Faculty of Science,
Technology
and Medicine

Web Programming

Volker Müller
University of Luxembourg

UNIVERSITÉ DU LUXEMBOURG

OO Programming with PHP

PHP supports "standard features" of OOP:

- Classes and inheritance
- Static class members
- Variable Visibility: public, protected, private
- Abstract class, exceptions, ….
- Namespaces (stored in directories, access uses "\")

Traits: Code Reuse in Single Inheritance Languages

Single inheritance = class can only have one direct super-class

Traits enable developer to reuse sets of methods freely in several independent classes living in different class hierarchies

Traits can not be instantiated (similar to abstract classes)

Namespaces

Namespaces were added to PHP to allow grouping files in different directories (like packages in Java)

Namespace names are case-insensitive !!

Namespace must be declared at top of code

Ex: namespace MyProject\Sub\Level;

→ File stored in directory MyProject/Sub/Level

PECL

PECL provides extensions to php

PECL extensions have to be compiled and stored as dynamic libraries, then loaded by php executable → compiler infrastructure must be available on machine

https://pecl.php.net/ contains information about available extensions

See Docker definition for Nginx-PHP-MongoDB as usage example

PHP-related Tools – PHP Archives

PHP Archives (file extension .phar): put entire PHP applications into a single archive file for easy distribution and installation

Created either by IDE or with command phar included in PHP installation

PHP class "Phar" also exists for dynamic generation within PHP code

PHP-related Tools – Phing

Phing: PHP project build system based on Apache ant, manages testing, package generation, documentation generation

Rich set of provided standard tasks, applied in XML configuration files

Available at https://www.phing.info/

PHP-related Tools - Composer

Composer: Dependency Manager for PHP, allows you to declare the libraries (with PHP code) your project depends on and it will manage (install/update) them for you: https://getcomposer.org/

Define dependencies in composer.json:

```
{   "require": {   "monolog/monolog": "1.0.*"   }   }
```

→ Run `php composer.phar install`

Default repo: packagist.org

PHP-related Tools - PHPUnit

PHPUnit = programmer oriented unit testing framework for PHP (https://phpunit.de)

Can be nicely integrated with composer

Uses annotations for meta-information

Idea similar to JUnit used for Java unit tests

Example TestCase

```
use PHPUnit\Framework\TestCase;

class DataTest extends TestCase
{
    /**   @dataProvider additionProvider  */
    public function testAdd($a, $b, $expected)
    {  $this->assertSame($expected, $a + $b);   }

    public function additionProvider()
    {   return [  [0, 0, 0],   [1, 1, 3] ];    }
```

PHP Annotations

Annotations = meta-data embedded in source code

PHP-Annotations are no official part of PHP, but used quite often

Similar form as in Java: @var

<u>But:</u> Annotations must be written inside comments

Add to project: composer require annotations

More details: php-annotations.readthedocs.io

Short Introduction to SOAP Web Services with PHP

- Allows direct access of web data
- Clients - Servers directly communicate over SOAP XML messages
- Language-independent
- Can be described in a WSDL file (also in XML)
- Alternative: RESTful Web Services (covered later)

SOAP Server (without WSDL)

```
class ADDER {
  function add($x, $y) {
    $x = intval($x);  $y = intval($y);
    if ($x <= 100 && $y <= 100) {  return ($x + $y); }
    else {  throw new SoapFault ("Server","Integers > 100"); }
}}
$server = new SoapServer (null, array('uri' => "http://soap1"));
$server->setClass("ADDER");
$server->handle( );
```

SOAP Client (without WSDL)

```
$client = new SoapClient (null, array(
 'location' => "http://127.0.0.1/SOAPserver/soapserver1.php",
  'uri' => "http://soap1"));
try {
$v = $client->add($x, $y);
 print "<p>$x + $y is equal to " . $v . "</p>";
}
catch (SoapFault $exception) {  echo $exception;  }
```

Assume $x, $y have some integer values

Web Service Description Language (WSDL)

XML based language that describes a model of web services

Also used to locate web services

W3C recommendation: http://w3.org

Four major elements: portType, message, types, binding

SOAP with WSDL

```
$server = new SoapServer
   ("http://127.0.0.1/SOAPserver/adder.wsdl");
$server->setClass("ADDER");
$server->handle( );
```

