CTEC3110
Secure Web Application Development

Lecture 3

Logging, Dependency Injection Containers & Namespaces

Lecture Content

- SLIM application architecture
- Logging output with Monolog
- Dependency Injection
  - Dependency Injection Container
- Namespaces

Post-Lecture Work

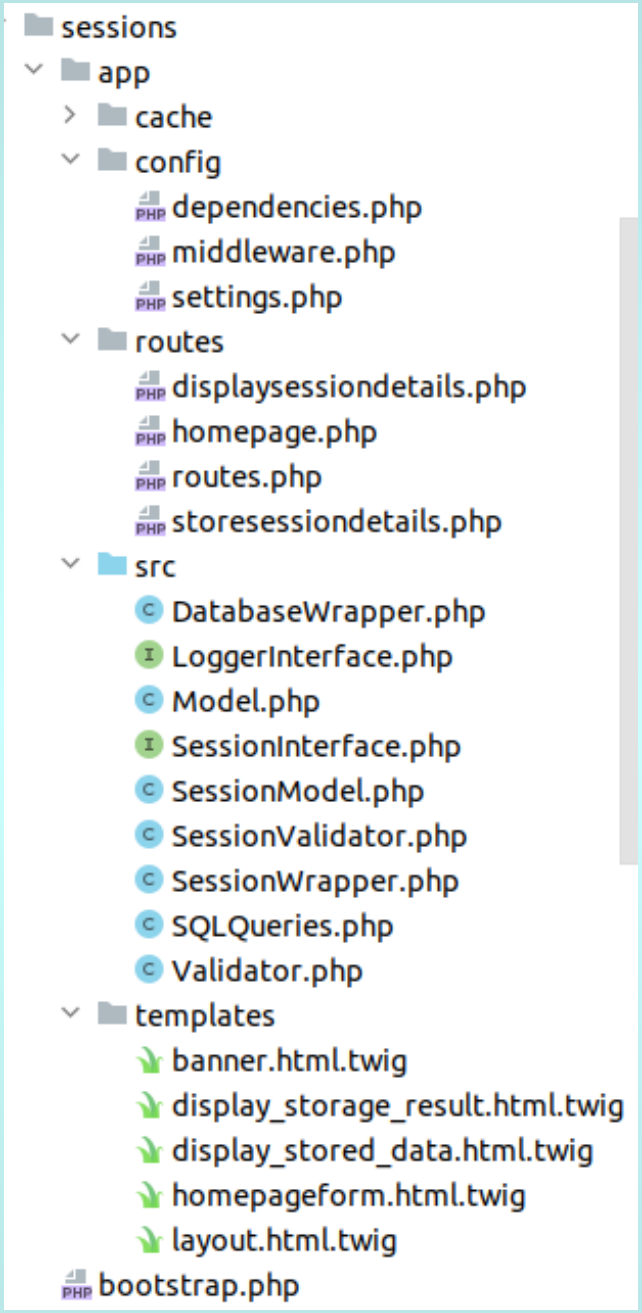
- Investigate any ethical and/or legal consequences of Logging
  - taking GDPR into consideration
- Watch the suggested video for a simple explanation of Dependency Injection
  - explain the Dependency Inversion Principle to a friend in your own words

References & Reading

- https://github.com/Seldaek/monolog
- https://stackify.com/php-monolog-tutorial/
- https://www.php-fig.org/psr/psr-3/
- https://tools.ietf.org/html/rfc5424
- https://www.youtube.com/watch?v=IKD2-MAkXyQ
- https://www.sitepoint.com/dependency-injection-with-pimple/
- https://www.sitepoint.com/php-53-namespaces-basics/

Lecture Content

- **SLIM application architecture**
- Logging output with Monolog
- Dependency Injection
  - Dependency Injection Container
- Namespaces



SLIM Application Architecture

- public_php
  - index.php
- includes
  - application/app
    - bootstrap.php
    - cache
    - config
    - routes
    - src
    - templates

Lecture Content

- SLIM application architecture
- **Logging output with Monolog**
- Dependency Injection
  - Dependency Injection Container
- Namespaces

Logging

- Logging is necessary to make data available for later analysis
  - information gathering
  - troubleshooting
  - generating statistics
  - auditing
  - profiling
  - security
  - identifying & resolving breaches
  - market analysis
  - user experience

Logging

- All (significant) changes of state in a web application should be logged
  - input validation failures
  - authentication and authorization failures
  - application errors
  - configuration changes
  - application start-ups and shut-downs
  - user registration and authentication
  - searching
  - page views

Logging

- Don’t log these events:
  - application source code
  - session identification values
  - access tokens
  - sensitive personal data
  - passwords
  - database connection strings
  - encryption keys
  - bank account and card holder data

Logging

- should be meaningful
- should contain context
- should be balanced
  - should not include too little or too much information
- messages should be understandable to humans and parseable by machines
- should be organised into different levels
- should be adapted to development and to production
- in more complex applications should be organised into multiple log files

Monolog

- PSR-3 compliant
  - allow libraries to receive a *Psr\Log\LoggerInterface* object and write logs to it in a simple and universal way
  - https://www.php-fig.org/psr/psr-3/
- Write logs as
  - files, sockets, inboxes, databases and various web services

Monolog

- *LoggerInterface*
  - exposes eight methods to write logs to the eight RFC 5424 levels
  - debug, info, notice, warning, error, critical, alert, emergency
    - https://tools.ietf.org/html/rfc5424

