Prueba técnica para AppliedIT

Adrián Crespo Delgado

Abril 2023

Índice

1. Instalación del proyecto

1.1. Descarga

El código fuente del proyecto se encuentra alojado en Github, en la url: `https://github.com/appliedit/prueba-php-adrian.git`. Para descargarlo, se puede descargar como ZIP y descomprimir posteriormente o hacerlo a través de git con el comando:

```
git clone https://github.com/appliedit/prueba-php-adrian.git
```

1.2. Instalación en xampp

Para su ejecución es necesario utilizar una aplicación de emulación local. Se recomienda xampp, ua que ha sido la utilizada para esta prueba. Una vez instalado el programa, navegar hasta:

```
Ruta instalación xampp/htdocs/
```

En esta ruta se recomienda crear la carpeta 'appliedit' y alojar el contenido de git en su interior. Si se desea llamar a la carpeta de otra forma, para el correcto funcionamiento del proyecto será necesario modificar una variable en al archivo .htaccess:

```
SetEnv SERVER appliedit
```

Modificar 'appliedit' por el nombre de la carpeta creada en htdocs.

1.3. Despliegue

Para desplegar la aplicación, abrir xampp y lanzar el servidor de Apache y MySQL. Lanzado el servidor, navegar hasta `localhost/phpmyadmin` e importar el script 'script.sql' que se encuentra en la carpeta de instalación. Una vez creada la base de datos, el proyecto está listo para su uso.

2. Estructura

Para la creación de este proyecto se ha utilizado una versión en PHP nativo del modelo-vista-controlador.

El proyecto se divide en:

2.0.1. assets

En este módulo se encuentran las imágenes, css, y archivos js que usará el proyecto.

2.0.2. controllers

Este módulo se encarga de la lógica de funcionamiento de la aplicación. Aquí recibimos los datos obtenidos de los formularios en las vistas y los procesamos, utilizando funciones de validación y funciones propias de los modelos creados. Se usan variables de sesión[1] para la vista de datos.

Ejemplo:

```php
<?php

if(isset($_POST['email']) && !empty($_POST['email']) &&
    isset($_POST['password']) && !empty($_POST['password'])) :

    if(filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) :
        $email = $_POST['email'];

    else :
        $_SESSION['error'] = "Your email is not valid, please try again.";

    endif;

    $password = $_POST['password'];

    $userMod = new UserModel($email, $password);
    $user = $userMod->logInUser();

    if($user) :
        $_SESSION['user'] = $userMod->getEmail();
        header('Location: /'.getenv('SERVER').'/index');
```

[1]Nota: Al no hacer uso de frameworks con librerías que unan las vistas con los controladores, los datos se pasan a través de variables de sesión a las cuales no puede acceder el usuario. De hacer uso de un framework, los datos se pasarían a través de json o estructuras similares.

```
    else :
        $_SESSION['error'] = "Your email or password is incorrect, please try agai
    endif;

elseif(isset($_POST['email']) && empty($_POST['email']) || isset($_POST['password'
    $_SESSION['error'] = "Complete the form before send it.";

endif;

?>
```

2.0.3. models

En este módulo almacenamos los datos de cada clase del proyecto, y creamos funciones propias de cada modelo para utilizarlas en el controlador modularizando las funcionalidades del crud. Las funciones aquí creadas son relacionadas con el propio modelo y su tabla en la base de datos correspondiente.

Un ejemplo de código podría ser el siguiente:

```
<?php

class UserModel {

    public $db;
    private $table = 'user';

    private $email;
    private $username;
    private $password;
    private $admin;

    public function __construct($email, $password = null) {
        $con = "mysql:host=".getenv('DB_HOST').";dbname=".getenv('DB_NAME');
        $this->db = new PDO($con, getenv('DB_USER'), '');

        //Encriptado md5 para password
```

```
        if($password) : $password = hash('md5', $password); endif;

        $this->email = $email;
        $this->password = $password;
    }

    public function getEmail() {
        return $this->email;
    }

    public function createUser() {
        $sql = "INSERT INTO " . $this->table . " (email, username, password)
        VALUES (?, ?, ?)";
        $stmt = $this->db->prepare($sql);
        $stmt->execute(array($this->email, $this->username, $this->password));

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }
}

?>
```

2.0.4. views

Este módulo se compone de las vistas de cada uno de los formularios o tablas del proyecto, además de elementos comunes tales como el header, footer y sidebar, que están modularizados para la consistencia del código, evitando duplicados o errores de cohesión.

