ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

PHP – Good Pratices

Sistemas de Informação 1

Credits and Disclaimer

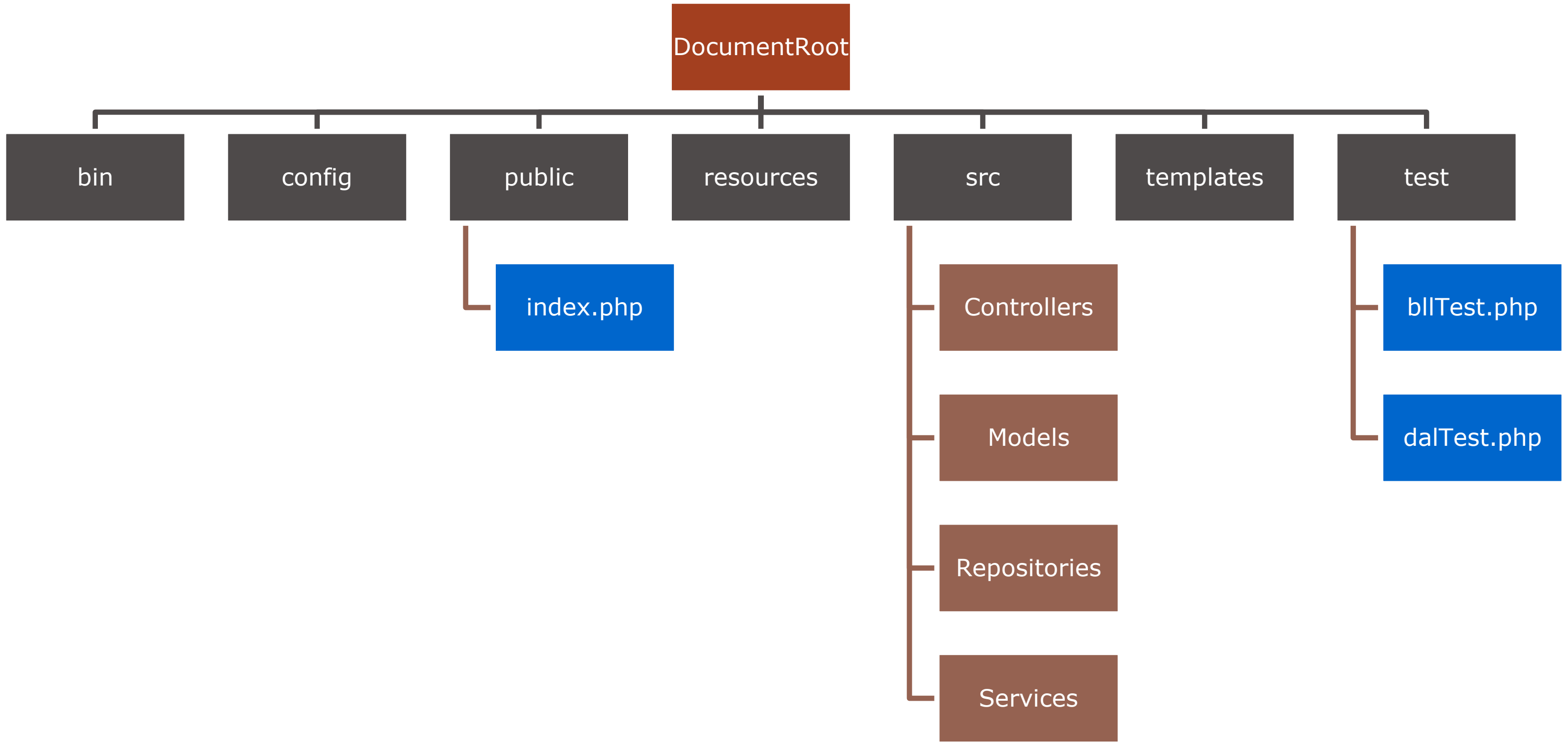
- This material/slides are adapted from:
  - SINF1 2022/23 slides produced by Constantino Martins
  - ARQSI 2014/15 slides
  - Books
  - Web sites:
    - https://www.php.net/;
    - https://www.w3schools.com/
    - ...

PHP – The Right Way

- Use the Current Stable Version (8.4)
  - PHP 8.x adds many new features over the older 7.x and 5.x versions.
  - The engine has been largely re-written, and PHP is now even quicker than older versions.
  - PHP 8 is a major update of the language and contains many new features and optimizations.

PHP – The Right Way

- Common Directory structure
  - Based on research into common practices across tens of thousands of PHP projects on GitHub, a PHP project should have the following structure:



PHP – The Right Way

- Code Sytle
  - You should write PHP code that adheres to a known standard like:
    - PSR-1: Basic Coding Standard
    - PSR-12: Extended Coding Style
    - PER Coding Style 2.0
    - PEAR Coding Standards
    - Symfony Coding Standards
  - English is preferred for all symbol names and code infrastructure. Comments may be written in any language easily readable by all current and future parties who may be working on the codebase.
  - This means other developers can easily read and work with your code, and applications that implement the components can have consistency even when working with lots of third-party code.

