Using PHP in Web Applications

Internet Applications, ID1354

Contents

Section

- Server-Side Tasks
  - Cookies
  - HTTP Sessions
  - HTTP Parameters
  - Application Scope and File Handling
- AJAX
- Server-Side Architecture
- The Flight Framework

Section

- Server-Side Tasks
  - Cookies
  - HTTP Sessions
  - HTTP Parameters
  - Application Scope and File Handling
- AJAX
- Server-Side Architecture
- The Flight Framework

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, for example IP address, installed software, fingerprinting browser information, social networks, pixel placement + url rewriting, etc.

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.
  - Likely 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.

Section

- Server-Side Tasks
  - Cookies
  - HTTP Sessions
  - HTTP Parameters
  - Application Scope and File Handling
- AJAX
- Server-Side Architecture
- The Flight Framework

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.

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.

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.
- To stop a malicious user from faking a cookie with a session id, you have to keep track of valid sessions on the server.

How does it work?

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

How does it work?

- 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 does it work?

- 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 does it work?

- 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 Example

At session start

```
const USER_KEY = 'user_key';
session_start();
//Assuming $user is an object with user data.
$_SESSION[USER_KEY] = serialize($user);
// If necessary to stop faked sessions.
add_to_my_valid_sessions(session_id());
```

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);
// If necessary to stop faked sessions.
add_to_my_valid_sessions(session_id());
```

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);
// If necessary to stop faked sessions.
add_to_my_valid_sessions(session_id());
```

During the session

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

At session end.

```
// If keeping track of valid sessions.
remove_from_my_valid_sessions(session_id());
session_destroy();
```

Section

- Server-Side Tasks
  - Cookies
  - HTTP Sessions
  - HTTP Parameters
  - Application Scope and File Handling
- AJAX
- Server-Side Architecture
- The Flight Framework

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 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.
- User input should be validated with JavaScript on the browser, since client-side validation is fast and reduces server load.

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.
- User input should be validated with JavaScript on the browser, since client-side validation is fast and reduces server load.
- Server-side validation is also needed since user might turn off JavaScript.

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 (isset($_POST[ADDRESS_KEY])) {
    $address = $_POST[ADDRESS_KEY];
}
```

Section

- Server-Side Tasks
  - Cookies
  - HTTP Sessions
  - HTTP Parameters
  - Application Scope and File Handling
- AJAX
- Server-Side Architecture
- The Flight Framework

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 file.

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 file.
- Other alternatives are a database, 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));
```

Section

- Server-Side Tasks
- AJAX
- Server-Side Architecture
- The Flight Framework

Loading an Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button.

Loading an Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button.
- Here, to load an entire page means that all HTML in the page is read from the server.

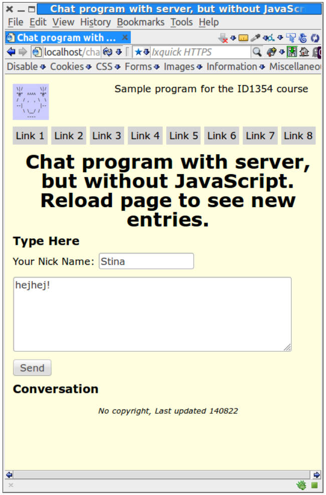
Loading an Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button.
- Here, to load an entire page means that all HTML in the page is read from the server.
- Dynamic data is included on the server, before the HTML is sent to the client, for example using a PHP program.

Loading an Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button.
- Here, to load an entire page means that all HTML in the page is read from the server.
- Dynamic data is included on the server, before the HTML is sent to the client, for example using a PHP program.
- This behavior is appropriate if the entire page content really must change, but that is often not the case.

Loading an Entire Page


Chat program with server, but without JavaScr

Sample program for the ID1354 course

Link 1 Link 2 Link 3 Link 4 Link 5 Link 6 Link 7 Link 8

Chat program with server, but without JavaScript. Reload page to see new entries.

Type Here

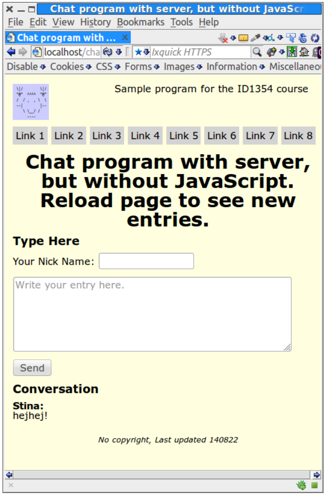
Your Nick Name: Stina

hejhej!

Send

Conversation

No copyright, Last updated 140822


Chat program with server, but without JavaScr

Sample program for the ID1354 course

Link 1 Link 2 Link 3 Link 4 Link 5 Link 6 Link 7 Link 8

Chat program with server, but without JavaScript. Reload page to see new entries.

Type Here

Your Nick Name:

Write your entry here.

Send

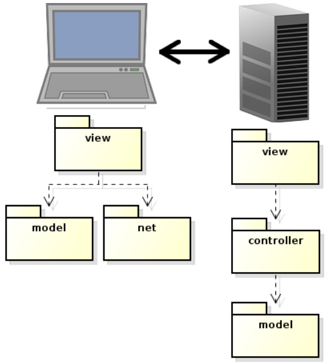
Conversation

Stina:
hejhej!

