Medicare

Desarrollado por Cardarelli Guido

Indice:

Tecnologias utilizadas:

Para este proyecto utilice las siguientes tecnologias:

- HTML
- CSS
- JavaScript
- PHP
- MySQL

Instalacion

PHP version: 8.2.4

Apache version: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4

- Mover carpeta `/medicare` al directorio `htdocs` donde se encuentre instalado apache.
- Modificar constantes de entorno en `/medicare/utils/config.php` , con las correspondientes credenciales según configuración local para acceder a la base de datos MySQL.

```
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASSWORD', '');
define('DB_NAME', 'medicare');
```

- Crear base de datos

```
CREATE DATABASE medicare
```

- Una vez creada la base de datos importar el archivo `medicareExport.sql` que se encuentra en `/medicare/database` para generar la estructura de la misma.

Estructura de archivos del proyecto

```
.
└── Medicare/
    ├── assets
    ├── controllers/
    │   ├── HomeController.php
    │   ├── LoginController.php
    │   ├── PrescriptionController.php
    │   ├── ProfessionalController.php
    │   └── RegisterController.php
    ├── database/
    │   └── database.sql
    ├── js/
    │   └── utils.js
    ├── models/
    │   ├── EntityModel.php
    │   ├── PatientModel.php
    │   ├── PrescriptionModel.php
    │   ├── ProfessionalModel.php
    │   ├── SpecialtyModel.php
    │   └── UserModel.php
    ├── utils/
    │   ├── autoload.php
    │   ├── config.php
    │   ├── functions.php
    │   └── routes.php
    ├── views/
```

```
    ├── layout/
    │   ├── start.php  (header)
    │   └── nav.php
    ├── finish.php (footer)
    ├── login/
    │   └── form.php
    ├── prescriptions/
    │   ├── details.php
    │   ├── edit.php
    │   ├── list.php
    │   └── new.php
    ├── professionals/
    │   ├── details.php
    │   ├── edit.php
    │   ├── list.php
    │   └── new.php
    ├── register/
    │   └── form.php
    ├── contact.php
    ├── error.php
    └── landing.php
```

Arquitectura MVC

La arquitectura Modelo-Vista-Controlador (MVC) es un patrón de diseño de software que se utiliza para organizar y estructurar aplicaciones web. Se divide en tres componentes principales:

- **Modelo**: Representa los datos y la lógica de negocios de la aplicación. Se encarga de interactuar con la base de datos y realizar operaciones relacionadas con los datos.
- **Vista**: Es la capa visual de la aplicación. Se encarga de mostrar la información al usuario y de recibir sus interacciones. Puede ser una página web, una interfaz gráfica, o cualquier otro medio de presentación.
- **Controlador**: Actúa como intermediario entre el modelo y la vista. Recibe las solicitudes del usuario, procesa la lógica de negocio y actualiza el modelo si es necesario. Luego, selecciona la vista correspondiente para mostrar los resultados al usuario.

La arquitectura MVC permite separar las responsabilidades y facilita el mantenimiento y la escalabilidad de la aplicación. Cada componente tiene un propósito específico y puede ser modificado o reemplazado sin afectar a los demás. Esto promueve el desarrollo modular y la reutilización de código.

Utils

El directorio `utils` contiene archivos y funciones que son utilizados como utilidades en el proyecto. Estas utilidades pueden incluir funciones comunes, clases de ayuda o cualquier otro código reutilizable que se utiliza en diferentes partes del proyecto.

En el contexto de este proyecto, el directorio `utils` puede contener archivos y funciones relacionados con la manipulación de rutas, validaciones de datos, funciones de utilidad para la base de datos, funciones de manejo de errores, entre otros.

El propósito principal del directorio `utils` es centralizar y organizar el código que se utiliza en varias partes del proyecto, brindando una forma conveniente de acceder y reutilizar estas utilidades en diferentes componentes del sistema.

Routes.php

Este es un archivo fundamental en el proyecto ya que funciona como enrutador de nuestra aplicacion y se encarga de validar todos los parametros que lleguen por url.

Este archivo retorna una funcion anonima que en su interior contiene todas las validaciones correspondientes, para asegurar la correcta navegacion y ejecucion de acciones en la aplicacion web.