Ejemplo de leftsidebar.php, común a todas las vistas:

```
<div id="leftsidebar">
    <div class="white-bg">
        <div class="text-center">
            <img src="assets/img/logo_login.svg" class="logo img-responsive">
        </div>
        <div class="menu">
            <h2>Administration</h2>
```

```
            <ul>
                <li><a href="index"><img src="assets/img/icon_companies.svg" class
            </ul>
        </div>
        <div class="footer fixed-bottom">
            <div class="d-block">
                <button class="btn" id="logout"><img src="assets/img/icon_logout.s
            </div>
            <img src="assets/img/powered.svg" class="img-responsive powered">
        </div>
    </div>
</div>
```

2.0.5. rutas

Las rutas son la unión principal de cada uno de los elementos anteriores. Se encargan de unir modelo-vista-controlador cuando un usuario navega a cada una de las páginas. Para comodidad en la navegación, se ha añadido una regla en el archivo .htaccess para que el usuario pueda utilizar estas rutas sin la extensión .php
Ejemplo de enrutamiento:

```php
<?php

session_start();

if(isset($_SESSION['user']) && !empty($_SESSION['user'])) :
    header('Location: /appliedit/index');

else :
    require_once("controllers/LoginController.php");

    include("views/layouts/components/header.html");
    include("views/layouts/LoginView.php");
    include("views/layouts/components/footer.html");

endif;

?>
```

3. Tecnologías

En este proyecto se han utilizado las siguientes tecnologías:

3.0.1. MySQL

Ha sido la tecnología elegida para almacenar los datos. Las tablas se han normalizado hasta la 3 forma normal, obteniendo las tablas:

- Activity
- Country
- Company
- User

```
CREATE DATABASE IF NOT EXISTS appliedit_prueba;
USE appliedit_prueba;

CREATE TABLE IF NOT EXISTS `User` (
    id INT PRIMARY KEY AUTO_INCREMENT ,
    email VARCHAR(255) NOT NULL UNIQUE,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    admin BIT DEFAULT 0
);

CREATE TABLE IF NOT EXISTS `Country` (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Activity` (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Company` (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    tax_number VARCHAR(255) NOT NULL,
    employees INT,
    countryId INT,
    zip_code VARCHAR(255),
    email VARCHAR(255),
    phone_number VARCHAR(255),
    activityId INT,
    risk BOOLEAN,
    payment ENUM('Bank', 'Credit', 'Cash'),
    FOREIGN KEY (countryId) REFERENCES Country(id),
    FOREIGN KEY (activityId) REFERENCES Activity(id)
);

INSERT INTO `User` (`email`,`username`,`password`,`admin`) VALUES ('admin@admin.co
INSERT INTO `Country` (`name`) VALUES ('Spain'),('France'),('England'),('Germany')
INSERT INTO `Activity` (`name`) VALUES ('Insurance'),('Advisory'),('Invest'),('Eng
```

3.0.2. Bootstrap 5

A través de bootstrap gestionamos la estructura y css de las vistas. La elección de bootstrap 5 ha sido porque añade facilidades para utilizar select switches y una sintaxis más claras de notificaciones y modales.

3.0.3. PHP 7.4

A través de PHP se hacen las peticiones a los controladores para obtener y mostrar datos en las vistas. El proyecto en general está realizado con esta tecnología.

```
public function logInUser() {
        $sql = "SELECT * FROM " . $this->table . " WHERE email = ? AND password =
        $stmt = $this->db->prepare($sql);
        $stmt->execute(array($this->email, $this->password));
        $res = $stmt->fetch(PDO::FETCH_ASSOC);

        if($res) :
            $this->username = $res['username'];
```

```
            $this->admin = $res['admin'];

        endif;

        return $res;
    }
```

Para las llamadas a la base de datos se ha prescindido de mysqli, y se ha optado por el PDO de php. Se ha elegido este gestor de transacciones permite el uso de otras bases de datos además de las sql, y se ha priorizado su uso por la seguridad que tiene frente a ataques a la hora de realizar consultas en la BD.

