



MANUAL TÉCNICO

INDICE

Introducción	i
Objetivo General	1
Objetivos Específicos	1
Estructura del Sistema	1
Justificación	1
Justificación de Hardware	1
Justificación de Software.....	1
. Creación de Base de Datos:	3
Diagrama de la Base de Datos:.....	3
Creación de Base de Datos.....	4
Conexión de BD a CodeIgniter.....	4
Controllers	5
1. Constructor __construct()	5
2. Función index()	5
3. Función agregar().....	5
4. Función guardar_estacion()	5
5. Función eliminar().....	6
6. Función iniciar_tiempo_normal()	6
7. Función iniciar_tiempo_regresivo()	6
8. Función detener_tiempo_regresivo()	6
9. Función detener_tiempo_normal()	6
Models.....	7
views.....	8
Flujo de la aplicación:	9
welcome_message.php.....	10
AWS	13
Conclusión	22

Introducción

El presente manual técnico detalla el desarrollo de un sistema de gestión de tiempo diseñado para estaciones de trabajo en entornos compartidos, como cafés internet. Este sistema permite a los usuarios monitorear y controlar el tiempo que utilizan en cada estación de manera individual a través de una interfaz gráfica moderna, intuitiva y eficiente. A lo largo del documento se presentan los aspectos técnicos de su implementación, los componentes que forman parte de su arquitectura, las tecnologías empleadas y la funcionalidad que ofrece el sistema tanto para usuarios como para administradores. La justificación del uso de hardware y software también se aborda, con el objetivo de garantizar que el sistema sea escalable, accesible y adaptable a diversos e

Objetivo General

Desarrollar un sistema de gestión de tiempo para estaciones en un entorno compartido (como cafés internet), que permita a los usuarios controlar de manera individual el tiempo que utilizan cada estación a través de una interfaz moderna, intuitiva y fácil de usar.

Objetivos Específicos

Implementar un sistema que soporte múltiples estaciones de manera simultánea, permitiendo gestionar de forma independiente el tiempo de cada una.

Desarrollar una interfaz gráfica atractiva que permita al administrador y a los usuarios interactuar con los cronómetros de manera eficiente y sin complicaciones.

Estructura del Sistema

Arquitectura General:

El sistema está basado en una arquitectura cliente-servidor. El frontend (HTML, CSS, JavaScript) proporciona una interfaz gráfica atractiva y fácil de usar, mientras que el backend, implementado en PHP utilizando CodeIgniter, maneja la lógica del sistema y las interacciones con la base de datos MySQL.

Justificación

Justificación de Hardware

El sistema ha sido diseñado para funcionar en infraestructuras estándar, lo que lo hace accesible y fácilmente adaptable en entornos comerciales o domésticos sin requerir hardware especializado.

Justificación de Software

El software utilizado para el desarrollo del sistema ha sido seleccionado para garantizar la eficiencia, escalabilidad y facilidad de implementación. A continuación, se describe la justificación de cada componente:

Frontend (HTML, CSS, JavaScript): El uso de tecnologías web estándar asegura que el sistema sea accesible desde cualquier dispositivo con un navegador moderno, sin la necesidad de instalar software adicional. Esto facilita el uso para el usuario final y reduce los costos de mantenimiento.



Backend (CodeIgniter- PHP): CodeIgniter es un framework ligero y flexible que permite una rápida implementación del backend. Al ser de código abierto y tener una comunidad activa, facilita el mantenimiento y la escalabilidad del proyecto. PHP, el lenguaje subyacente, es ampliamente utilizado y soportado en la mayoría de los entornos de servidores, lo que lo convierte en una opción sólida y confiable.



Base de Datos (MySQL): MySQL es una base de datos relacional muy estable y eficiente para manejar grandes volúmenes de datos. Su capacidad de integrar fácilmente consultas complejas y su amplia adopción en el mercado garantizan que el sistema pueda crecer y manejar más estaciones y usuarios sin comprometer el rendimiento.

XAMPP: Este paquete que combina Apache, MySQL, PHP y Perl es ideal para entornos de desarrollo locales. Es fácil de instalar y configurar, y proporciona un entorno de servidor completo sin la necesidad de implementar un servidor remoto durante la fase de desarrollo.



Visual Studio Code: Este editor de código se utiliza para el desarrollo del proyecto. Es ligero, de código abierto y tiene un vasto ecosistema de extensiones que facilita la programación, depuración y control de versiones del sistema.



AJAX (Asynchronous JavaScript and XML) es una técnica de desarrollo web que permite a las aplicaciones web comunicarse con el servidor y actualizar partes de una página web de manera asíncrona, sin necesidad de recargar la página completa. Esto significa que se pueden enviar y recibir datos del servidor en segundo plano, y actualizar el contenido de la página con esos datos sin interrumpir la interacción del usuario.



. Creación de Base de Datos:

Diagrama de la Base de Datos:

estaciones

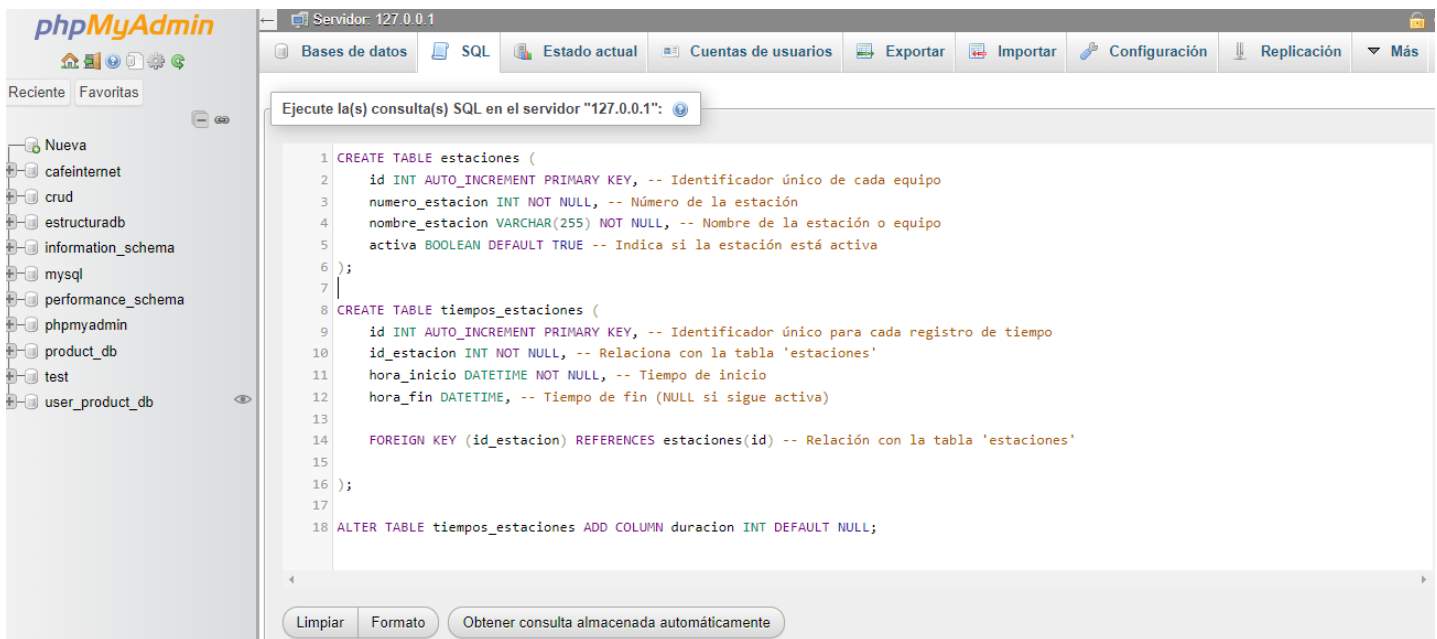
Columna	Tipo	Nulo
id (Primaria)	int(11)	No
numero_estacion	int(11)	No
nombre_estacion	varchar(255)	No
activa	tinyint(1)	Sí