La url de la aplicacion esta pensada de la siguiente manera:

```
medicare.com/{category}/{id}/{a}
```

En donde:

category: representa un `controller` de nuestra aplicacion con el que se quiere interactuar, el protagonista de la seccion.

id: un identificador unico numerico para cada una de las instancias del modelo que se encuentra bajo esta categoria. Su uso fundamental es la busqueda e identificacion de estos registros en la base de datos.

a: la accion que se desee realizar. Ya sea crear, modificar, o eliminar `new | edit | delete`

En base a estos parametros, lo que se va a validar es la existenia de un `controller` bajo el nombre de la <**category**>. Para luego en caso de que este exista , ejecutar la accion <**a**> solicitada.

```
<?php
return function(){
  // Extraemos parametros de la url
  // ...
  //validaciones previas
  if(! class_exists($controller)){
    return [
      'data'=> 'Lo sentimos, el vinculo al que desea ingresar no existe',
      'view'=> 'error'
    ];
  }

  /*Si todo va bien, y se pasan las validaciones
  llamamos a la accion del controller correspondiente*/
```

```
  return $controller::$action()
}
```

De no existir se retornara un arreglo que contendra el error correspondiente y la a mostrarse.

Functions.php

Contiene una serie de funciones que seran utilizadas a en varias partes del proyecto, sirven a modo de utilidad.

Algunos ejemplos: `isLogged()` , `isPost()` , `hasEmptyFields()`

Config.php

En este archivo se definen algunas constantes globales que seran de gran utilidad para configurar el proyecto, como los datos de coneccion a la base de datos, las direcciones del sistema de ficheros para algunas rutas utilizadas con frecuencia como pueden ser `/views` o la direccion base del proyecto.

Ademas se configura la muestra de errores y se define la zona horaria en caso de utilizar los metodos de fechas que provee php.

Autoload.php

Este archivo permite autoimprotar las diferentes clases creadas en el proyecto proyecto. Esto es de gran utilidad ya que nos permite evitar errores por falta de importaciones o errores en los nombres de las mismas. Por otra parte facilita el incremento progresivo del proyecto.

Para esto se utiliza el metodo `glob()` que busca todos los nombres de ruta que coinciden con un patron que le pasemos como parametro.

Controller

Cuenta con una serie de metodos, que se encargan de la logica de negocio y que retornan las diferentes vistas a renderizar en la aplicacion.

View

Representan la capa visual de la aplicacion, es decir aquella en la cual interactua el usuario, siendo esta fundamentalmente una serie de “plantillas html” que dependiendo algunas caracteristicas de la misma, renderizan contenido condicionalmente o listan informacion almacenada en nuestra base de datos.

htaccess

Los archivos .htaccess son archivos de texto que nos permiten definir como debera responder un servidor web ante las peticiones de los usuarios

Esto nos permite general url amigables para el usuario, que sean de simple acceso y ademas que nos permitan tener una mejor organizacion en nuestro codigo.

```
// inicia el motor de re-escritura de urls del servidor
RewriteEngine = On
// regla que relaciona una url falsa a un archivo real del filesystem.
RewriteRule url_falsa | archivo_real

//Ejemplo

RewriteRule professional index.php?cat=professional

// Esto quiere decir si vamos a http://localhost/80/{directorio-proyecto}/professional
// apache ejecutara el archivo index.php enviando como parametros en la peticion get cat = professional
```

Expreciones regulares en htaccess

Tambien se pueden usar expreciones regulares, para asi poder tener un control mas exhaustivo de como manejamos las url de nuestra aplicacion.

```
RewriteRule (professional|prescription) index.php?cat=$1

Cada parentesis se puede bindear a un ($n), teniendo asi una relacion 1 a 1 () y $n

ej (()) $1 $2
```

Model

Los modelos representan las entidades de la base de datos y contienen los métodos para interactuar con ellas. Cada modelo tiene un conjunto de métodos para realizar operaciones básicas como obtener todos los registros, obtener un registro por su ID, verificar la existencia de un registro y crear un nuevo registro.

EntityModel

Contiene las metodos para las 4 interacciones basicas con la base de datos. Select, Insert, Update. Ademas este tiene la conneccion con la base de datos, permitiendo asi, encapsular la logica de acceso a datos.