No copyright, Last updated 140822

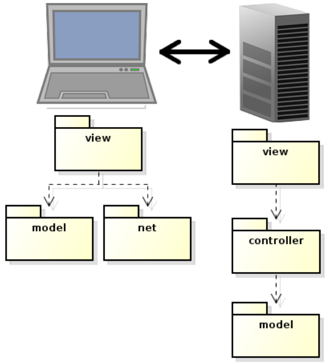
Consider for example the sample chat application. All that happens when the user clicks **`Send`** is that the new entry is added, the rest of the page is untouched.

Repetition: The MVVM Pattern



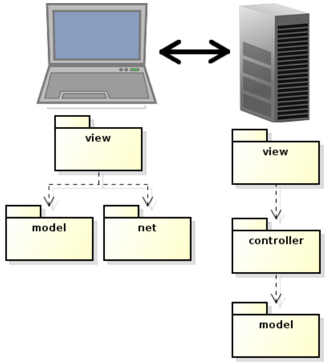
- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.

Repetition: The MVVM Pattern



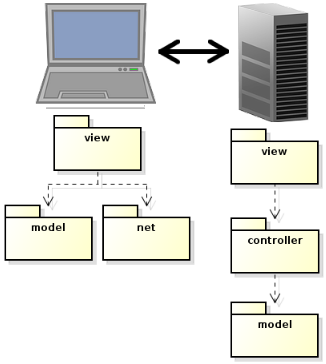
- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.

Repetition: The MVVM Pattern



- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.
- Therefore, the viewmodel will always contain the current state of the application.

Repetition: The MVVM Pattern



- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.
- Therefore, the viewmodel will always contain the current state of the application.
- The browser view must reflect the viewmodel state, preferably using the observer pattern.

AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.

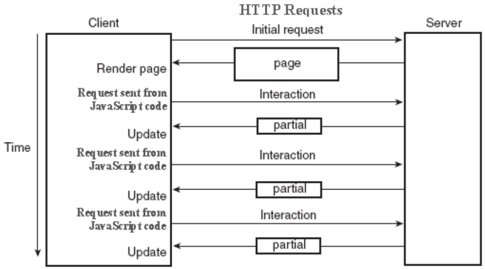
AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.
- AJAX is basically a way to use existing technologies, such as JavaScript, HTTP and XML.
  - No new language or markup.

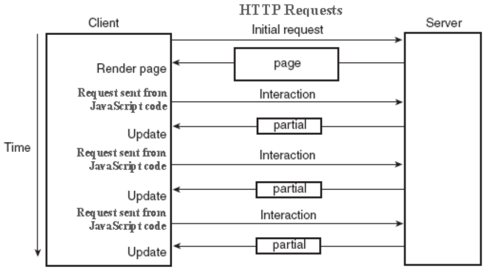
AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.
- AJAX is basically a way to use existing technologies, such as JavaScript, HTTP and XML.
  - No new language or markup.
- The only thing specific for AJAX is a JavaScript object, called **`XMLHttpRequest`**, which is standardized by W3C.

How Does It Work?

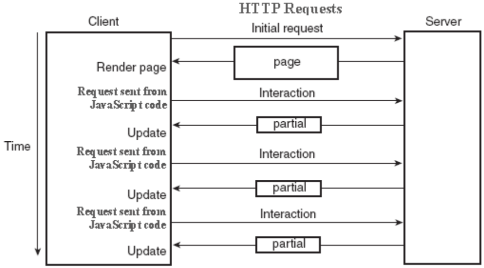


How Does It Work?



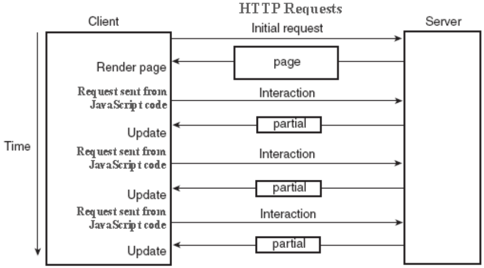
- Web page is loaded only on first request.

How Does It Work?



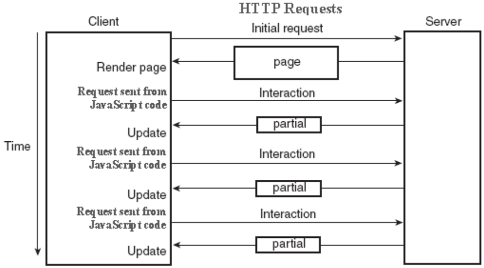
- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using **`XMLHttpRequest`**.

How Does It Work?



- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using `XMLHttpRequest`.
- The server returns only data, no markup.

How Does It Work?



- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using **`XMLHttpRequest`**.
- The server returns only data, no markup.
- Returned data is available to JavaScript code, and is used to update the web page, by updating the DOM.

How Does It Work? (Cont'd)

- Note that AJAX requests are ordinary HTTP requests, using ordinary HTTP methods.

How Does It Work? (Cont'd)

- Note that AJAX requests are ordinary HTTP requests, using ordinary HTTP methods.
- The server directs the request to the specified resource, just as when loading a HTML document.

How Does It Work? (Cont’d)

- Note that AJAX requests are ordinary HTTP requests, using ordinary HTTP methods.
- The server directs the request to the specified resource, just as when loading a HTML document.
  - Actually, the server can not tell who issued the request.