tiempos_estaciones

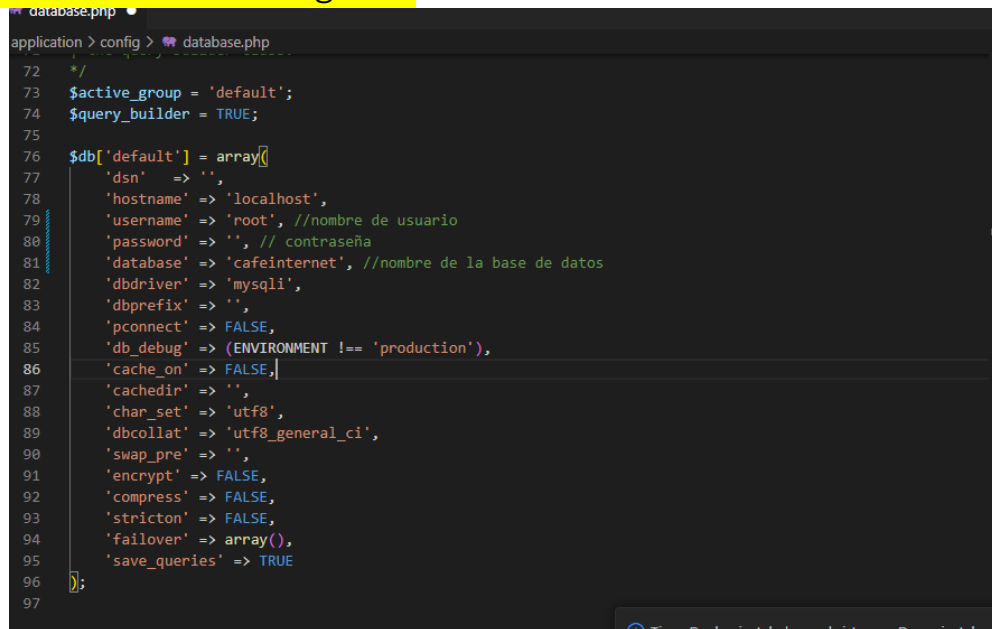
Columna	Tipo	Nulo
id (Primaria)	int(11)	No
id_estacion	int(11)	No
hora_inicio	datetime	No
hora_fin	datetime	Sí
duracion	int(11)	Sí



Creación de Base de Datos.



Conexión de BD a CodeIgniter



Controllers

Los **controllers** reciben las solicitudes del usuario a través de la URL, procesan la lógica necesaria (consultar bases de datos, aplicar reglas de negocio, etc.) y luego pasan los datos resultantes a la vista para su presentación. Básicamente, controlan el flujo de la aplicación y manejan las interacciones del usuario.

Welcome.php tiene múltiples funciones que están diseñadas para manejar la lógica de la aplicación relacionada con la gestión de estaciones y la monitorización de tiempos (cronómetros y temporizadores). A continuación, te detallo cada sección del código:

1. Constructor `__construct()`

- Carga el modelo `Estacion_model` para interactuar con la base de datos y realizar operaciones específicas relacionadas con las estaciones.
- Este modelo es necesario para realizar consultas y manipulaciones en la base de datos, como la inserción o actualización de registros.

2. Función `index()`

- Recupera todas las estaciones que están activas (cuya columna activa es igual a 1) desde la base de datos.
- Para cada estación activa, verifica si hay un tiempo activo (es decir, si ya hay un cronómetro en marcha y no ha terminado).
- Carga la vista `welcome_message` pasando los datos de las estaciones, que incluye si hay un cronómetro o temporizador activo para cada estación.

3. Función `agregar()`

- Muestra un formulario para agregar una nueva estación.
- Carga la vista `agregar_estacion`, que probablemente contendrá el formulario de ingreso de datos para una nueva estación.

4. Función `guardar_estacion()`

- Recoge los datos ingresados en el formulario de la función `agregar()` (número de estación y nombre).
- Inserta esos datos en la base de datos utilizando el modelo `Estacion_model`.
- Redirige a la página principal (`welcome`) una vez que se ha guardado correctamente la nueva estación.

5. Función eliminar()

Desactiva una estación cambiando el valor de su columna activa a 0 (sin eliminarla completamente de la base de datos).

- Redirige a la página principal después de desactivar la estación.

6. Función iniciar_tiempo_normal()

- Comienza un cronómetro normal para una estación específica.
- Verifica si ya hay un cronómetro activo para esa estación (un registro sin hora_fin).
- Si no hay uno activo, inserta un nuevo registro con la hora de inicio.
- Retorna una respuesta en formato JSON indicando si el cronómetro fue iniciado con éxito o si ya había un tiempo en curso.

7. Función iniciar_tiempo_regresivo()

- Comienza un temporizador regresivo para una estación específica con una duración definida.
- Similar a la función de cronómetro normal, verifica si ya hay un tiempo en curso para la estación.
- Si no hay uno activo, inserta un nuevo registro con la hora de inicio y la duración del temporizador.
- Retorna una respuesta JSON indicando si el temporizador fue iniciado correctamente.

8. Función detener_tiempo_regresivo()

- Detiene un temporizador regresivo que está en curso para una estación específica.
- Calcula la duración total desde el inicio hasta el momento actual.
- Actualiza el registro de tiempo en la base de datos con la hora_fin y la duración total en segundos.
- Retorna una respuesta JSON indicando si el temporizador fue detenido correctamente.

9. Función detener_tiempo_normal()

- Similar a la función detener_tiempo_regresivo(), pero se utiliza para detener cronómetros normales.
- Calcula la duración total desde que comenzó el cronómetro y actualiza la base de datos con la hora de fin y la duración en segundos.

- Retorna una respuesta JSON indicando si el cronómetro fue detenido.

```

1  <?php
2  defined('BASEPATH') OR exit('No direct script access allowed');
3
4  class Welcome extends CI_Controller {
5
6      public function __construct() {
7          parent::__construct();
8          $this->load->model('Estacion_model'); // Cargar el modelo
9      }
10
11     public function index() {
12         // Obtener estaciones activas
13         $this->db->where('activa', 1);
14         $estaciones = $this->db->get('estaciones')->result_array();
15
16         foreach ($estaciones as $estacion) {
17             // Verificar si hay un tiempo activo para esta estación
18             $this->db->where('id_estacion', $estacion['id']);
19             $this->db->where('hora_fin', NULL);
20             $tiempo_activo = $this->db->get('tiempos_estaciones')->row_array();
21
22             // Si hay un tiempo activo, indicarlo
23             $estacion['tiempo_activo'] = !empty($tiempo_activo);
24         }
25
26         $data['estaciones'] = $estaciones;
27         $this->load->view('welcome_message', $data);
28     }
29 }