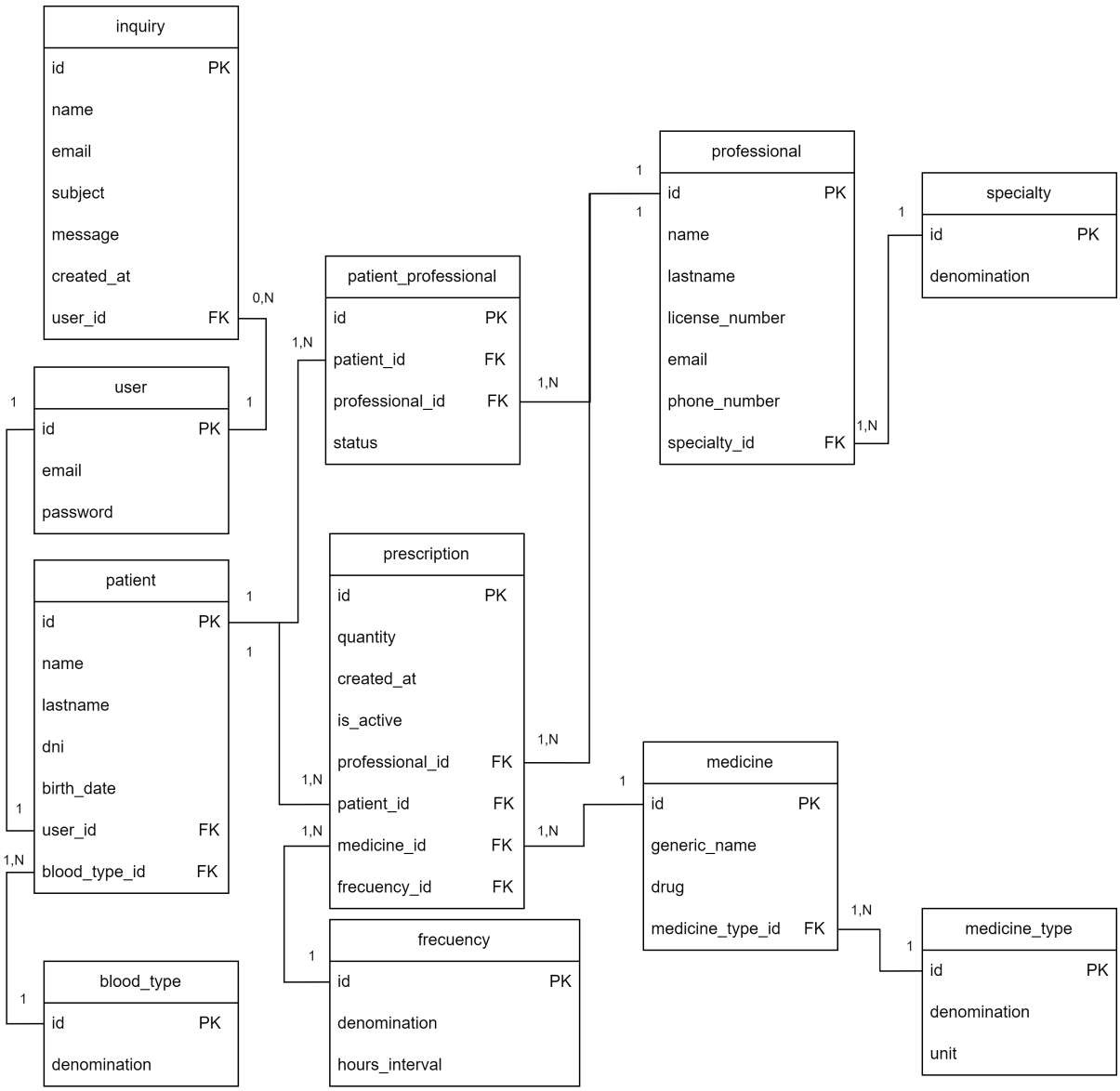
```
public function select(
    $columns = '*', // String con las columnas a seleccionar
    $filters = [],  // Joins, where, order by, group by
    $onlyOneResult = false // Bandera que define si se devuelve un unico registro o mas
```

```
  ) {
  // Implementación select...
}

public function insert($data){
  // Implementación insert...
}

public function update($data, $id){
  // Implementación update...
}

public function delete($id){
  // Implementación deleet...
}

// Coneccion a la base de datos
private function connect(){}
```

Base de datos

Para este proyecto utilice una base de datos relacional SQL por medio del motor MySQL. La misma cuenta con la siguiente estructura:

Diagrama relacional


inquiry
id PK
name
email
subject
message
created_at
user_id FK
0,N
1,N
patient_professional
id PK
patient_id FK
professional_id FK
status
1,N
professional
id PK
name
lastname
license_number
email
phone_number
specialty_id FK
1,N
specialty
id PK
denomination
user
id PK
email
password
patient
id PK
name
lastname
dni
birth_date
user_id FK
blood_type_id FK
prescription
id PK
quantity
created_at
is_active
professional_id FK
patient_id FK
medicine_id FK
frecuency_id FK
medicine
id PK
generic_name
drug
medicine_type_id FK
medicine_type
id PK
denomination
unit
blood_type
id PK
denomination
frecuency
id PK
denomination
hours_interval