How Does It Work? (Cont’d)

- Note that AJAX requests are ordinary HTTP requests, using ordinary HTTP methods.
- The server directs the request to the specified resource, just as when loading a HTML document.
  - Actually, the server can not tell who issued the request.
- An AJAX request is normally handled by a program, for example PHP, which generates a response containing the new data.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- XML is an obvious option, but it has some drawbacks:

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- XML is an obvious option, but it has some drawbacks:
    - Interpreting an XML document requires extra code.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- XML is an obvious option, but it has some drawbacks:
  - Interpreting an XML document requires extra code.
  - Using a XSLT stylesheet to generate a view is a bit tricky.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- XML is an obvious option, but it has some drawbacks:
  - Interpreting an XML document requires extra code.
  - Using a XSLT stylesheet to generate a view is a bit tricky.
  - XML documents are quite long and wordy.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- XML is an obvious option, but it has some drawbacks:
  - Interpreting an XML document requires extra code.
  - Using a XSLT stylesheet to generate a view is a bit tricky.
  - XML documents are quite long and wordy.
- Therefore, **J**ava**S**cript **O**bject **N**otation, JSON, is normally used instead of XML.
  - Compact and easy to translate to JavaScript objects.

JSON

- The JSON syntax is very simple:

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

```
"firstName":"Olle"
```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with { and }:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with **{** and **}**:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

  - Arrays are denoted with **[** and **]**:

    ```
    "employees":[
      {"firstName":"Olle", "lastName":"Olsson"},
      {"firstName":"Stina", "lastName":"Nilsson"}
    ]
    ```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with `{` and `}`:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

  - Arrays are denoted with `[` and `]`:

    ```
    "employees":[
      {"firstName":"Olle", "lastName":"Olsson"},
      {"firstName":"Stina", "lastName":"Nilsson"}
    ]
    ```

  - Data types are JavaScript types, for example string, **“abcd”**; integer, **`123`**; boolean, **`false`**

JSON is not an Alternative to XML

- Note that JSON is not a general alternative to XML. There is nothing like namespace, DTD, Schema, XSLT or anything else of all the XML standards.

JSON is not an Alternative to XML

- Note that JSON is not a general alternative to XML. There is nothing like namespace, DTD, Schema, XSLT or anything else of all the XML standards.
- JSON is just a format suitable for transferring JavaScript values.

The Client: jQuery AJAX Methods

- Instead of covering the **`XMLHttpRequest`** object, we will look at some convenient jQuery functions.

The Client: jQuery AJAX Methods

- Instead of covering the **`XMLHttpRequest`** object, we will look at some convenient jQuery functions.
- **`getJSON`** sends a HTTP GET request. Data can be included as a query string and the response is parsed as JSON data.

The Client: jQuery AJAX Methods

- Instead of covering the **`XMLHttpRequest`** object, we will look at some convenient jQuery functions.
- **`getJSON`** sends a HTTP GET request. Data can be included as a query string and the response is parsed as JSON data.
-
```
$.getJSON(url, "reqData=" + someVariable,
          function(returnedData) {
            //Handle returnedData, which is
            //the received JSON data, parsed
            //to a JavaScript variable.
          });
```

  An HTTP GET request is sent to the URL specified in **`url`**. The request has the query string **`reqData=<value of someVariable>`** and the anonymous callback function is executed when the server's response arrives.

jQuery AJAX Methods (Cont'd)

- **`post`** sends data with a HTTP POST request.

jQuery AJAX Methods (Cont'd)

- `post` sends data with a HTTP POST request.
- `$.post(url, "data=" + someVariable);`

  An HTTP POST request is sent to the URL specified in **`url`**. The request has the body **`data=<value of someVariable>`**.

jQuery AJAX Methods (Cont'd)

- `post` sends data with a HTTP POST request.
- `$.post(url, "data=" + someVariable);`

  An HTTP POST request is sent to the URL specified in `url`. The request has the body `data=<value of someVariable>`.
- jQuery has many more AJAX methods.

The Server: JSON handling in PHP

- Remember that an AJAX request is a normal HTTP request.

The Server: JSON handling in PHP

- Remember that an AJAX request is a normal HTTP request.
- Therefore, to handle an AJAX request is no different from other request handling.

The Server: JSON handling in PHP

- Remember that an AJAX request is a normal HTTP request.
- Therefore, to handle an AJAX request is no different from other request handling.
- What is specific for AJAX interaction, is that we have to generate a JSON response.

The Server: JSON handling in PHP

- Remember that an AJAX request is a normal HTTP request.
- Therefore, to handle an AJAX request is no different from other request handling.
- What is specific for AJAX interaction, is that we have to generate a JSON response.

- `json_encode($aPhpObject)`

  The **`json_encode`** PHP method encodes the PHP object in **`aPhpObject`** to JSON representation.

JSON handling in PHP (Cont'd)

```
class SomeClass implements \JsonSerializable {
   private $some_var;
   ...
   public function jsonSerialize() {
      $json_obj = new \stdClass;
      $json_obj->someVar = $this->some_var;
      ...
      return $json_obj;
   }
```

JSON handling in PHP (Cont'd)

```
class SomeClass implements \JsonSerializable {
   private $some_var;
   ...
   public function jsonSerialize() {
      $json_obj = new \stdClass;
      $json_obj->someVar = $this->some_var;
      ...
      return $json_obj;
   }
```

- The object that shall be JSON encoded must be of a class that implements **`JsonSerializable`**.

JSON handling in PHP (Cont'd)

```
class SomeClass implements \JsonSerializable {
   private $some_var;
   ...
   public function jsonSerialize() {
      $json_obj = new \stdClass;
      $json_obj->someVar = $this->some_var;
      ...
      return $json_obj;
   }
```

- The object that shall be JSON encoded must be of a class that implements **`JsonSerializable`**.
- That class must have a method called **`jsonSerialize`**, which returns an object containing all relevant fields.