```

Models

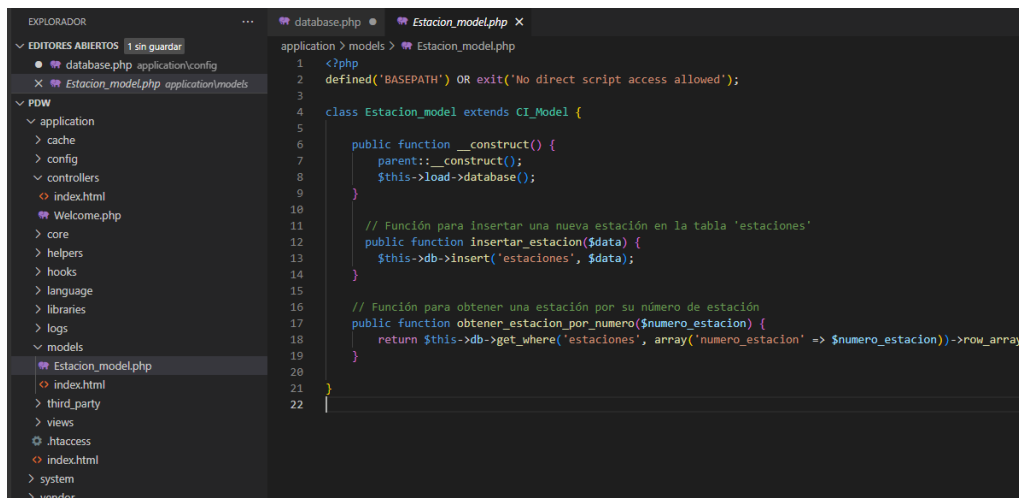
En CodeIgniter, los models son responsables de interactuar directamente con la base de datos y manejar la lógica de los datos. Los models contienen las funciones que permiten realizar operaciones como consultas, inserciones, actualizaciones o eliminaciones de registros en la base de datos.

En el archivo **Estacion_model.php**, tenemos un modelo en CodeIgniter que se utiliza para interactuar con la base de datos, específicamente con la tabla estaciones. A continuación se describe el código:

__construct(): Este es el constructor de la clase Estacion_model. Utiliza el método `parent::__construct()` para invocar el constructor de la clase padre CI_Model. También carga la base de datos utilizando `$this->load->database()`, lo que permite realizar operaciones de consulta SQL.

insertar_estacion(\$data): Esta función toma un arreglo de datos (`$data`) como parámetro e inserta un nuevo registro en la tabla estaciones. Utiliza la función `$this->db->insert()` de CodeIgniter para realizar la inserción.

obtener_estacion_por_numero(\$numero_estacion): Esta función busca y obtiene una estación específica de la tabla estaciones basada en el valor de `numero_estacion`. Utiliza el método `$this->db->get_where()` para consultar la base de datos y devolver el primer registro encontrado como un arreglo asociativo (`row_array()`).



views

En CodeIgniter, views (vistas) son archivos que contienen la presentación o la interfaz de usuario de una aplicación. Se encargan de mostrar datos al usuario y usualmente están formados por HTML, CSS y a veces JavaScript, junto con datos dinámicos enviados desde los controladores. Las vistas no contienen lógica de negocio ni interactúan directamente con la base de datos; en cambio, solo muestran la información procesada por los controladores.

El archivo **agregar_estacion.php** es una vista en CodeIgniter que presenta un formulario para que los usuarios puedan agregar una nueva estación en la base de datos. A continuación se detalla su estructura:

Descripción del código:

Encabezado HTML y metadatos:

Define el tipo de documento HTML5 (<!DOCTYPE html>).

Establece el conjunto de caracteres en UTF-8 para asegurar que todos los caracteres se rendericen correctamente.

La meta etiqueta viewport asegura que la página sea responsiva en dispositivos móviles.

El título de la página es "Agregar Estación".

Uso de Bootstrap 4:

La página utiliza Bootstrap 4 para el diseño y la estilización del formulario, lo cual le da una apariencia limpia y responsiva.

Estructura del cuerpo (<body>):

La clase container mt-5 se utiliza para crear un espacio centrado en la página con un margen superior para mejorar la apariencia del diseño.

Título del formulario: Un encabezado (<h2>) indica que el propósito del formulario es agregar una nueva estación.

Formulario:

El formulario usa el método POST y la acción está vinculada a la ruta welcome/guardar_estacion, que pertenece al controlador Welcome. Esta ruta se resuelve usando la función base_url() para asegurar que la URL sea correcta, incluso si el proyecto está en un subdirectorio.

Campos del formulario:

Número de Estación: Un campo de tipo number para ingresar el número de la estación. Este campo es obligatorio (required).

Nombre de Estación: Un campo de tipo text donde se puede ingresar el nombre de la estación. También es obligatorio.

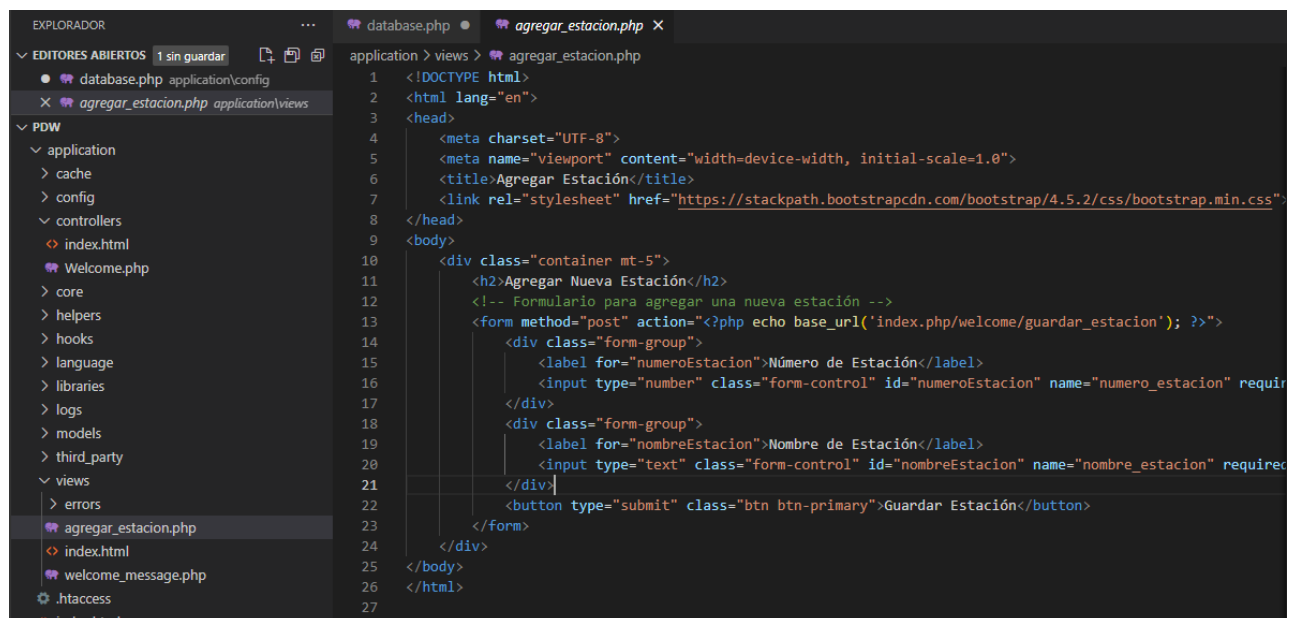
Botón de envío: Un botón que al ser presionado envía el formulario para guardar la estación.

Flujo de la aplicación:

El usuario accede a la vista agregar_estacion para agregar una nueva estación.

Ingresa el número y el nombre de la estación en los campos correspondientes.