Server

Client

```
 $client = new SoapClient
   ("http://127.0.0.1/SOAPserver/adder.wsdl");
  try {
  $v = $client->add($x, $y); ....
```

WSDL Caching

WSDL caching is per default enabled to be more efficient

You should disable WSDL caching, otherwise you might get "strange results" during debugging

```
ini_set ("soap.wsdl_cache_enabled", "0");
```

NoSQL Databases

NoSQL DB provides mechanism for storage and retrieval of data not modeled as tabular relations

Increasingly used in big data and real-time web applications

Some use SQL-like query language, but not always

Some examples: Cassandra, SAP HANA, Apache CouchDB, MongoDB, Redis ...

Comparison NoSQL – Relational DB

SQL-DB provide more rigid, very structured way of storing data with well-defined schema

Data requirements not clear, massive amounts of unstructured data → NoSQL offer much greater flexibility and ease of access

Storing data in bulk like this requires extra processing effort and more storage than highly organized SQL data (but we have the Cloud)

Reasons to use NoSQL database

Storing large volumes of data that often have little to no structure

Making the most of cloud computing and cloud storage to get massive high availability data stores (TB of data)

Data in normal relational DB would require many joins

Rapid development (“agile development”)

Example: Redis (https://redis.io)

Open source, in-memory data structure store

Supports some build-in data structures: strings, hashes, lists, sets, sorted sets with range queries, bitmaps, ...

Built-in replication, Lua scripting, transactions, different levels of on-disk replication

Provides high availability and automatic partitioning with Redis Cluster

Using Redis with Docker

Run Redis container:

```
docker run -d --name RE -v ~/d:/data redis redis-server
```

Run Redis client interface (like psql):

```
docker exec -it RE redis-cli
```

Help command:

```
help @string / @list / @set …
```

Redis Tutorial: http://www.tutorialspoint.com/redis

Example 2: MongoDB

Free and open-source cross-platform document-oriented database program

MongoDB uses JSON-like documents with schemas

MongoDB supports field, range queries, regular expression searches, fields can be indexed

Load balancing and replication supported

MongoDB with Docker

Run MongoDB container:

```
docker run -d --name MO -v ~/d:/data/db mongo:latest
```

Connect to client interface “MongoDB Shell”:

```
docker exec -it MO mongo
```

Use the build-in help for some commands

MongoDB Tutorial: http://www.tutorialspoint.com/mongodb

Example: Nginx – PHP - MongoDB

On Moodle, I prepared a small docker application composed of Nginx, PHP 7.4, MongoDB

Example illustrates how "queries" can be sent to MongoDB

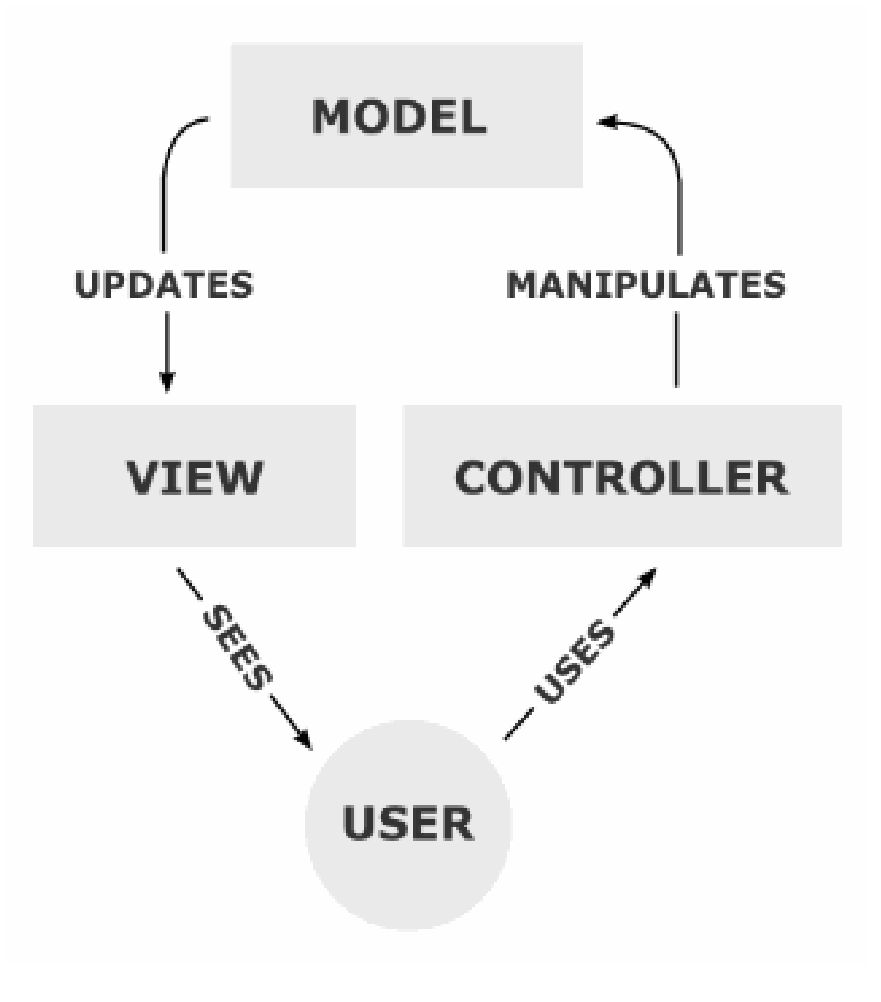
Uses composer with package mongodb/mongodb