JSON handling in PHP (Cont'd)

```
class SomeClass implements \JsonSerializable {
   private $some_var;
   ...
   public function jsonSerialize() {
      $json_obj = new \stdClass;
      $json_obj->someVar = $this->some_var;
      ...
      return $json_obj;
   }
```

- The object that shall be JSON encoded must be of a class that implements **`JsonSerializable`**.
- That class must have a method called **`jsonSerialize`**, which returns an object containing all relevant fields.
- That returned object must be possible to encode with **`json_encode`**.

AJAX Security

- AJAX security issues arise primarily for two reasons.

AJAX Security

- AJAX security issues arise primarily for two reasons.

1. The client inserts data from server into the DOM. An attacker may place malicious JavaScript in this data, which is executed on the client without the user knowing.

AJAX Security

- AJAX security issues arise primarily for two reasons.

1. The client inserts data from server into the DOM. An attacker may place malicious JavaScript in this data, which is executed on the client without the user knowing.
2. An attacker can submit malicious data in an AJAX call. This threat is not AJAX specific, but AJAX often drastically increases the number of requests.This creates new doors on the server, which must all be protected.

AJAX Security (Cont'd)

Here are a few brief basic rules for mitigating these threats.

AJAX Security (Cont'd)

Here are a few brief basic rules for mitigating these threats.

1. Only insert data in the content of HTML body elements like **`div`**, **`p`**, **`td`**, etc.

AJAX Security (Cont’d)

Here are a few brief basic rules for mitigating these threats.

1. Only insert data in the content of HTML body elements like **div**, **p**, **td**, etc.
2. Convert dangerous characters in received data, as specified below, before inserting any data at all.

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
```

AJAX Security (Cont'd)

Here are a few brief basic rules for mitigating these threats.

1. Only insert data in the content of HTML body elements like `div`, `p`, `td`, etc.
2. Convert dangerous characters in received data, as specified below, before inserting any data at all.

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
```

3. All security controls for submitted data must be performed on the server. Examples of such controls are authentication (log in) and data validation.

AJAX Security (Cont'd)

- Much more is needed to build a safe web application.

eet

AJAX Security (Cont'd)

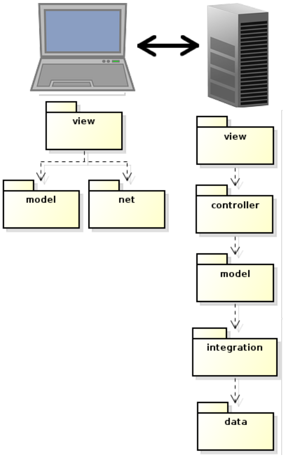
- Much more is needed to build a safe web application.
- Good reading from Open Web Application Security Project (OWASP)
  - https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

AJAX Security (Cont'd)

- Much more is needed to build a safe web application.
- Good reading from Open Web Application Security Project (OWASP)
  - https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
- And from Symantec
  - http://www.symantec.com/connect/articles/ajax-security-basics

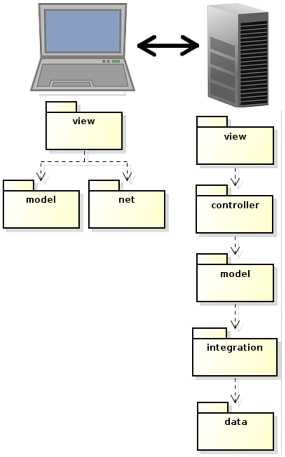
Section

Remember: Server-Side Layers



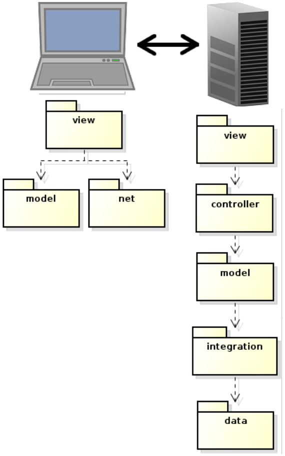
- The server has the same layers as a stand-alone MVC architecture.

Remember: Server-Side Layers



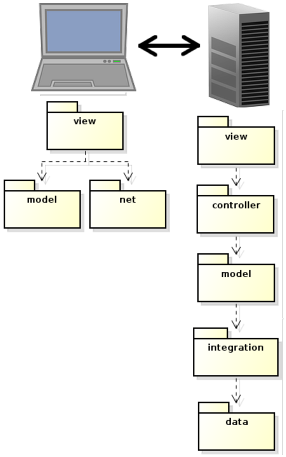
- The server has the same layers as a stand-alone MVC architecture.
- The server's view layer gets HTTP requests and creates HTML/JSON responses.

Remember: Server-Side Layers



- The server has the same layers as a stand-alone MVC architecture.
- The server's view layer gets HTTP requests and creates HTML/JSON responses.
- The MVC pattern states that all UI related code shall be in the view. From controller and down there is only plain object-oriented code.

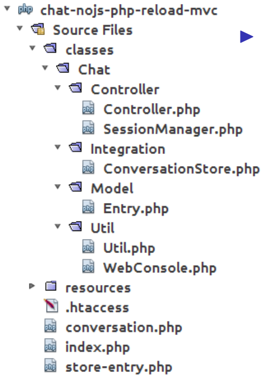
Remember: Server-Side Layers



- The server has the same layers as a stand-alone MVC architecture.
- The server's view layer gets HTTP requests and creates HTML/JSON responses.
- The MVC pattern states that all UI related code shall be in the view. From controller and down there is only plain object-oriented code.