Al hacer clic en el botón "Guardar Estación", los datos se envían al método guardar_estacion() del controlador Welcome, que luego inserta estos datos en la base de datos.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Agregar Estación</title>
7   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
8 </head>
9 <body>
10   <div class="container mt-5">
11     <h2>Agregar Nueva Estación</h2>
12     <!-- Formulario para agregar una nueva estación -->
13     <form method="post" action="{?php echo base_url('index.php/welcome/guardar_estacion'); ?}">
14       <div class="form-group">
15         <label for="numeroEstacion">Número de Estación</label>
16         <input type="number" class="form-control" id="numeroEstacion" name="numero_estacion" required="">
17       </div>
18       <div class="form-group">
19         <label for="nombreEstacion">Nombre de Estación</label>
20         <input type="text" class="form-control" id="nombreEstacion" name="nombre_estacion" required="">
21       </div>
22       <button type="submit" class="btn btn-primary">Guardar Estación</button>
23     </form>
24   </div>
25 </body>
26 </html>
27
```

welcome_message.php

crea una interfaz web para gestionar estaciones de monitoreo, que incluye la capacidad de iniciar y detener temporizadores, tanto de cuenta regresiva como de tiempo normal. A continuación, se describe cada sección y funcionalidad del código:

1. Estructura HTML y CSS

- **HTML Head:**
 - Se incluyen las referencias a Bootstrap 4.5.2 para el estilo y a una hoja de estilos personalizada (styles.css).
 - Se incluye jQuery 3.5.1 para facilitar la manipulación del DOM y la gestión de eventos.
- **HTML Body:**

Un botón de "Agregar Estación" que redirige a una página para añadir nuevas estaciones.

Un contenedor (`#estaciones-container`) que presenta una serie de tarjetas para cada estación registrada.

Cada tarjeta contiene:

Un encabezado con el número de la estación y un botón para eliminarla.

Un cuerpo con:

- El nombre de la estación.
- Un temporizador que muestra la cuenta regresiva o el tiempo transcurrido.
- Un campo de entrada para definir la duración de la cuenta regresiva.
- Botones para:
 - Iniciar Tiempo Definido: Inicia una cuenta regresiva con la duración especificada.
 - Iniciar Tiempo Normal: Inicia un cronómetro que cuenta el tiempo transcurrido.
 - Reiniciar: Detiene cualquier temporizador en curso y reinicia la cuenta.
 - Detener: Detiene cualquier temporizador en curso.

2. JavaScript para Control de Temporizadores

- **Variables Globales:**

`countdownIntervals` y `normalTimerIntervals` mantienen los intervalos activos para las cuentas regresivas y los cronómetros normales, respectivamente.

countdownEndTimes y **normalTimerEndTimes** almacenan los tiempos de fin o inicio para cada temporizador.

- **Funciones:**

loadTimersFromLocalStorage(): Carga el estado de los temporizadores desde `localStorage` para restaurar su estado al recargar la página.

saveTimersToLocalStorage(): Guarda el estado actual de los temporizadores en `localStorage`.

confirmarEliminacion(estacionId): Muestra una confirmación para eliminar una estación y redirige a la página de eliminación si se confirma.

startCountdown(estacionId): Inicia una cuenta regresiva con la duración especificada. Envía una solicitud AJAX para actualizar el servidor y guarda el estado localmente.

startNormalTimer(estacionId): Inicia un cronómetro normal. Envía una solicitud AJAX al servidor y guarda el estado localmente.

updateCountdown(estacionId): Actualiza el temporizador de cuenta regresiva y maneja la finalización.

updateNormalTimer(estacionId): Actualiza el cronómetro normal para mostrar el tiempo transcurrido.

pad(number): Formatea el tiempo para asegurarse de que siempre se muestren dos dígitos.

resetAndStartTimer(estacionId): Reinicia el temporizador y comienza un nuevo cronómetro normal después de detener cualquier temporizador en curso.

stopTimer(estacionId): Detiene cualquier temporizador en curso y actualiza el estado del servidor.

setAndStartTime(estacionId, minutos): Configura un temporizador de cuenta regresiva con una duración predefinida en minutos y lo inicia.

- **Inicialización:**

`window.onload:` Carga los temporizadores desde `localStorage` al iniciar la página.

3. Integración del Backend

El código incluye solicitudes AJAX para interactuar con un servidor backend en los siguientes puntos:

- Iniciar y detener cuentas regresivas y cronómetros normales.

- Reiniciar el temporizador y eliminar estaciones.

Estas solicitudes se envían a las siguientes rutas del servidor:

- iniciar_tiempo_regresivo
- iniciar_tiempo_normal
- detener_tiempo_regresivo
- detener_tiempo_normal
- reiniciar_tiempo
- eliminar

Cada solicitud maneja respuestas JSON para proporcionar mensajes de éxito o error, los cuales se muestran a través de alertas en el navegador.

```

EXPLORADOR
...
database.php welcome_message.php x agregar_estacion.php
EDITORES ABIERTOS 1 sin guardar
  database.php application/config
  welcome_message.php application/views
  agregar_estacion.php application/views
PDW
  application
    cache
    config
    controllers
      index.html
    Welcome.php
    core
    helpers
    hooks
    language
    libraries
    logs
    models
    third_party
    views
      errors
      agregar_estacion.php
      index.html
      welcome_message.php
      htaccess