PHP – The Right Way

- Design Patterns
  - When you are building your application, it is helpful to use common patterns in your code and common patterns for the overall structure of your project.
    - Using common patterns is helpful because it makes it much easier to manage your code and lets other developers quickly understand how everything fits together.
  - If you use a framework then most of the higher-level code and project structure will be based on that framework, so a lot of the pattern decisions are made for you. But it is still up to you to pick out the best patterns to follow in the code you build on top of the framework. If, on the other hand, you are not using a framework to build your application then you must find the patterns that best suit the type and size of application that you’re building.

PHP – The Right Way

- Security
  - The basics of web application security are based in the topics:

    1. Code-data separation.
       - When data is executed as code, you get SQL Injection, Cross-Site Scripting, Local/Remote File Inclusion, etc.
       - When code is printed as data, you get information leaks (source code disclosure or, in the case of C programs, enough information to bypass ASLR).

    2. Application logic.
       - Missing authentication or authorization controls.
       - Input validation.

PHP – The Right Way

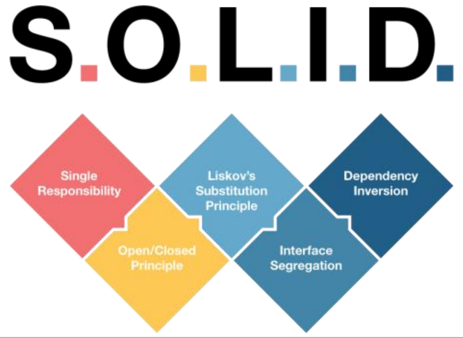
- Security
  - The basics of web application security are based in the topics:

3. Operating environment.
   - PHP versions.
   - Third party libraries.
   - The operating system.

4. Cryptography weaknesses.
   - Weak random numbers.
   - Chosen-ciphertext attacks.
   - Side-channel information leaks.

PHP – The Right Way

- STUPID vs SOLID Programming
  - SOLID is an acronym that stands for the five fundamental concepts in software design.
    - Single Responsibility Principle
    - Open/Closed Principle
    - Liskov Substitution Principle
    - Interface Segregation Principle
    - Dependency Inversion Principle



PHP – The Right Way

- SOLID Programming
  - Single Responsibility Principle
    - It states that “*A class should have only one reason to change.*”
    - This means that every class should only have responsibility over a single part of the functionality provided by the software.
    - The largest benefit of this approach is that it enables improved code reusability.
  - Open/Closed Principle
    - It states that “*Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.*”
    - This means that we should design our modules, classes and functions in a way that when a new functionality is needed, we should not modify our existing code but rather write new code that will be used by existing code.
      - Practically speaking, this means that we should write classes that implement and adhere to interfaces, then type-hint against those interfaces instead of specific classes.

PHP – The Right Way

- SOLID Programming
  - Liskov Substitution Principle
    - It states that "*Child classes should never break the parent class's type definitions.*"
    - This means that subtypes must be substitutable for their base types.
  - Interface Segregation Principle
    - It states that "*No client should be forced to depend on methods it does not use.*"
    - This means that instead of having a single monolithic interface that all conforming classes need to implement, we should instead provide a set of smaller, concept-specific interfaces that a conforming class implements one or more of.
  - Dependency Inversion Principle
    - It states that one should "*Depend on Abstractions. Do not depend on concretions.*".
    - Put simply, this means our dependencies should be interfaces/contracts or abstract classes rather than concrete implementations.

PHP – The Right Way

- STUPID vs SOLID Programming
  - STUPID is also an acronym that stands for 6 of the most common ways programmers write bad code.
    - Singleton
    - Tight Coupling
    - Untestability
    - Premature Optimization
    - Indescriptive Naming
    - Duplication

PHP – The Right Way

- STUPID Programming
  - Singleton
    - The singleton pattern is a software design pattern that **restricts the instantiation of a class to a singular instance**.
    - Singletons are controversial, and they are often considered anti-patterns. You should avoid them.
    - The use of a singleton is not the problem, but the symptom of a problem. Here are two reasons why:
      - Programs using global state are very difficult to test;
      - Programs that rely on global state hide their dependencies.

PHP – The Right Way

- STUPID Programming
  - Tight Coupling
    - Tight coupling, also known as strong coupling, is a generalization of the Singleton issue.
    - Basically, you should reduce coupling between your modules.
      - Coupling is the degree to which each program module relies on each one of the other modules.
    - If making a change in one module in your application requires you to change another module, then coupling exists.
      - For instance, you instantiate objects in your constructor's class instead of passing instances as arguments. That is bad because it doesn't allow further changes such as replacing the instance by an instance of a sub-class, a mock or whatever.