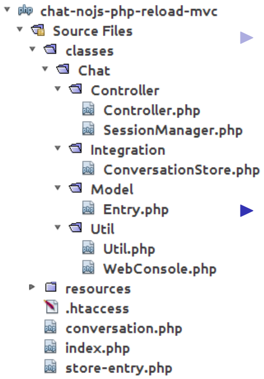
- This means that controller and lower layers are coded exactly as for a stand-alone application. Only the view is specific for a web application.

File Structure


```
▾ chat-nojs-php-reload-mvc
  ▾ Source Files
    ▾ classes
      ▾ Chat
        ▾ Controller
            Controller.php
            SessionManager.php
        ▾ Integration
            ConversationStore.php
        ▾ Model
            Entry.php
        ▾ Util
            Util.php
            WebConsole.php
    ▸ resources
      .htaccess
      conversation.php
      index.php
      store-entry.php
```

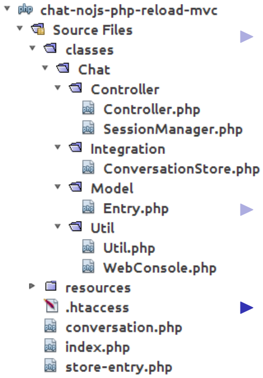
- It is a good practice to organize server-side code as in a Java application. One file per class and one directory per namespace.

File Structure


```
▾ chat-nojs-php-reload-mvc
  ▾ Source Files
    ▾ classes
      ▾ Chat
        ▾ Controller
            Controller.php
            SessionManager.php
        ▾ Integration
            ConversationStore.php
        ▾ Model
            Entry.php
        ▾ Util
            Util.php
            WebConsole.php
    ▸ resources
      .htaccess
      conversation.php
      index.php
      store-entry.php
```

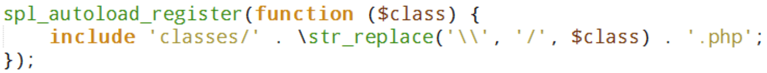
- It is a good practice to organize server-side code as in a Java application. One file per class and one directory per namespace.
- Place all classes in a separate directory, for example `classes`.

File Structure



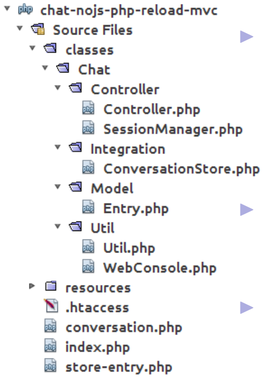
```
▾ chat-nojs-php-reload-mvc
  ▾ Source Files
    ▾ classes
      ▾ Chat
        ▾ Controller
            Controller.php
            SessionManager.php
        ▾ Integration
            ConversationStore.php
        ▾ Model
            Entry.php
        ▾ Util
            Util.php
            WebConsole.php
    ▸ resources
      .htaccess
      conversation.php
      index.php
      store-entry.php
```

- It is a good practice to organize server-side code as in a Java application. One file per class and one directory per namespace.
- Place all classes in a separate directory, for example `classes`.
- This enables autoloading classes, see below. We are relieved of **`include`** and **`require`** statements.



```
spl_autoload_register(function ($class) {
    include 'classes/' . \str_replace('\\', '/', $class) . '.php';
});
```

File Structure



```
chat-nojs-php-reload-mvc
  Source Files
    classes
      Chat
        Controller
          Controller.php
          SessionManager.php
        Integration
          ConversationStore.php
        Model
          Entry.php
        Util
          Util.php
          WebConsole.php
    resources
    .htaccess
    conversation.php
    index.php
    store-entry.php
```

- It is a good practice to organize server-side code as in a Java application. One file per class and one directory per namespace.
- Place all classes in a separate directory, for example **`classes`**.
- This enables autoloading classes, see below. We are relieved of **`include`** and **`require`** statements.

```
spl_autoload_register(function ($class) {
    include 'classes/' . \str_replace('\\', '/', $class) . '.php';
});
```

- We can also protect classes from direct HTTP access by denying access to the **`classes`** directory.

What About the Views?

- We would like to place the view in classes in the **`View`** directory. However:

What About the Views?

- We would like to place the view in classes in the **`View`** directory. However:
  - We do not want HTML in our PHP classes.

What About the Views?

- We would like to place the view in classes in the **`View`** directory. However:
  - We do not want HTML in our PHP classes.
  - We do not want HTTP access to our classes directory.

What About the Views?

- We would like to place the view in classes in the **`View`** directory. However:
  - We do not want HTML in our PHP classes.
  - We do not want HTTP access to our classes directory.
  - We can not write a URL that addresses a method in a PHP class. A URL can only address a file.

What About the Views?

- We would like to place the view in classes in the **View** directory. However:
  - We do not want HTML in our PHP classes.
  - We do not want HTTP access to our classes directory.
  - We can not write a URL that addresses a method in a PHP class. A URL can only address a file.
- Therefore, we need a PHP file without classes to interpret the HTTP request and direct it to the correct classes.

What About the Views?

