitor and stores variables based on this UID.
- The UID is stored on the client, for example in a cookie or as part of URLs, and included in each request to the server.
- The only way to terminate a session is to manually unset all data related to the session in the server-side code.
- If a session is not explicitly terminated, it times out after an interval specified in server configuration, and session data is removed.

How is session data saved?

- We must understand that the lifetime of a PHP variable is limited to the execution of the program where it is created.

How is session data saved?

- We must understand that the lifetime of a PHP variable is limited to the execution of the program where it is created.
- This means that a variable created in one request will not exist in later requests.

How is session data saved?

- We must understand that the lifetime of a PHP variable is limited to the execution of the program where it is created.
- This means that a variable created in one request will not exist in later requests.
- Therefore, the content of **`$_SESSION`** must be stored externally to the PHP interpreter.

How is session data saved?

- We must understand that the lifetime of a PHP variable is limited to the execution of the program where it is created.
- This means that a variable created in one request will not exist in later requests.
- Therefore, the content of **`$_SESSION`** must be stored externally to the PHP interpreter.
- This storage is called a session save handler, and is configurable. Normally, and also normally by default, a file is used.

Session Management

- A session is started with the **`session_start`** function.

Session Management

- A session is started with the `session_start` function.
- To associate data with a session, use the **`$_SESSION`** superglobal.

Session Management

- A session is started with the `session_start` function.
- To associate data with a session, use the `$_SESSION` superglobal.
- To delete all data from the session, use the `session_destroy` function.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.
- This is normally done using a cookie.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.
- This is normally done using a cookie.
  - After **`session_start`** is called, PHP will look for a cookie named **`PHPSESSID`**.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.
- This is normally done using a cookie.
  - After **`session_start`** is called, PHP will look for a cookie named **`PHPSESSID`**.
  - If it is present, its value will be used as the id of the current session.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.
- This is normally done using a cookie.
  - After **`session_start`** is called, PHP will look for a cookie named **`PHPSESSID`**.
  - If it is present, its value will be used as the id of the current session.
  - If it is not present, it will be created and its value will be set to the id of the current session.

How is a session identified?

- To fill the **`$_SESSION`** superglobal with the current user's data, the session save handler must be able to identify the user.
- This is normally done using a cookie.
  - After **`session_start`** is called, PHP will look for a cookie named **`PHPSESSID`**.
  - If it is present, its value will be used as the id of the current session.
  - If it is not present, it will be created and its value will be set to the id of the current session.
- We must understand that the **`PHPSESSID`** cookie is the link between a browser and that browser's session data on the server.

Session Example

At session start

```
const USER_KEY = 'user_key';
session_start();
//Assuming $user is an object with user data.
$_SESSION[USER_KEY] = serialize($user);
```

Session Example

At session start

```
const USER_KEY = 'user_key';
session_start();
//Assuming $user is an object with user data.
$_SESSION[USER_KEY] = serialize($user);
```

During the session

```
if (isset($_SESSION[USER_KEY])) {
    $my_data = unserialize($_SESSION[USER_KEY]);
}
```

Session Example

At session start

```
const USER_KEY = 'user_key';
session_start();
//Assuming $user is an object with user data.
$_SESSION[USER_KEY] = serialize($user);
```

During the session

```
if (isset($_SESSION[USER_KEY])) {
    $my_data = unserialize($_SESSION[USER_KEY]);
}
```

At session end.

```
session_destroy();
```

Question 2

Section

- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

HTTP Parameters

- The **`$_GET`** and **`$_POST`** superglobals are used to retrieve HTTP parameters, for example user input in a form.

HTTP Parameters

- The **`$_GET`** and **`$_POST`** superglobals are used to retrieve HTTP parameters, for example user input in a form.
- **`$_GET`** is an array with all parameters in a HTTP GET request, **`$_POST`** is a similar array for a POST request.

HTTP Parameter Example

The following code retrieves the value of the **address** parameter, which might originate from an HTML form.

```
//The text field where the user types the address
//must have the attribute name='address'

const ADDRESS_KEY = 'address';
if (!empty($_POST[ADDRESS_KEY])) {
    $address = $_POST[ADDRESS_KEY];
}
```

Question 3

Section

- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

Application Scope Data

- As opposed to other server-side technologies, PHP does not have something like a **`$_SESSION`** superglobal that is shared between different users.