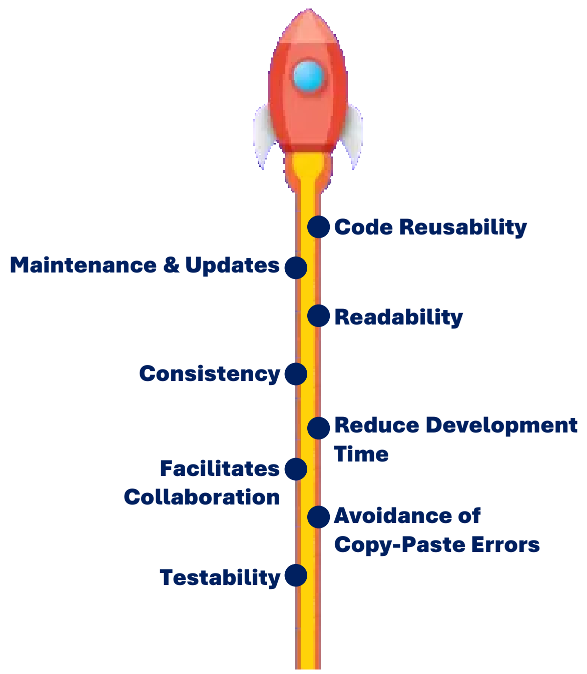
PHP – The Right Way

- STUPID Programming
  - Untestability
    - In most cases, the impossibility of testing is caused by a tight coupling.
  - Premature Optimization
    - This refers to spending excessive effort optimizing code before it's necessary, often before the code even has a fully functional implementation.
    - Premature Optimization is often described as "*the root of all evil*".

INVESTED QUITE SOME TIME TO OPTIMISE MY CODE PREMATURELY
APPLICATION IS NOW 0.001 SECONDS FASTER THAN BEFORE

PHP – The Right Way

- STUPID Programming
  - Indescriptive Naming
    - Name your classes, methods, attributes, and variables <u>properly</u>... and don’t abbreviate!
    - You write code for people, not for computers.
      - Computers just understand 0 and 1;
      - Programming languages are for humans.
  - Duplication
    - Please *Don’t Repeat Yourself (DRY)* and *Keep It Simple, Stupid (KISS)*.
    - Be lazy the right way - write code only once!

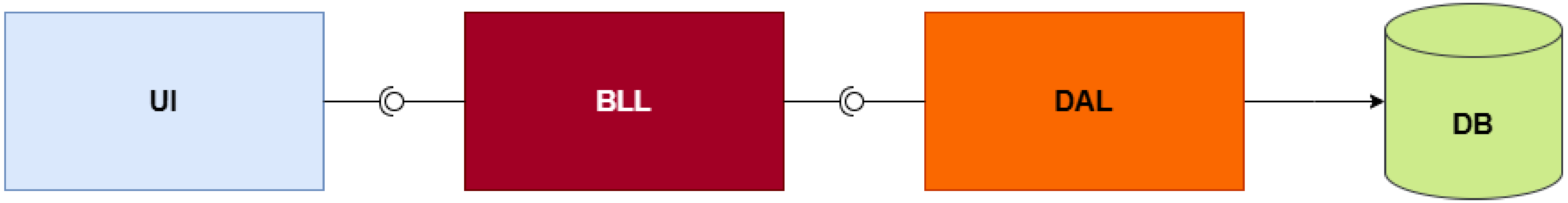


PHP – The Right Way

- Because we want to code SMART, we will consider three software layers
  - User Interface - the part of an application that allows users to interact with it.
    - It is responsible for presenting information to the user and handling their input, making the application accessible and user-friendly.
  - Business Logic Layer - the part of an application that handles the core business rules and processes, acting as an intermediary between the presentation (user interface) and data access layers.
    - It encapsulates the logic needed to manipulate data, execute workflows, and manage interactions between the three layers, ensuring the application adheres to business rules.
  - Data Access Layer - a software layer that simplifies and manages access to data, typically stored in a database or other persistent storage.
    - It acts as an intermediary between the application's business logic and the data storage system, allowing the business logic to interact with the data without needing to know the specific details of the underlying data source.

PHP – The Right Way

- Desired Software Architecture



PHP – The Right Way

- Desired Software Architecture

index.php

```
<?php include "BusinessLogicLayer.php"; ?>
<html>
   <head><title>Desired architecture example</title></head>
   <body>
       <div style="text-align:right;width:100%;">
            <?php getTodaysDate() ?>
       </div>
       <?php getAllStudents(); ?>
   </body>
</html>
```

PHP – The Right Way

- Desired Software Architecture

BusinessLogicLayer.php

```
<?php
   include "DataAccessLayer.php";

   function getTodaysDate() {
       echo "Today is " . Date("d/m/Y") . "<br/>";
   }

   function getAllStudents() {
       $dal = new DAL();
       $students = $dal->getStudents();
       foreach( $students as $row ){
          echo $row['name'] . "-" . $row['age'];
       }
   }

?>
```

PHP – The Right Way

- Desired Software Architecture

DataAccessLayer.php

```php
<?php
    class DAL{
       private $conn;
       function __construct(){
          $this->conn = new mysqli('localhost', 'username', 'password', 'StudentsDB');
       }
       function getStudents(){
          if($this->conn){
                $recordset = $this->conn->query("SELECT * FROM students");
                $dataset = $recordset->fetch_all(MYSQLI_ASSOC);
                $this->conn->close();
                return $dataset;
          }
          return null;
       }
    }
?>
```