- We would like to place the view in classes in the `View` directory. However:
  - We do not want HTML in our PHP classes.
  - We do not want HTTP access to our classes directory.
  - We can not write a URL that addresses a method in a PHP class. A URL can only address a file.
- Therefore, we need a PHP file without classes to interpret the HTTP request and direct it to the correct classes.
- If the response is a HTML document, we also need to include a HTML file, since we do not want to mix the HTML document with the PHP classes.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.
  - Avoid writing new code which means introducing new bugs.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.
  - Avoid writing new code which means introducing new bugs.
  - Thoroughly tested and proven to work well.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.
  - Avoid writing new code which means introducing new bugs.
  - Thoroughly tested and proven to work well.
  - Lots of documentation, easy to get help.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.
  - Avoid writing new code which means introducing new bugs.
  - Thoroughly tested and proven to work well.
  - Lots of documentation, easy to get help.
  - Infrastructure code is difficult to write.

Warning: Infrastructure Code!

- We now realize there will be quite a lot of code that is identical for each application.
- This is called infrastructure code and is a strong call for a framework.
- We need a framework:
  - Reuse code from previous applications.
  - Avoid the big risk of bad architecture.
  - Avoid writing new code which means introducing new bugs.
  - Thoroughly tested and proven to work well.
  - Lots of documentation, easy to get help.
  - Infrastructure code is difficult to write.
  - Preferably, the framework should use callbacks, i.e., the framework calls our code. Thus, the framework also handles flow control.

Exactly What is the Framework's Task?

- First, we will look at the chat application without a framework, to get a feeling for what is needed.

Exactly What is the Framework's Task?

- First, we will look at the chat application without a framework, to get a feeling for what is needed.
- Then, we will identify what we need the framework to do.

Exactly What is the Framework's Task?

- First, we will look at the chat application without a framework, to get a feeling for what is needed.
- Then, we will identify what we need the framework to do.
- Third, we will look at the chat with a framework.

Without Framework, **index.php**

```
require_once 'classes/Chat/Util/Util.php';
\Chat\Util\Util::initRequest();

$controller = \Chat\Controller\SessionManager::getController();
$conversation = $controller->getConversation();
\Chat\Controller\SessionManager::storeController($controller);

include 'conversation.php';
```

- **index.php** displays the chat web page with the current conversation.

Without Framework, **index.php**

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- `index.php` displays the chat web page with the current conversation.
- Line 1 loads the `Util` class. Since the autoloader is not yet registered, it is loaded manually.

Without Framework, **`index.php`**

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- **`index.php`** displays the chat web page with the current conversation.
- Line 1 loads the **`Util`** class. Since the autoloader is not yet registered, it is loaded manually.
- Line 2 calls the **`initRequest`** method, which performs tasks similar for all requests.

Without Framework, **`index.php`**

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- **`index.php`** displays the chat web page with the current conversation.
- Line 1 loads the **`Util`** class. Since the autoloader is not yet registered, it is loaded manually.
- Line 2 calls the **`initRequest`** method, which performs tasks similar for all requests.
- Line 4 gets the controller stored in the current session. Remember that all state is lost after a request. Therefore, we have to store the controller, with its references to the model, in the session.

index.php (Cont'd)

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- Line 5 is the method call to the controller to handle the HTTP request to show the conversation. It is retrieved and stored in **`$conversation`**.

`index.php` (Cont'd)

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- Line 5 is the method call to the controller to handle the HTTP request to show the conversation. It is retrieved and stored in **`$conversation`**.
- Line 6 again stores the controller in the session, for use in the next request.

`index.php` (Cont’d)

```
1 require_once 'classes/Chat/Util/Util.php';
2 \Chat\Util\Util::initRequest();
3
4 $controller = \Chat\Controller\SessionManager::getController();
5 $conversation = $controller->getConversation();
6 \Chat\Controller\SessionManager::storeController($controller);
7
8 include 'conversation.php';
```

- Line 5 is the method call to the controller to handle the HTTP request to show the conversation. It is retrieved and stored in **`$conversation`**.
- Line 6 again stores the controller in the session, for use in the next request.
- Line 8 shows the next view. Note that **`$conversation`** is available in that view.

Util.php

```
 1 public static function initRequest() {
 2     spl_autoload_register(function ($class) {
 3         include 'classes/' . \str_replace('\\', '/', $class)
 4                 . '.php';
 5     });
 6 
 7     if (\session_id() === '') {
 8         \session_start();
 9     }
10     self::defineHttpParams();
11 }
```

- Lines 2-5 registers the autoloader.

Util.php

```
 1 public static function initRequest() {
 2     spl_autoload_register(function ($class) {
 3         include 'classes/' . \str_replace('\\', '/', $class)
 4                 . '.php';
 5     });
 6
 7     if (\session_id() === '') {
 8         \session_start();
 9     }
10     self::defineHttpParams();
11 }
```

- Lines 2-5 registers the autoloader.
- Lines 7-9 starts a session if there is none.

Util.php

```
 1 public static function initRequest() {
 2     spl_autoload_register(function ($class) {
 3         include 'classes/' . \str_replace('\\', '/', $class)
 4                 . '.php';
 5     });
 6 
 7     if (\session_id() === '') {
 8         \session_start();
 9     }
10     self::defineHttpParams();
11 }
```

- Lines 2-5 registers the autoloader.
- Lines 7-9 starts a session if there is none.
- Line 10 creates constants for HTTP parameter keys:

```
const SYMBOL_PREFIX = "CHAT_";

private static function defineHttpParams() {
    self::defineHttpParam('AUTHOR_KEY', 'nickName');
    self::defineHttpParam('MSG_KEY', 'msg');
}
private static function defineHttpParam($param, $value) {
    define(self::SYMBOL_PREFIX . $param, $value);
}
```

SessionManager.php

```php
public static function getController() {
    if (isset($_SESSION[self::CONTROLLER_KEY])) {
        return unserialize($_SESSION[self::CONTROLLER_KEY]);
    } else {
        return new \Chat\Controller\Controller();
    }
}

public static function storeController(
            \Chat\Controller\Controller $controller) {
    $_SESSION[self::CONTROLLER_KEY] = serialize($controller);
}
```

- **`SessionManager`** has methods for storing and reading the controller to/from the **`$_SESSION`** superglobal.