Application Scope Data

- As opposed to other server-side technologies, PHP does not have something like a **`$_SESSION`** superglobal that is shared between different users.
- If data is to be shared between different users, such a mechanism must be constructed.

Application Scope Data

- As opposed to other server-side technologies, PHP does not have something like a **`$_SESSION`** superglobal that is shared between different users.
- If data is to be shared between different users, such a mechanism must be constructed.
- A simple approach is to store data with application scope in a database.

Application Scope Data

- As opposed to other server-side technologies, PHP does not have something like a **`$_SESSION`** superglobal that is shared between different users.
- If data is to be shared between different users, such a mechanism must be constructed.
- A simple approach is to store data with application scope in a database.
- Other alternatives are a text file, an xml file or a plug-in such as memcached, `http://www.memcached.org/`, which stores key/value pairs in memory.

File Handling

- Simple file handling can be done with **`file_put_contents`**, which writes to a file, and **`file_get_contents`**, which reads.

File Handling

- Simple file handling can be done with **`file_put_contents`**, which writes to a file, and **`file_get_contents`**, which reads.

```
\file_put_contents($path_to_file,
                   $data, FILE_APPEND);
```

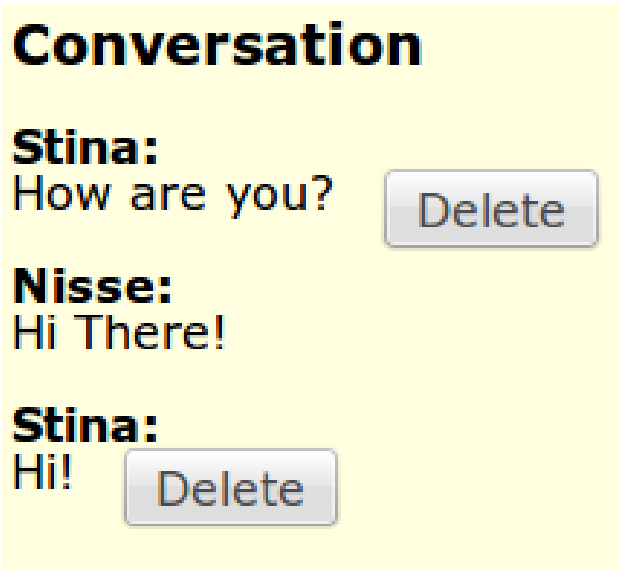
```
\file_get_contents($path_to_file));
```

Question 4

Section

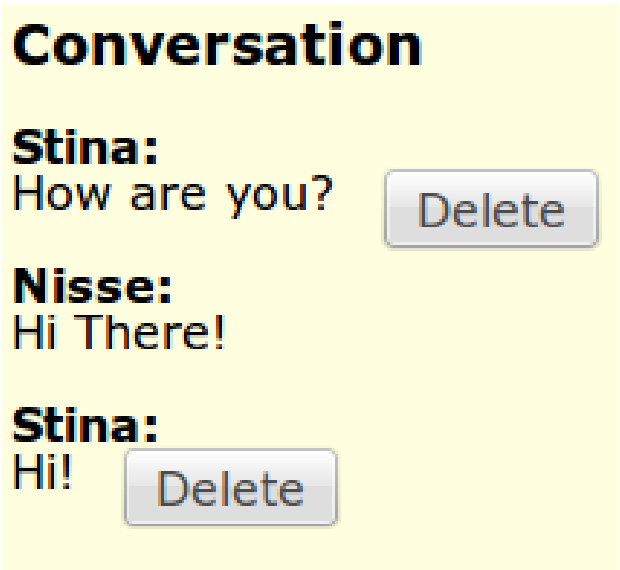
- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

The Problem



- There is a list with buttons (or links) for multiple items, like the chat application example to the left.

The Problem



- There is a list with buttons (or links) for multiple items, like the chat application example to the left.
- How can we know which button the user clicked? In this chat example, how can we know which entry Stina wants to delete?