Info on available methods:
https://docs.mongodb.com/php-library/current/tutorial

Note that every NoSQL DB has its own syntax

PHP Frameworks

Excursus: MVC



Controller: mediates input, converting it to commands for the model or view

Model: application data and business rules

View: output representation of data

Where should a Framework help?

Speed up web app development

Provide common functionalities – input validation, templating, attack protection

Work efficiently with databases

Simplify maintenance

Support efficient testing of application

CakePHP (cakephp.org)

PHP framework based on MVC:

- Based on "Convention over Configuration" principle – basic organizational structure required
- Translations, database access, caching, validation, authentication, ... built in
- Built-in tools for input validation, CSRF protection, form tampering protection, SQL injection prevention, and XSS prevention

Laravel (laravel.com)

Open-source PHP 7.x framework using the MVC paradigm

Provides modular packaging system with dedicated dependency manager, different ways for accessing relational databases, utilities that aid in application deployment and maintenance, …

Increasingly popular in last years

Symfony Framework (symfony.com)

PHP framework for web applications, aims at speeding up creation and maintenance of web sites

OO design: provides many decoupled, reusable components (also usable in other products)

Tries to realize design principle separation of concerns: separate computer program into distinct features that overlap in functionality as little as possible

Implements MVC

Tutorials on Symfony

On symfonycasts.com, many video tutorials on different aspects of Symfony are available, symfony.com contains a detailed tutorial

These might be quite helpful if you have to dig deeper into the system

I will provide now a few basic examples, see the official website for more details

Project Setup

Symfony is completely based on composer

Initial project setup: composer create-project symfony/website-skeleton MYPROJECT

Other components can also be installed at any time with composer (symfony.com/doc/current/components)

For a page, you have to define a controller and a route to the page

Example Controller

```
<?php   // src/Controller/LuckyNumberController.php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Routing\Annotation\Route;

class LuckyNumberController extends AbstractController   {

    /**

     * @Route("/lucky/number", name="lucky_number")

     */

    public function index()

    {  …. return $this->render('lucky_number/index.html.twig',

            [ 'number' => 1 ]);  }  }
```

Creating Controllers

Controllers can be generated quickly with auto-generation:

```
php bin/console make:controller
```

→ Skeleton of a controller will be created

Note that every controller inherits from `AbstractController`

`bin/console` provides options for various auto-generations and other admin tasks

Routes

Routes can be defined with an annotation (@Route) or in an external configuration file config/routes.yaml

YAML format is exactly the same format as used in Docker configuration files

YAML is a human friendly data serialization standard for all prog. languages → yaml.org

Dynamic Routes

Routes are not necessarily static, but they can contain dynamic parts of "defined format" whose value can be accessed in annotated method:

```
/**  @Route("/blog/{page}", name="blog_list",
 *                    requirements = {"page" = "\d+"})
*/
 public function list($page) { ….}
```

Twig (twig.symfony.com)

Modern templating engine for PHP: "fast, secure, flexible"

Very concise syntax:

{{var}} will output value of variable "var"

{{var | escape}} will output "var" with escaping

Twig is realizing the view in Symfony

In controller: $this->render(filename_to_twig_file, array_of_variables_with_values);

Why Templating?