3.0.4. Js y Jquery

Para añadir funcionalidad extra al proyecto, se han utilizado funciones js y jquery para algunas validaciones de datos en el lado del cliente. Aunque en general se han creado funciones para el proyecto, también se ha realizado una función click en 'logout', para tener variedad en el desarrollo del proyecto. Ejemplo:

```
$(document).ready(function(){

    $("#logout").click(function(event) {
        event.preventDefault();
        $.ajax({
            type: 'POST',
            url: 'logout.php',
            data: { ok : 1 },
            success: function() {
                window.location.href = 'login';
            }
        });
    });

});
```

3.0.5. Ajax

Algunas de las peticiones de formularios de la aplicación están realizadas con esta tecnología, ya que ofrece una respuesta asíncrona y podemos validad/mostrar datos sin necesidad de recargar la url. Se ha utilizado en las funciones 'logout', 'viewcompany' y 'deletecompany'. Ejemplo:

```
function viewCompany(id) {
    $.ajax({
        type: 'POST',
        url: 'index.php',
        data: {
            id: id,
            view : 1
        },
        success: function(response) {
            console.log(response);
            window.location.href = "panel";
        },
        error: function(xhr, status, error) {
            console.log(error);
        }
    });
}
```

4. Validaciones

Las validaciones se han realizado tanto en lado del cliente como en el lado del servidor. Para el lado del cliente se han utilizado los atributo 'type' en los inputs, además de validaciones a través de js, comprobando valores nulos y haciendo uso de expresiones regulares. Ejemplo:

```
function addCompany(id) {
    $errors = false;
    $name = $('#companyForm input[name="name"]');
    $country = $('#companyForm select[name="country"]');
    $activity = $('#companyForm select[name="activity"]');
```

```
    $taxNumber = $('#companyForm input[name="tax_number"]');
    $email = $('#companyForm input[name="email"]');
    $employees = $('#companyForm input[name="employees"]');
    $phoneNumber = $('#companyForm input[name="phone_number"]');
    $zipCode = $('#companyForm input[name="zip_code"]');

    var regex_email =
    /^([a-zA-Z0-9_.+-])+\@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/;
    var regex_taxNumber =
    /^[A-Z]\d{8}$/i;
    var regex_phoneNumber =
    /^(\+\d{1,3}[\s-]?)?\(?\d{3}\)?[\s-]?\d{3}[\s-]?\d{3,4}$/;

    $name.toggleClass('is-invalid', !$name.val());
    $country.toggleClass('is-invalid', !$country.val() ||
        $country.val() == 0);
    $activity.toggleClass('is-invalid', !$activity.val() ||
        $activity.val() == 0);
    $taxNumber.toggleClass('is-invalid', !$taxNumber.val() ||
        !regex_taxNumber.test($taxNumber.val()));
    $email.toggleClass('is-invalid', !$email.val() ||
        !regex_email.test($email.val()));
    $employees.toggleClass('is-invalid', !$employees.val() ||
        isNaN(Number($employees.val())));
    $phoneNumber.toggleClass('is-invalid', !$phoneNumber.val() ||
        !regex_phoneNumber.test($phoneNumber.val()));
    $zipCode.toggleClass('is-invalid', !$zipCode.val());

    if($('.is-invalid').length) {
        $errors = true;
    }

    if(!$errors) {
        $('#submit-button').click();
    }
}
```