Monolog

```php
<?php

use Monolog\Logger;
use Monolog\Handler\StreamHandler;

require 'vendor/autoload.php';

$logs_file_path = '/p3t/phpappfolder/logs/';
$logs_file_name = 'tester.log';
$logs_file = $logs_file_path . $logs_file_name;
```

Monolog

```php
// create a log channel
$log = new Logger( name: 'logger');
$log->pushHandler(new StreamHandler($logs_file,  level: Logger::WARNING));

// add records to the log
echo 'Adding entries to the log file';
$log->warning( message: 'Testing the Monolog library');
$log->error( message: 'Bar');
```

- pushHandler()
  - Creates a handler of the required type and parameters
- StreamHandler()
  - creates a PHP output stream (to a file)

Monolog

```php
// create a log channel
$log->pushHandler(new StreamHandler($logs_file_warning, level: Logger::WARNING));
$log->pushHandler(new StreamHandler($logs_file_notice, level: Logger::NOTICE));

// add records to the log
echo 'Adding entries to the log file';
$log->notice( message: 'Testing the Monolog library');
$log->warning( message: 'Testing warnings');
```

- Multiple Handlers
  - the order of handlers changes what output ends up where

Lecture Content

- SLIM application architecture
- Logging output with Monolog
- **Dependency Injection**
  - Dependency Injection Container
- Namespaces

Dependency

- A dependency is any other object type which a class has a direct relationship with. When a class depends directly upon another object type, it can be described as being coupled to that type.
  - https://stackoverflow.com/questions/46709170/difference-between-dependency-injection-and-dependency-inversion

Dependency Injection

- Design pattern
- Injection of dependencies
- One class should not be *dependent* upon another

Dependency Injection

- What is a dependency?
  - one class is dependent upon another
  - *Student* is dependent upon *Database*

```php
<?php

class Student{

    function __construct(){
        $db_handle = new Database();
    }
}
```

Dependency Injection

- Injecting a dependency?
  - *Database* object is passed into *StudentDI*

```php
class StudentDI{
    private $db_handle;

    function __construct($db_handle){
        $this->db_handle = $db_handle;
    }
}

$db_handle = new Database();
$student_di = new StudentDI($db_handle);
```

Dependency Injection

- Injecting a dependency?
  - database object is passed into Student_DI
  - using a setterMethod()

```
class StudentDI{
    private $db_handle;

    function __construct($db_handle){
        $this->db_handle = $db_handle;
    }

    public function setDbHandle($db_handle){
        $this->db_handle = $db_handle;
    }
}

$db_handle = new Database();
$student_di = new StudentDI($db_handle);
$student_di->setDbHandle($db_handle);
```

Dependency Injection

- The dependencies have been decoupled

  “decouples a classes’s construction from the construction of its dependencies”

Dependency Inversion Principle

- Code should depend upon abstractions
  - in PHP, this means Interfaces
    - define mandatory methods, but no state (content)
- Different dependencies can be substituted
  - subject to consistent interface
  - this decouples code from lower level implementations

Dependency Injection Container

- But, now developers need to
  - know all dependencies
  - instantiate them

Dependency Injection Container

- A DIC (aka Service Locator) solves this
  - map of required dependencies
  - code to instantiate dependencies
  - complex dependencies are resolved transparently
  - replacing a dependency only means updating the container
  - code is now more modular
    - and therefore, more reusable

Pimple

- SLIM uses the Pimple library as a DIC by default
  - others are available
- See https://www.sitepoint.com/dependency-injection-with-pimple/ for more examples and discussions

Dependency Injection Container

- in bootstrap.php

```php
use DI\Container;
use Slim\Factory\AppFactory;
```

```php
// Create Container using PHP-DI
    $container = new Container();

// Inject the container into the AppFactory, then instantiate the app
    AppFactory::setContainer($container);
```

Dependency Injection Container

- in bootstrap.php

```php
// make the settings available
$settings = require $config_dir . 'settings.php';
$settings($container, $app_dir);
```

```php
// make the dependencies available
$dependencies = require $config_dir . 'dependencies.php';
$dependencies($container, $app);
```

Dependency Injection Container

- dependencies.php
  - create a callback for all dependencies

```php
use DI\Container;


// Register components in a container
return function (Container $container, App $app)
{
```

Dependency Injection Container

- dependencies.php
  - add a callback for each dependency

```php
use Sessions\SessionValidator;
```

```php
    $container->set(
        'validator',
        function ()
        {
            $validator = new SessionValidator();
            return $validator;
        }
    );
```

Lecture Content

- SLIM application architecture
- Logging output with Monolog
- Dependency Injection
  - Dependency Injection Container
- **Namespaces**

Namespaces

- Used to avoid collisions of class/method names
- Keyword *namespace*
  - first command in script
    - except *declare()*
- All code that follows belongs to the namespace

Namespaces

```
<?php
namespace MyProject;
```

- Also possible – sub-namespaces
- Eg

  MyProject\SubName

  MyProject\Database\MySQL

  CompanyName\MyProject\Common\Widget

Namespaces in SLIM

- SLIM can define namespaces in the *composer.json* script as part of the autoloader:

```
"autoload": {
    "psr-4": {
        "Sessions\\": "sessions/app/src",
```

- Accessed in the dependencies script:

```
$container->set(
    'validator',
    function ()
    {
        $validator = new Sessions\SessionValidator();
        return $validator;
    }
);
```

Lecture Summary

- SLIM application architecture
- Logging output with Monolog
- Dependency Injection
  - Dependency Injection Container
- Namespaces