The Solution, Hidden Field

- Make a form for each item in the list.

The Solution, Hidden Field

- Make a form for each item in the list.
  - In this chat example, that means one form for each entry that has a **`Delete`** button.

The Solution, Hidden Field

- Make a form for each item in the list.
  - In this chat example, that means one form for each entry that has a `Delete` button.
- Each form includes a hidden field, which holds an identifier for the list item where the form is placed.

The Solution, Hidden Field

- Make a form for each item in the list.
  - In this chat example, that means one form for each entry that has a `Delete` button.
- Each form includes a hidden field, which holds an identifier for the list item where the form is placed.
  - In this example, we use the time when the entry was written as identifier.

The Solution, Hidden Field

- A hidden field is not displayed in the browser, but included when the form is submitted.

The Solution, Hidden Field

- A hidden field is not displayed in the browser, but included when the form is submitted.
  - The HTML for the chat conversation is listed below.

```
<p class="author">Stina:</p>
<p class="entry">How are you?</p>
<form action="delete-entry.php">
    <input type="hidden" value="1439884094" name="timestamp">
    <input type="submit" value="Delete">
</form>
<p class="author">Nisse:</p>
<p class="entry">Hi There!</p>
<p class="author">Stina:</p>
<p class="entry">Hi!</p>
<form action="delete-entry.php">
    <input type="hidden" value="1439884028" name="timestamp">
    <input type="submit" value="Delete">
</form>
```

The Solution, Server Code

- On the server, we simply read the timestamp of the submitted form and delete the entry with that timestamp.

The Solution, Server Code

- On the server, we simply read the timestamp of the submitted form and delete the entry with that timestamp.
- Code is not complete, just illustrates the principle. Complete code is found on course web page.

```
for ($i = count($entries) - 1; $i >= 0; $i--) {
    $entry = unserialize($entries[$i]);
    if ($entry->getTimestamp() ==
        $_GET[CHAT_TIMESTAMP_KEY]) {
        $entry->setDeleted(true);
        $entries[$i] = serialize($entry);
        break;
    }
}
```

Question 5

Section

- Cookies
- HTTP Sessions
- HTTP Parameters
- Application Scope and File Handling
- To Identify a List Item
- Architecture

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.
- Low coupling, Objects and subsystems do not depend on each other more than necessary.

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.
- Low coupling, Objects and subsystems do not depend on each other more than necessary.
- Encapsulation, Objects and subsystems do not reveal their internals.

A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.

A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.

A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.
- This means using one PHP file for each possible HTTP request.

A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.
- This means using one PHP file for each possible HTTP request.
- However, handling everything related to a particular HTTP request in a separate file has big disadvantages:

A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.
- This means using one PHP file for each possible HTTP request.
- However, handling everything related to a particular HTTP request in a separate file has big disadvantages:
  - Low cohesion since that file will do everything.

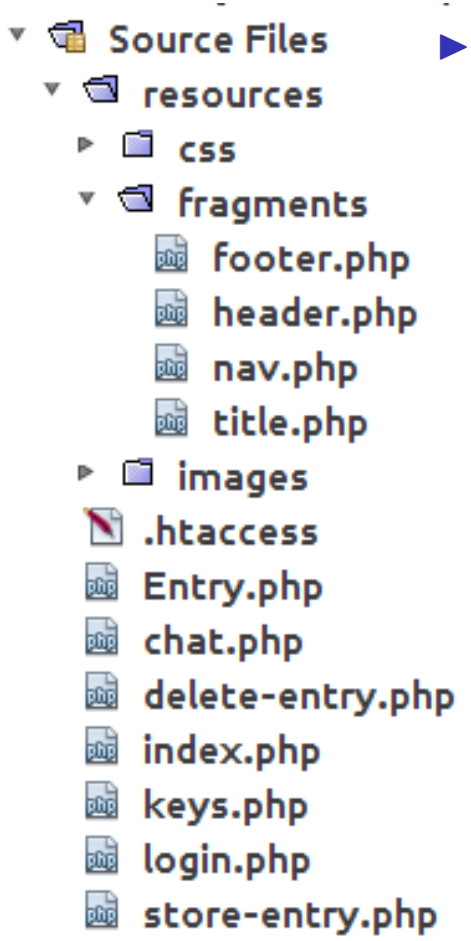
A Very Simple Architecture

- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.
- This means using one PHP file for each possible HTTP request.
- However, handling everything related to a particular HTTP request in a separate file has big disadvantages:
  - Low cohesion since that file will do everything.
  - High coupling since code for view handling, database access, etc, will be placed in the same file.