De la misma forma, en el lado servidor a través de los controladores tam-

bién se han realizado validaciones. Estas validaciones hacen uso de comprobación de datos nulos, expresiones regulares y posterior manejo de mensajes de errores. Las validaciones realizadas han sido las siguientes:

```php
<?php

function isValidId($id, $array) {
    $idArray = array_column($array, "id");
    return in_array($id, $idArray);
}

function validateRegex($value, $regex, $errorMsg) {
    $res = preg_match($regex, $value);

    if (!$res) :
        $_SESSION['error'] = $errorMsg;
    endif;

    return $res;
}

function validateField($value, $errorMsg, $type = null) {
    if(empty($value)) :
        $_SESSION['error'] = $errorMsg;

        return false;

    elseif($type) :
        switch($type) :
            case 'email' :
                validateRegex($value,
                '/^([a-zA-Z0-9_.+-])+\@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/',
                $errorMsg);
            break;

            case 'tax_number' :
                validateRegex($value,
                '/^[A-Z]\d{8}$/i', $errorMsg);
```

```
            break;

            case 'phone_number' :
                validateRegex($value,
                '/^(\+\d{1,3}[\s-]?)?\(?\d{3}\)?[\s-]?\d{3}[\s-]?\d{3,4}$/',
                $errorMsg);
            break;

            case 'employees' :
                if(is_numeric($value)) :
                    return true;

                else :
                    $_SESSION['error'] = $errorMsg;

                endif;
            break;
        endswitch;
    else :
        return true;

    endif;
}

?>
```

- **Email**: Para comprobar que este campo sea válido se ha utilizado la siguiente función de php, además de la expresión regular:

  `FILTER_VALIDATE_EMAIL`
  `'/^([a-zA-Z0-9_.+-])+\@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/'`

- **Password**: Aquí no se ha realizado una validación de datos, sin embargo para evitar posibles ataques tales como sql injection, se ha utilizado la tecnología PDO de PHP en todo el proyecto, que protege las sentencias sql antes de ejecutarlas. Este campo cuenta con encriptado md5, por lo que una función se encarga de encriptar la contraseña para más seguridad y hacer la posterior comprobación.

Si bien no se ha realizado una validación de este campo, se ha añadido a modo de comentario una posible función PHP que valida a través de una expresión regular que este campo cumpla con al menos 8 caracteres, entre los cuales debe haber minúscula, mayúscula, un número y un caracter especial:

```
function validate_password($password) {
    $regex = '/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[^a-zA-Z0-9]).{8,}$/';
    return preg_match($regex, $password);
}
```

- **Name**: La comprobación de este campo consiste en que no sea nulo o esté vacío.
- **Country**: La comprobación de este campo consiste en que no sea nulo o esté vacío, o no sea el valor por defecto del formulario.
- **Activity**: La comprobación de este campo consiste en que no sea nulo o esté vacío, o no sea el valor por defecto del formulario.
- **Tax Number**: La comprobación de este campo consiste en que no sea nulo o esté vacío, y que siga el siguiente patrón: Una letra y 8 números.

  ```
  '/^[A-Z]\d{8}$/i'
  ```

- **Phone Number**: La comprobación de este campo consiste en que no sea nulo o esté vacío, y que siga el patrón de números válidos en Europa.

  ```
  '/^(\+\d{1,3}[\s-]?)?\(?\d{3}\)?[\s-]?\d{3}[\s-]?\d{3,4}$/'
  ```

- **Employees**: La comprobación de este campo consiste en que no sea nulo o esté vacío, y sea un número válido.
- **Zip Code**: Este campo no tiene validación, ya que no hay un consorcio claro en la forma que tiene en los distintos países. Es un campo de tipo varchar, ya que aunque en España sea un valor numérico, en otros países se compone de letras y números, no siguiendo un patrón estándar.

5. Uso

Para testear la aplicación se invita a probar el responsive, la validación de los datos, la entrada sin logueo en las distintas urls. La url principal del proyecto es /index, que muestra el panel de companies. Sin embargo, no es posible acceder a este panel sin loguearse previamente, por lo que será redireccionado a /login. Las urls están protegidas, no es posible acceder a ellas sin una sesión de usuario activa.

Una vez se haya iniciado sesión, será redireccionado hacia el panel de gestión principal '/index'. Desde aquí podremos crear una nueva empresa, redireccionándonos. El formulario avisará de los posibles errores en la creación de la misma, y con una notificación en caso de que haya sido válido. Una vez creada la empresa, podremos volver al panel global de gestión, y desde aquí tendremos la posibilidad de editar o borrar la misma. Si elegimos editar, seremos redireccionados a un formulario similar al de creación, pero que contiene los datos de la empresa a editar, notificando del éxito o el error en la operación. Si decidimos eliminar la empresa, tanto desde la tabla como desde el formulario de edición, un popup nos preguntará si queremos eliminarla, y seremos notificados si se elimina con éxito.