The view, `conversation.php`

- The view should consist of only HTML. Unfortunately, this goal is not reached:

The view, **`conversation.php`**

- The view should consist of only HTML. Unfortunately, this goal is not reached:
- First, since there are header, footer and navigation fragments that appear on each page, we have to include them to avoid duplicated code. These inclusions are PHP statements, see lines 2 and 6 below.

```
1 ...
2     <header class="section group">
3         <?php include ’resources/fragments/header.html’; ?>
4     </header>
5 <main class="section group">
6     <nav class="section group">
7         <?php include ’resources/fragments/nav.html’; ?>
8     </nav>
9 ...
```

The view (Cont'd)

- Second, to generate the conversation view from the **`$conversation`** variable is also PHP code.

```
  ...
<div class="col span_4_of_4">
    <?php
    if (!empty($conversation)) {
        foreach ($conversation as $entry) {
            echo("<p class='author'>" . $entry->getNickName()
                  . ":</p>");
            foreach ($entry->getMsg() as $line) {
                echo("<p class='entry'>" . $line . "</p>");
            }
        }
    }
    ?>
</div>
  ...
```

Other Layers, No Problem

- Now we have seen all view code, which is normally the hardest part of a web application.

Other Layers, No Problem

- Now we have seen all view code, which is normally the hardest part of a web application.
- Controller and lower layers are plain PHP code, created with normal object-oriented analysis, design and programming methodologies.

Let's Look for Infrastructure Code

- In **`index`**, **`Util`** and **`SessionManager`** there is no code at all specific for this application!

Let’s Look for Infrastructure Code

- In **`index`**, **`Util`** and **`SessionManager`** there is no code at all specific for this application!
- One could argue that the call to the controller in **`index.php`** is application specific. However we are rid of also this line if the framework allows us to specify a URL-to-method mapping, which most frameworks do.

Let's Look for Infrastructure Code

- In `index`, `Util` and `SessionManager` there is no code at all specific for this application!
- One could argue that the call to the controller in `index.php` is application specific. However we are rid of also this line if the framework allows us to specify a URL-to-method mapping, which most frameworks do.
- One could also argue that the names of the HTTP parameters are application specific, but most frameworks enable specifying those as method parameters in the URL to method mapping.

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:
  - Class loading, i.e., include PHP class files.

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:
  - Class loading, i.e., include PHP class files.
  - Routing, which means to map a URL to a specified method in a specified class.

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:
  - Class loading, i.e., include PHP class files.
  - Routing, which means to map a URL to a specified method in a specified class.
  - HTTP parameters. It should be possible to specify how parameters are passed as arguments to the methods specified by the routing rules.

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:
  - Class loading, i.e., include PHP class files.
  - Routing, which means to map a URL to a specified method in a specified class.
  - HTTP parameters. It should be possible to specify how parameters are passed as arguments to the methods specified by the routing rules.
  - HTTP sessions

Infrastructure Code (Cont'd)

- Therefore, the framework must handle:
  - Class loading, i.e., include PHP class files.
  - Routing, which means to map a URL to a specified method in a specified class.
  - HTTP parameters. It should be possible to specify how parameters are passed as arguments to the methods specified by the routing rules.
  - HTTP sessions
  - Templating, which means to generate a view from data, we need something to replace the PHP code looping through the **`$conversation`** variable.

Infrastructure Code (Cont’d)

- The framework must handle:

Infrastructure Code (Cont’d)

- The framework must handle:
    - Composite views, there should be a mechanism to specify fragments (header, footer etc) for inclusion without having to mix HTML and PHP.

Infrastructure Code (Cont'd)

- The framework must handle:
  - Composite views, there should be a mechanism to specify fragments (header, footer etc) for inclusion without having to mix HTML and PHP.
  - Not only should it be possible to reuse the fragments, also the page layout should be reused. This means only the content of the main area should be specific for a page.

Infrastructure Code (Cont'd)

- There are also many other requirements that should be managed by the framework, but which we have skipped in this small example. Some examples are:

Infrastructure Code (Cont'd)

- There are also many other requirements that should be managed by the framework, but which we have skipped in this small example. Some examples are:
  - Navigation, to map a request to the next view without hardcoding.

Infrastructure Code (Cont'd)

- There are also many other requirements that should be managed by the framework, but which we have skipped in this small example. Some examples are:
  - Navigation, to map a request to the next view without hardcoding.
  - Validation of HTTP parameters.

Infrastructure Code (Cont'd)

- There are also many other requirements that should be managed by the framework, but which we have skipped in this small example. Some examples are:
  - Navigation, to map a request to the next view without hardcoding.
  - Validation of HTTP parameters.
  - Various security issues, like authentication (to log in).

Section

- Server-Side Tasks
- AJAX
- Server-Side Architecture
- The Flight Framework

PHP Frameworks

- There are **many** PHP frameworks, of different size and quality.