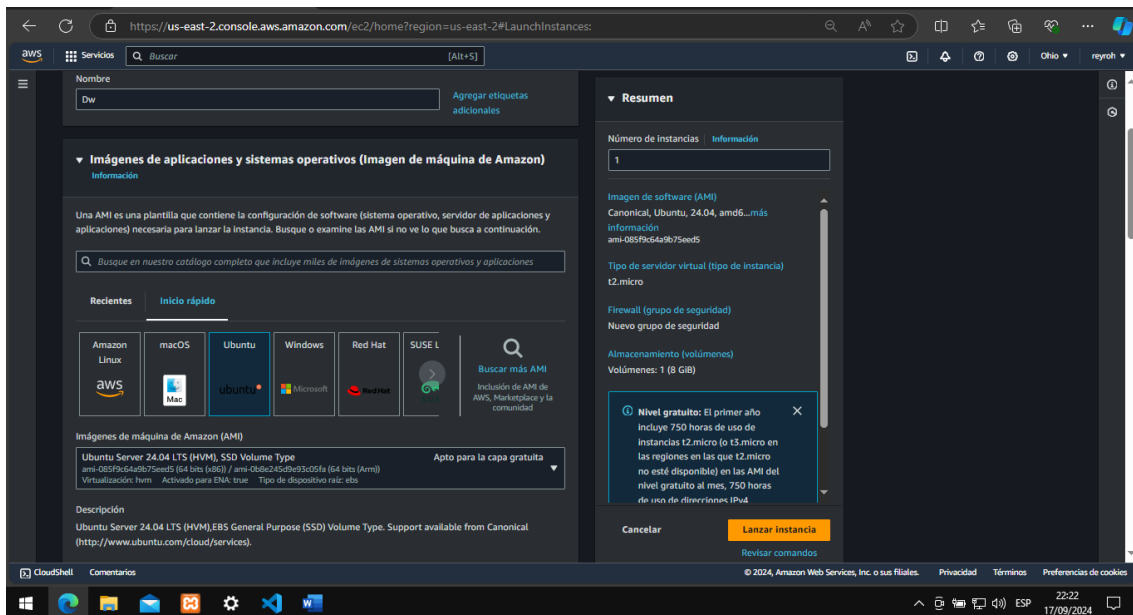
application > views > welcome_message.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Estaciones de Monitoreo</title>
7   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
8   <link rel="stylesheet" href="<?php echo base_url('vendor/css/styles.css?v=1.0'); ?>">
9   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
10 </head>
11
12 <body>
13   <div class="container mt-5">
14     <!-- Botón para agregar una nueva estación -->
15     <a href="<?php echo base_url('index.php/welcome/agregar'); ?>" class="btn btn-success mb-3">Agregar
16
17     <!-- Contenedor para las tarjetas de estaciones -->
18     <div class="row" id="estaciones-container">
19       <?php if (!empty($estaciones)): ?>
20         <?php foreach ($estaciones as $estacion): ?>
21           <div id="card-estacion-<?php echo $estacion['numero_estacion']; ?>" class="col-md-4 mb-3">
22             <div class="card">
23               <div class="card-header">
24                 Estación N° <?php echo $estacion['numero_estacion']; ?>
25               <!-- Botón para eliminar la estación con imagen -->
26               <button onclick="confirmarEliminacion(<?php echo $estacion['id']; ?>)" clas
27               <img src="<?php echo base_url('vendor/imgs/elim.png'); ?>" alt="Elimina

```

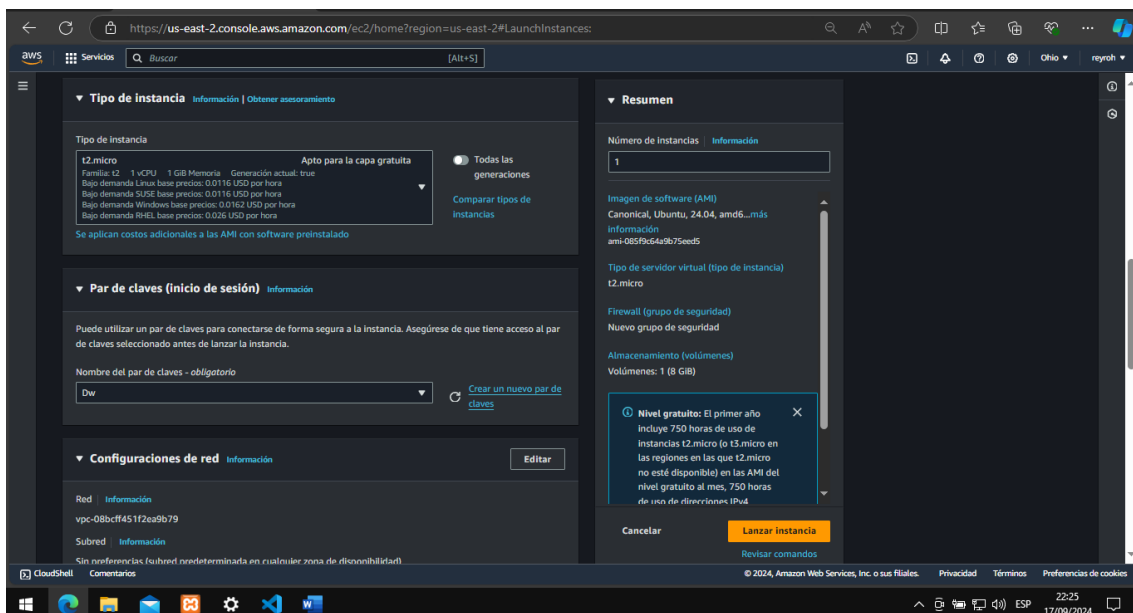
AWS

Al momento de registrar una cuenta en el servicio de AWS se debe de tener en cuenta la región de Ohio o el norte de california para aplicar a la capa t2.micro

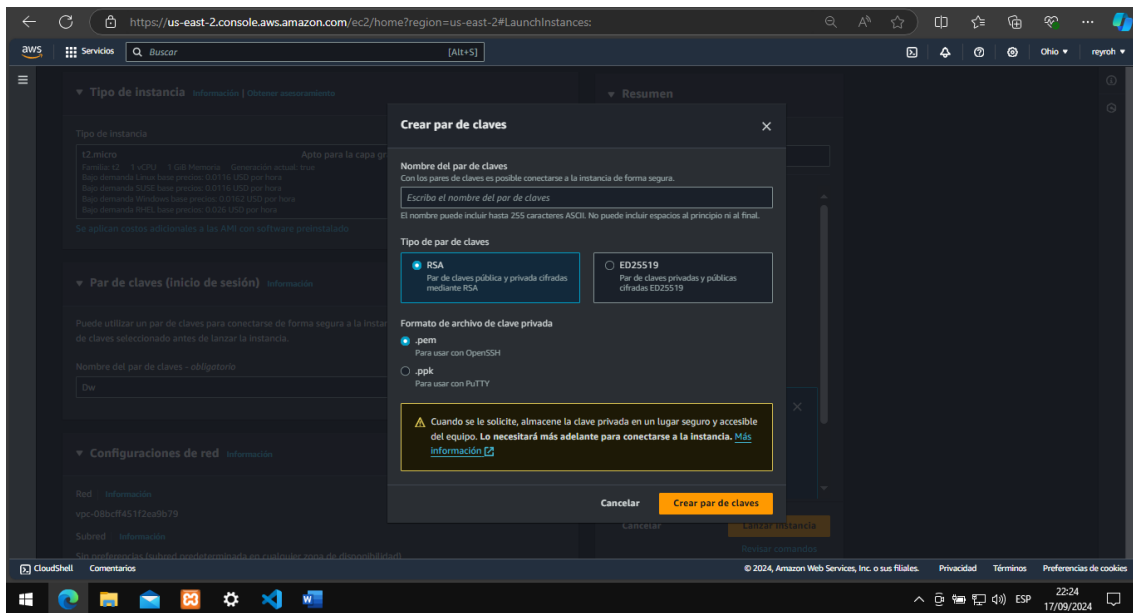
Configuración Inicial del servidor



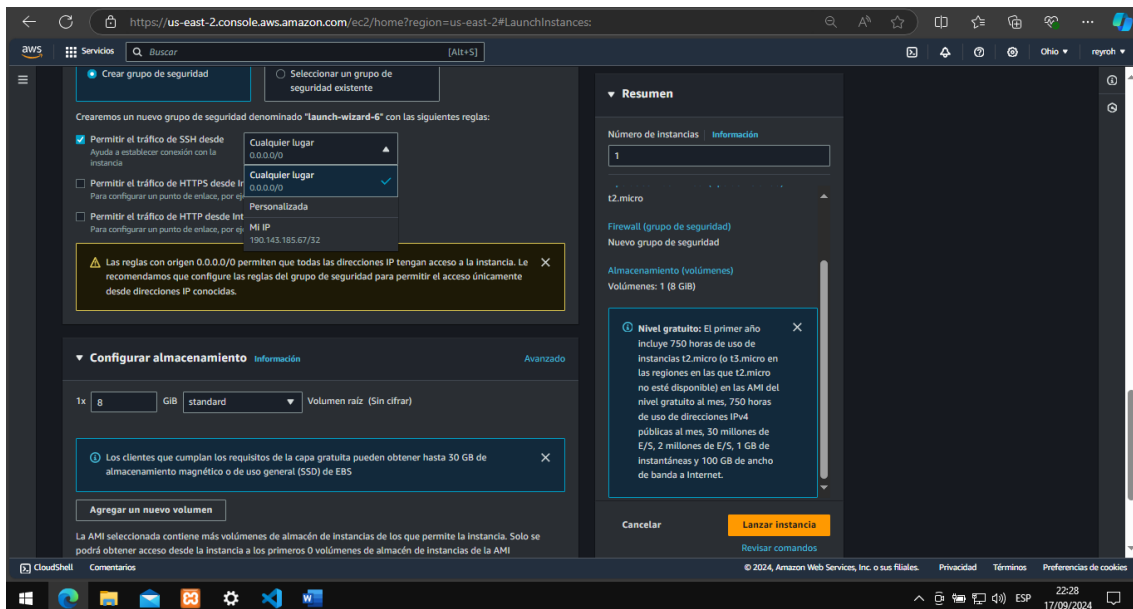
Damos un nombre a nuestro servidor en nuestro caso DW seleccionando el sistema operativo Ubuntu



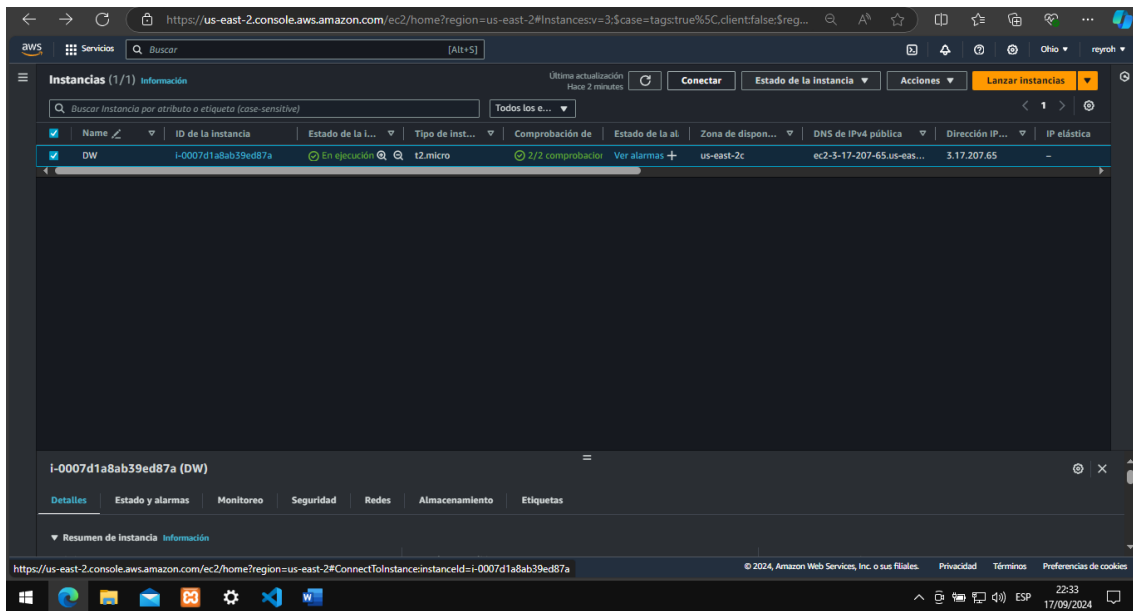
IMPORTANTE: se debe seleccionar la capa t2.micro para generar un cosoto de operaciones mínimos si llegara a facturar en algún momento



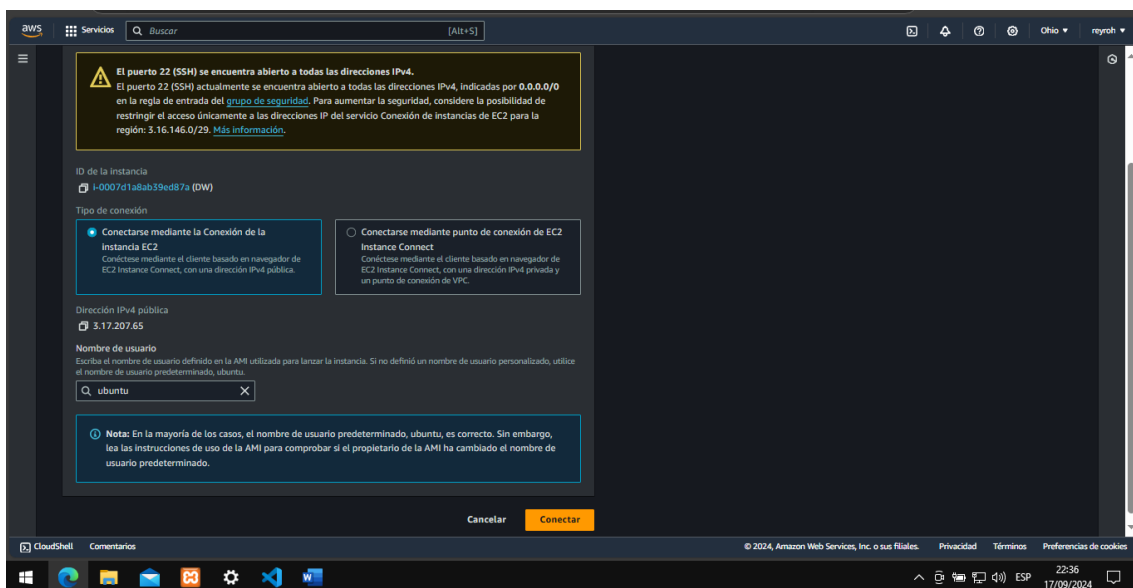
Generamos la llave acceso asignando un nombre y eligiendo la extensión “.pem”



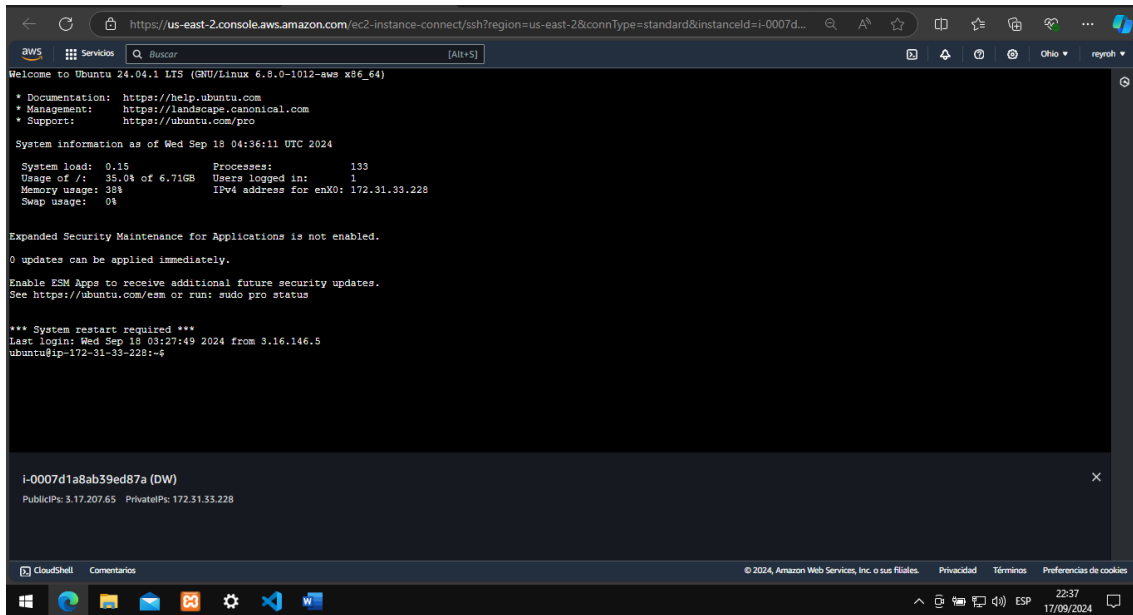