A Very Simple Architecture

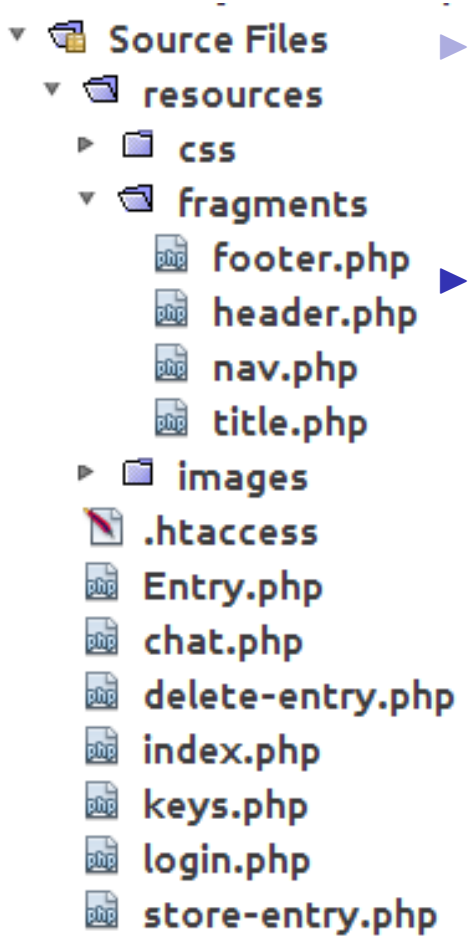
- Server-side architecture is covered extensively later in the course.
- For now, we will use a very simple architecture.
- This means using one PHP file for each possible HTTP request.
- However, handling everything related to a particular HTTP request in a separate file has big disadvantages:
  - Low cohesion since that file will do everything.
  - High coupling since code for view handling, database access, etc, will be placed in the same file.
  - Duplicated code since similar code will appear in several such files.

A *Slightly* Better Architecture



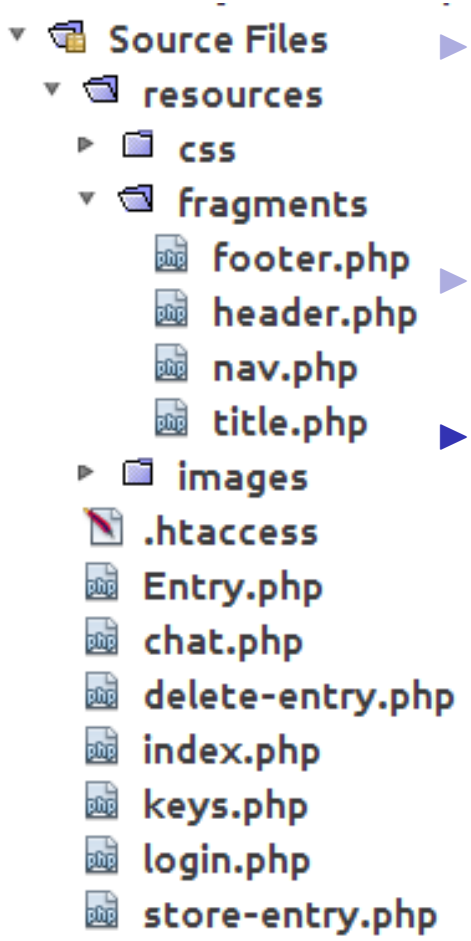
- Fragments (header, footer, etc) are placed in a separate directory and included in each page.