PHP Frameworks

- There are **many** PHP frameworks, of different size and quality.
- Some interesting and often used frameworks are Zend, Symfony, Yii, Laravel and Phalcon.

PHP Frameworks

- There are **many** PHP frameworks, of different size and quality.
- Some interesting and often used frameworks are Zend, Symfony, Yii, Laravel and Phalcon.
- Here, we will have a look at the Flight framework.

The Flight Framework

- The Flight framework, `http://flightphp.com/`, is very small and relatively easy to understand.

The Flight Framework

- The Flight framework, `http://flightphp.com/`, is very small and relatively easy to understand.
- Yet, it provides some very useful features, for example class loading, routing and composite views.

The Flight Framework

- The Flight framework, `http://flightphp.com/`, is very small and relatively easy to understand.
- Yet, it provides some very useful features, for example class loading, routing and composite views.
- Other features, on the other hand, are missing. For example there is no templating or session handling.

The Flight Framework

- The Flight framework, `http://flightphp.com/`, is very small and relatively easy to understand.
- Yet, it provides some very useful features, for example class loading, routing and composite views.
- Other features, on the other hand, are missing. For example there is no templating or session handling.
- Now, we will look at parts of the Flight framework and how it changes the Chat application.

Routing

- With Flight, all requests, independent of URL, shall be directed to **`index.php`**.

Routing

- With Flight, all requests, independent of URL, shall be directed to **`index.php`**.
- **`index.php`** contains a set of routes from URLs to functions.

```
\Flight::route('/', function() {
   // Action to be taken for the URL /
});
```

Routing

- With Flight, all requests, independent of URL, shall be directed to **`index.php`**.
- **`index.php`** contains a set of routes from URLs to functions.

```
\Flight::route('/', function() {
   // Action to be taken for the URL /
});
```

- These routes are checked for each request, the method indicated by the first found matching route is called.

HTTP Parameters

- Routes can include parameters, a path element prefixed with @ is a parameter.

HTTP Parameters

- Routes can include parameters, a path element prefixed with @ is a parameter.
- Here, the parameters **author** and **message** are passed to the request handling method.

```
\Flight::route('/new-msg/@author/@message',
               function($author, $message) {
     // Action to be taken for
     // the URL /new-msg
});
```

HTTP Parameters

- Routes can include parameters, a path element prefixed with @ is a parameter.
- Here, the parameters **author** and **message** are passed to the request handling method.

```
\Flight::route('/new-msg/@author/@message',
               function($author, $message) {
     // Action to be taken for
     // the URL /new-msg
});
```

- Note that the parameters shall be path elements of the URL, not HTTP request parameters.

Composite Views

- Flight provides a basic mechanism for composite views. It is possible to define a HTML layout page, into which the page fragments are inserted.

```
<body>
    <header class="section group">
        <?php echo $header_content; ?>
    </header>
    <main class="section group">
        <nav class="section group">
            <?php echo $nav_content; ?>
        </nav>
        <?php echo $body_content; ?>
        <footer class="section group">
            <?php echo $footer_content; ?>
        </footer>
    </main>
</body>
```

Composite Views

- The fragments must be defined.

```
\Flight::render($path_to_header_fragment, NULL,
                'header_content');
// More fragment definitions
```

Composite Views

- The fragments must be defined.

```
\Flight::render($path_to_header_fragment, NULL,
                'header_content');
// More fragment definitions
```

- An object can be passed to a fragment.

```
\Flight::render($view, array($model_name => $model),
                'body_content');
```

Composite Views

- The fragments must be defined.

```
\Flight::render($path_to_header_fragment, NULL,
                'header_content');
// More fragment definitions
```

- An object can be passed to a fragment.

```
\Flight::render($view, array($model_name => $model),
                'body_content');
```

- Finally, the HTML page is rendered.

```
\Flight::render($this->layout, NULL);
```

The Abstract Executor

- Some important parts are missing in Flight.

The Abstract Executor

- Some important parts are missing in Flight.
  - We still need PHP code to create the composite view.

The Abstract Executor

- Some important parts are missing in Flight.
  - We still need PHP code to create the composite view.
  - Store and retrieve controller in session superglobal.

The Abstract Executor

- Some important parts are missing in Flight.
  - We still need PHP code to create the composite view.
  - Store and retrieve controller in session superglobal.
  - Pass HTTP parameters to setters and getters.

The Abstract Executor

- Some important parts are missing in Flight.
  - We still need PHP code to create the composite view.
  - Store and retrieve controller in session superglobal.
  - Pass HTTP parameters to setters and getters.
  - This is handled in the class **`AbstractExecutor`**. Normally, there is such a class in the framework.

Mission Completed!

- Now consider the chat application with both knockout and flight.

Mission Completed!

- Now consider the chat application with both knockout and flight.
- Apart from **`AbstractExecutor`** and a few lines in **`index.php`** there is no infrastructure code!!

Mission Completed!

- Now consider the chat application with both knockout and flight.
- Apart from **`AbstractExecutor`** and a few lines in **`index.php`** there is no infrastructure code!!
- Adding more functionality involves only new implementations of **`AbstractExecutor`**, new routes in **`index.php`** and ordinary object-oriented code in controller and lower layers.

Mission Completed!

- Now consider the chat application with both knockout and flight.
- Apart from **`AbstractExecutor`** and a few lines in **`index.php`** there is no infrastructure code!!
- Adding more functionality involves only new implementations of **`AbstractExecutor`**, new routes in **`index.php`** and ordinary object-oriented code in controller and lower layers.
- All this is application specific!