Documentation

Inhalt

1. Component System

1.1 Service Class

(OPTIONAL)

First, create the service file for the component in the services folder. Name it something.service.php and load it in the index.php at the top.

In your newly created service file, paste the following code:

```php
<?php
class MyService {

    /** @var MyService  */
    public static $instance;

    function __construct()
    {

    }
}

// INIT SERVICE
MyService::$instance = new MyService ();
```

This is the basic structure of your service.

If you need specific dependencies in your class, you can add them via the constructor:

```php
<?php
class MyService {

    /** @var MyService  */
    public static $instance;
    /** @var Database  */
    protected $db;

    function __construct(Database $database)
    {
        $this->db = $database;
    }
}

// INIT SERVICE
MyService::$instance = new MyService(Database::$instance);
```

The service in instanced once and has only member functions for easier debugging and stateful development. You should also know that you do not need a new service for every component.

1.2 Component Class

(REQUIRED)

Now create the component file. Name it something.component.php and copy the following code.

```php
class MyComponent
{

    /** @var MyComponent */
    public static $instance;

    function __construct()
    {

    }

}

// INIT COMPONENT
MyComponent::$instance = new MyComponent();
```

The components can react depending on your constructor and call functions from the service classes.

For example, a reactive constructor for GET variables could look like this:

```
function __construct()
{
    if (!isset($_GET['foo']))
        return;

    switch ($_GET['foo']) {
        case "bar":
        MyService::$instance->bar($_GET['foo']);
        break;
    }
}
```

You can also write functions in your component for specific rendering or other “only-component” related functionalities.

1.3 Visibility/Linking

(REQUIRED)

Finally, the component needs to be included at the right permission level. To do this, go to the index.php and insert the require_once statement in the correct spot.

```
<!-- Main container -->
    <div class="container">
        <?php
        /// Load components
        // Load logged in components
        if ($loggedIn) {
            $accessStrength = $userType->getAccessStrength();

            // Normal user components
            if ($accessStrength >= UserType::User()->getAccessStrength()) {
                require_once "components/user.component.php";
            }
            // Game creator components
            if ($accessStrength >= UserType::Creator()->getAccessStrength()) {
                require_once "components/creator.component.php";
            }
            // Admin components
            if ($accessStrength >= UserType::Admin()->getAccessStrength()) {
                require_once "components/admin.component.php";
            }
        } else {
            // if someone isn´t logged in
            require_once "components/public.component.php";
        }
        ?>
    </div>
```

2. Adding new UserType

If you want to add a new UserType to your website, simply extend the constructor with your type name and add the static creation function.

For example add into constructor if statements:

```
else if (preg_match("/^MyUserType$/i", $type)) {
            $this->typeString = "MyUserType";
            $this->accessStrength = 42;
}
```

And add into class:

```
public static function MyUserType()
{
    return new UserType("MyUserType");
}
```

After that, you can load new components in the index.php via the access strength of your newly created usertype with

```
// MyUserType components
if ($accessStrength >= UserType::MyUserType()->getAccessStrength()) {
    require_once "components/MyUserType.component.php";
}
```

3. Models

Models are very simple to understand. They are just containers that hold data. For example for Users, Games, Pictures, etc.

Feel free to implement new models for a better structure.