Configuramos el trafico de red para conexiones privadas o conexiones publicas y en tipo de volumen seleccionamos standard (para que el costo se mantenga en mínimo) y lanzamos nuestra instancia



Luego de comprobar que nuestra instancia esta en línea procedemos a darle al botón “conectar” para acceder a la consola de nuestro servidor, cabe aclarar que también se puede establecer una conexión “ssh” mediante una terminal Ubuntu o Linux, pero se debe preparar el archivo “DW.pem” antes de establecer la conexión



En este caso estableceremos la conexión mediante la consola de AWS



Esta será la vista de nuestro entorno de trabajo, procedemos a configurar el servidor con los siguientes comandos.

- `sudo apt update`
- `sudo apt upgrade -y`

Utilizamos estos comandos para actualizar los paquetes de nuestro servidor

- `sudo apt install apache2 -y`
- `sudo systemctl start apache2`
- `sudo systemctl enable apache2`

Utilizamos los siguientes comandos para instalar apache, luego de instaldo procedemos a iniciar el servicio y dejarlo configurado cada que se reinicie el sistema

- `sudo apt install software-properties-common`
- `sudo add-apt-repository ppa:ondrej/php`
- `sudo apt update`
- `sudo apt install php7.4 php7.4-cli php7.4-fpm php7.4-json php7.4-common php7.4-mysql php7.4-zip php7.4-gd php7.4-mbstring php7.4-curl php7.4-xml php7.4-bcmath -y`

Utilizamos los siguientes comandos para instalar apache PHP y sus dependencias necesarias para el proyecto

- `sudo apt install mariadb-server -y`
- `sudo systemctl start mariadb`
- `sudo systemctl enable mariadb`

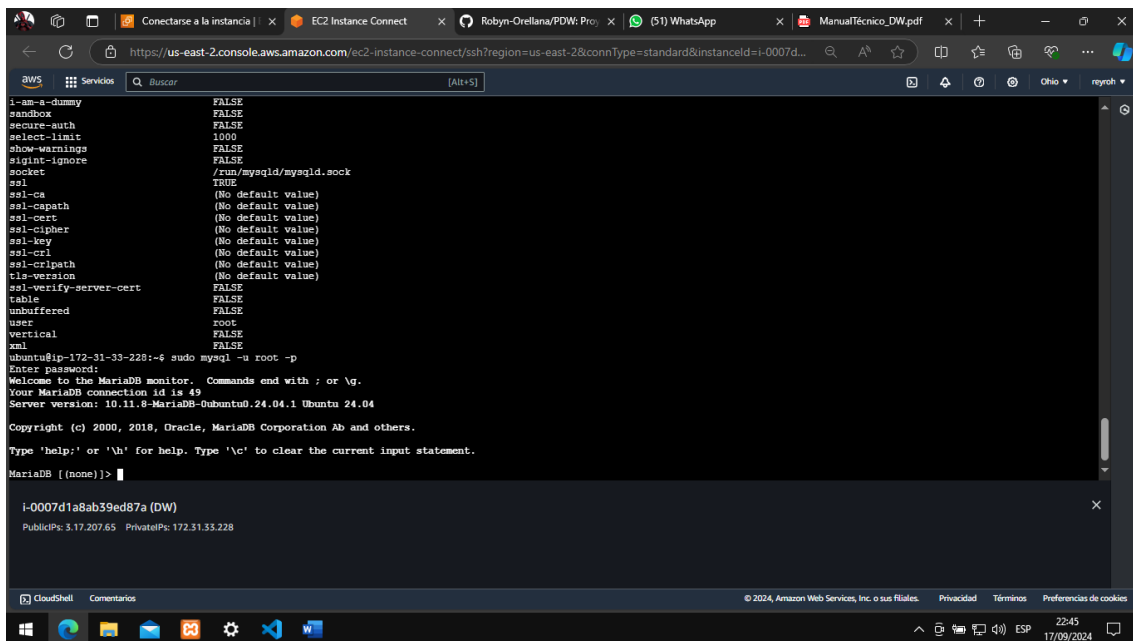
los siguientes comandos nos sirven para instalar nuestro gestor de base de datos que en este caso será "MariaDB"

- `sudo mysql_secure_installation`

El siguiente comando sirve para configurar la clase de nuestra instancia de base de datos, la cual será en nuestro caso “Test123.”

- `sudo mysql -u root -p`

El siguiente comando nos sirve para acceder a nuestro gestor de base de datos MariaDB para poder crear nuestra base de datos