A *Slightly* Better Architecture



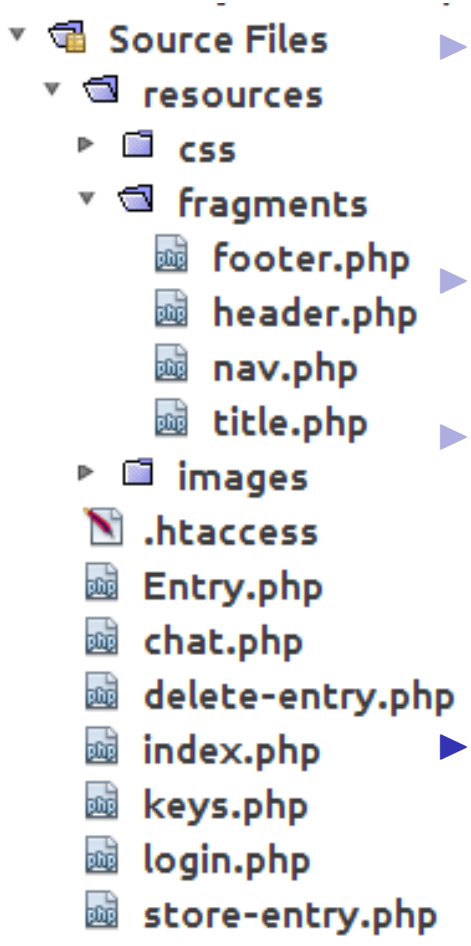
- Fragments (header, footer, etc) are placed in a separate directory and included in each page.
- View (HTML code) is placed in separate files, **`chat.php`** and **`index.php`**.

A *Slightly* Better Architecture



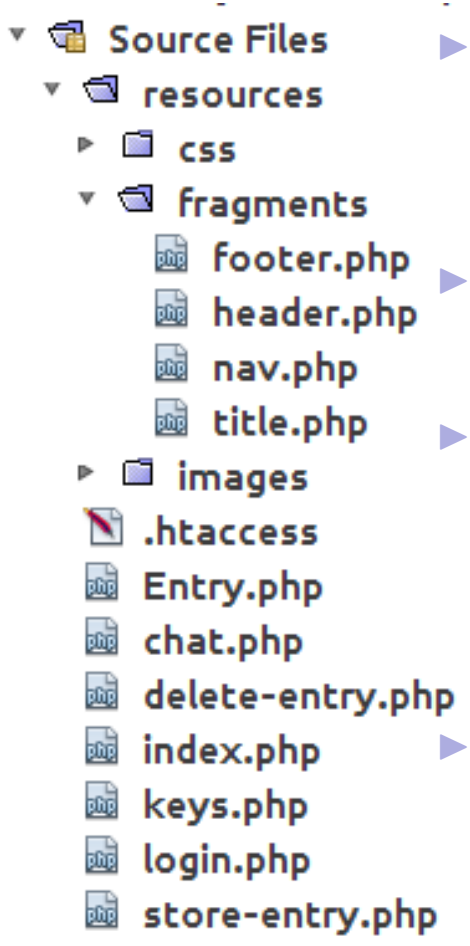
- Fragments (header, footer, etc) are placed in a separate directory and included in each page.
- View (HTML code) is placed in separate files, **`chat.php`** and **`index.php`**.
- **`Entry.php`** is a class that represents an entry in the conversation. It is included where needed in the HTTP request handling PHP files.

A *Slightly* Better Architecture



- Fragments (header, footer, etc) are placed in a separate directory and included in each page.
- View (HTML code) is placed in separate files, **`chat.php`** and **`index.php`**.
- **`Entry.php`** is a class that represents an entry in the conversation. It is included where needed in the HTTP request handling PHP files.
- **`keys.php`** holds some constants that are used in multiple places. It is included where needed in the HTTP request handling PHP files.

A *Slightly* Better Architecture



- Fragments (header, footer, etc) are placed in a separate directory and included in each page.
- View (HTML code) is placed in separate files, **`chat.php`** and **`index.php`**.
- **`Entry.php`** is a class that represents an entry in the conversation. It is included where needed in the HTTP request handling PHP files.
- **`keys.php`** holds some constants that are used in multiple places. It is included where needed in the HTTP request handling PHP files.

- The files handling HTTP requests are **`login.php`**, **`store-entry.php`** and **`delete-entry.php`**