Templates ensure that all pages will have the same overall structure

Templates can also be used recursively to define parts of a page with a common layout

Templates ensure that same CSS and JS code is included in every page

One web application can make use of many different templates

Twig or PHP for the View?

Symfony can also use views defined with plain PHP

Advantages of Twig:

- Focus explicitly on design
- Twig can do nice things: whitespace control, sandboxing, automatic HTML escaping, inclusion of custom functions
- Inheritance build-in
- Twig templates are compiled and cached → speed

Twig Syntax Examples

{{ ... }} → output something

{% ... %} → do something (run code in body)

{# ... #} → Twig comment

Filters: {{ title | upper }}

Long list of available filters: abs, batch, date, escape, format, json_encode, lower, upper, path, reverse, url_encode …. (see documentation for full list)

A Note on Links in Templates

In Twig templates, links to other pages should never be hard-coded as string

In the definition of routes, we have seen a route name

Use the route name in every template:

```
<a href="{{path("routename")}}"> … </a>
```

This allows changing the route address without affecting other code (as long as name unchanged)

Twig Syntax Examples (cont.)

```
{% for i in 1..10 %}
  <div class="{{cycle([‘even’, ‘odd’], i)}}">
      ....
  </div>
{% endfor %}
```

Twig Syntax Examples (cont.)

```
<ul>  {% for user in users if user.active %}
              <li>{{ user.username }}</li>
        {% else %}
              <li>Inactive user</li>
       {% endfor %}
</ul>
```

Twig Templating – Base Template

```
<title>{% block title %}Test{% endblock %}</title> ….
  <div id="sidebar">
        {% block sidebar %}
           <ul> <li><a href="/">Home</a></li>
                 <li><a href="/blog">Blog</a></li>
           </ul>
        {% endblock %}
  </div> ...
```

Blocks can have default values

Twig Templating – Using Base Template

```
{% extends 'base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}
```

If no value provided for block, then the default value from base template used

Template are usually stored in the /templates directory or sub-directories

UNIVERSITÉ DU LUXEMBOURG

Twig Documentation

Twig offers many more features

For a complete overview with many detailed examples, see the official website twig.symfony.com/doc/2.x/

Doctrine

Doctrine is another very useful (third party) library tightly integrated into Symfony

Doctrine provides ORM ("Object-Relational Mapping")

Tables in DB $\leftrightarrow$ Classes in PHP

Users work with PHP classes, Doctrine "automatically" synchronizes these with used DB to make changes persistent $\rightarrow$ these classes represent the model

Auto-Created Entities

Enter connection details in .env (in debug mode)

php bin/console doctrine:database:create

php bin/console make:entity will create the PHP class

php bin/console make:migration and php bin/console make:migrations:migrate will create the resp. DB table

Using Doctrine – Writing Data

Persist writes changes to a cache, but not yet to the DB → this is done by flush

```
use App\Entity\Student;
  ….
  $entityManager = $this->getDoctrine()->getManager();
  $student = new Student();
  $student->setName('VM'); ….
  $entityManager->persist($student);
  $entityManager->flush();
```

Using Doctrine – Reading Data

```
use App\Entity\Student;

  ….

  $this->getDoctrine()->getRepository(Student::class)

      -> find(1);
```

Repository for each entity class also automatically generated, provides list of functions provided to find entities satisfying some conditions

Other Features in Doctrine

Insert, update, delete are naturally possible

Search can also provide an array with conditions:

```
...-> findBy(array('age' => array(20,30,40)))
```

Relationships can be represented

Explicit query in a language similar to SQL also possible

See the tutorial for more details

Remark: ORM in JavaEE (JPA) very similar

Final Remarks

Only a small glimpse of possibilities shown (debugging and testing support still missing)

Learning Symfony requires some time, but then development can be done quite fast

I encourage you to go through the tutorial and to invest this time → 3rd (optional) exercise with Symfony gives 10% extra points if done

Next Meeting

The Basics of JavaScript