```
i-am-a-dummy FALSE
sandbox FALSE
secure-auth FALSE
select-limit 1000
show-warnings FALSE
sigint-ignore FALSE
socket /run/mysqld/mysqld.sock
ssl TRUE
ssl-ca (No default value)
ssl-capath (No default value)
ssl-cert (No default value)
ssl-cipher (No default value)
ssl-key (No default value)
ssl-crl (No default value)
ssl-crlpath (No default value)
tls-version (No default value)
ssl-verify-server-cert FALSE
table FALSE
unbuffered FALSE
user root
vertical FALSE
xsl FALSE

ubuntu@ip-172-31-33-228:~$ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \q.
Your MariaDB connection id is 49
Server version: 10.11.8-MariaDB-0ubuntu0.24.04.1 Ubuntu 24.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]>

i-0007d1a8ab39ed87a (DW)
PublicIPs: 3.17.207.65 PrivateIPs: 172.31.33.228
```

Utilizamos el siguiente Script:

```
CREATE DATABASE cafeinternet;
```

```
USE cafeinternet;
```

```
CREATE TABLE estaciones (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY, -- Identificador único de cada equipo
```

```
    numero_estacion INT NOT NULL, -- Número de la estación
```

```
    nombre_estacion VARCHAR(255) NOT NULL, -- Nombre de la estación o equipo
```

```
    activa BOOLEAN DEFAULT TRUE -- Indica si la estación está activa
```

```
);
```

```
CREATE TABLE tiempos_estaciones (
```

```

id INT AUTO_INCREMENT PRIMARY KEY, -- Identificador único para cada registro de tiempo
id_estacion INT NOT NULL, -- Relaciona con la tabla 'estaciones'
hora_inicio DATETIME NOT NULL, -- Tiempo de inicio
hora_fin DATETIME, -- Tiempo de fin (NULL si sigue activa)

FOREIGN KEY (id_estacion) REFERENCES estaciones(id) -- Relación con la tabla 'estaciones');

```

```

ALTER TABLE tiempos_estaciones ADD COLUMN duracion INT DEFAULT 0;

```

```

GRANT ALL PRIVILEGES ON cafeinternet.* TO 'ubuntu'@'localhost' IDENTIFIED BY 'Test123.';
GRANT ALL PRIVILEGES ON cafeinternet.* TO 'root'@'localhost' IDENTIFIED BY 'Test123.';
FLUSH PRIVILEGES;
EXIT;

```

Una vez configurada nuestra base de datos podreemos a desplegarlos a la siguiente ruta

- `cd /var/www/html`

en la cual ejecutaremos el siguiente script

- `sudo mkdir P`
- `sudo git clone https://github.com/Robyn-Orellana/PDW P`
- `sudo chown -R www-data:www-data /var/www/html/P`
- `sudo chmod -R 755 /var/www/html/P`

el siguiente script crea una capeta con nombre “P”, luego de ello se clona el repositorio de GitHub donde esta cargado el proyecto y se le dan permisos a la carpeta para poder ser utilizada por apache

- `sudo nano /etc/apache2/sites-available/P.conf`

con el siguiente comando creamos un archivo de configuración para apache e ingresamos el siguiente script

```

<VirtualHost *:80>

```

```

    ServerAdmin webmaster@localhost

```

```

    DocumentRoot /var/www/html/P

```

```

    ServerName ip-publica

```

```
<Directory /var/www/html/P>
```

```
Options Indexes FollowSymLinks
```

```
AllowOverride All
```

```
Require all granted
```

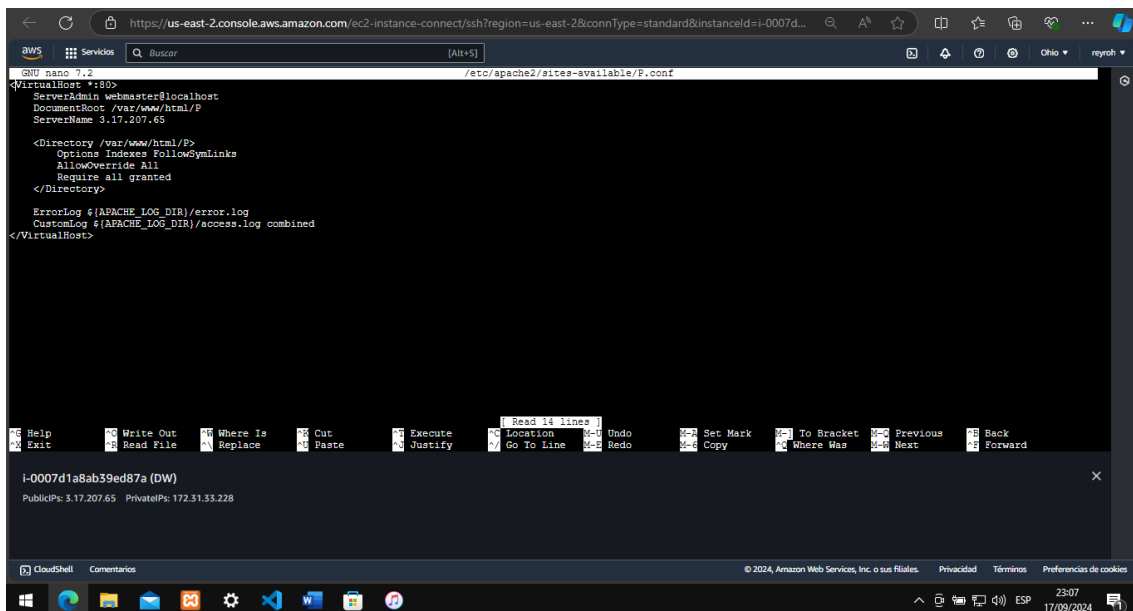
```
</Directory>
```

```
ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
</VirtualHost>
```

El cual se vera de la siguiente manera en nuestra terminal



```
GNU nano 2.2 /etc/apache2/sites-available/P.conf
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/html/P
  ServerName 3.17.207.65

  <Directory /var/www/html/P>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
  </Directory>

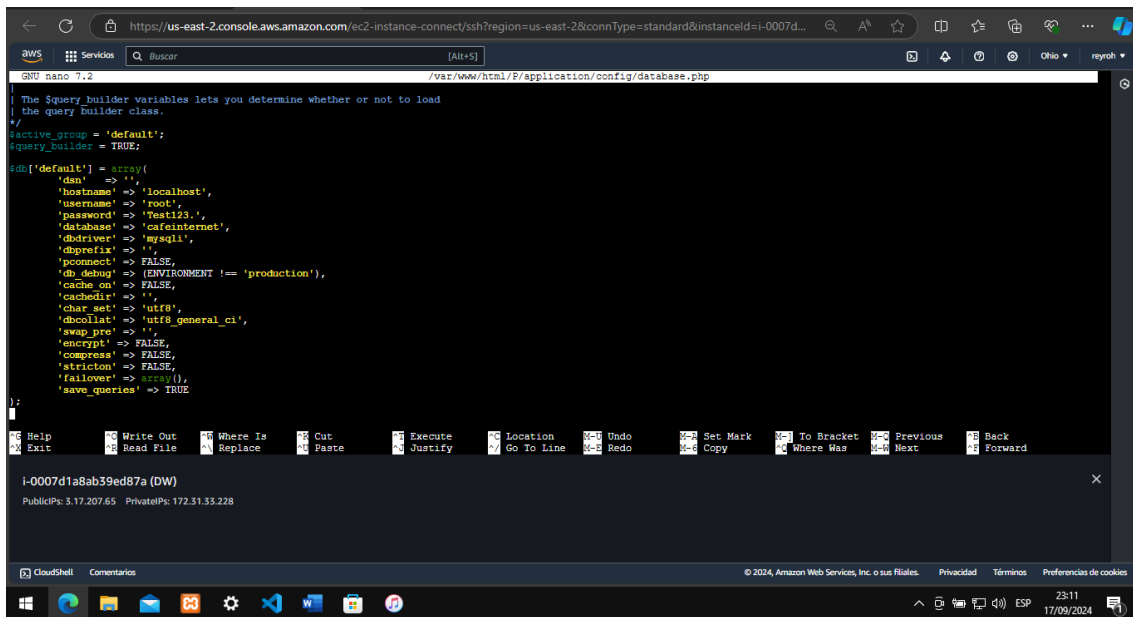
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Luego habilitamos el archivo que acabamos de configurar con los siguientes comandos

- `sudo a2ensite P.conf`
- `sudo a2enmod rewrite`

utilizamos el siguiente script para agregar nuestras credenciales de base de datos

- `sudo nano /var/www/html/P/application/config/database.php`

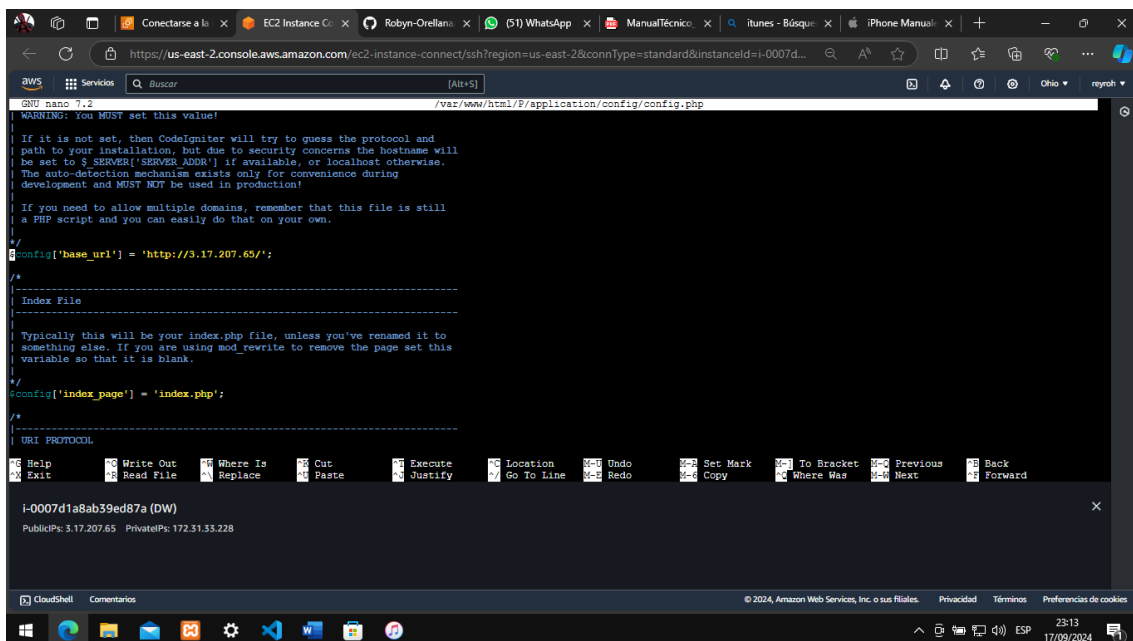


```
GNU nano 2.2 /var/www/html/P/application/config/database.php
/*
 * The $query_builder variable lets you determine whether or not to load
 * the query builder class.
 */
$query_builder = TRUE;

$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => 'Test123.',
    'database' => 'cafeinternet',
    'dbdriver' => 'mysql',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

Guardamos la configuración y procedemos a utilizar el siguiente comando

- `sudo nano /var/www/html/P/application/config/config.php`



```
GNU nano 2.2 /var/www/html/P/application/config/config.php
WARNING: You MUST set this value!
If it is not set, then CodeIgniter will try to guess the protocol and
path to your installation, but due to security concerns the hostname will
be set to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.
The auto-detection mechanism exists only for convenience during
development and MUST NOT be used in production!

If you need to allow multiple domains, remember that this file is still
a PHP script and you can easily do that on your own.

$config['base_url'] = 'http://3.17.207.65/';

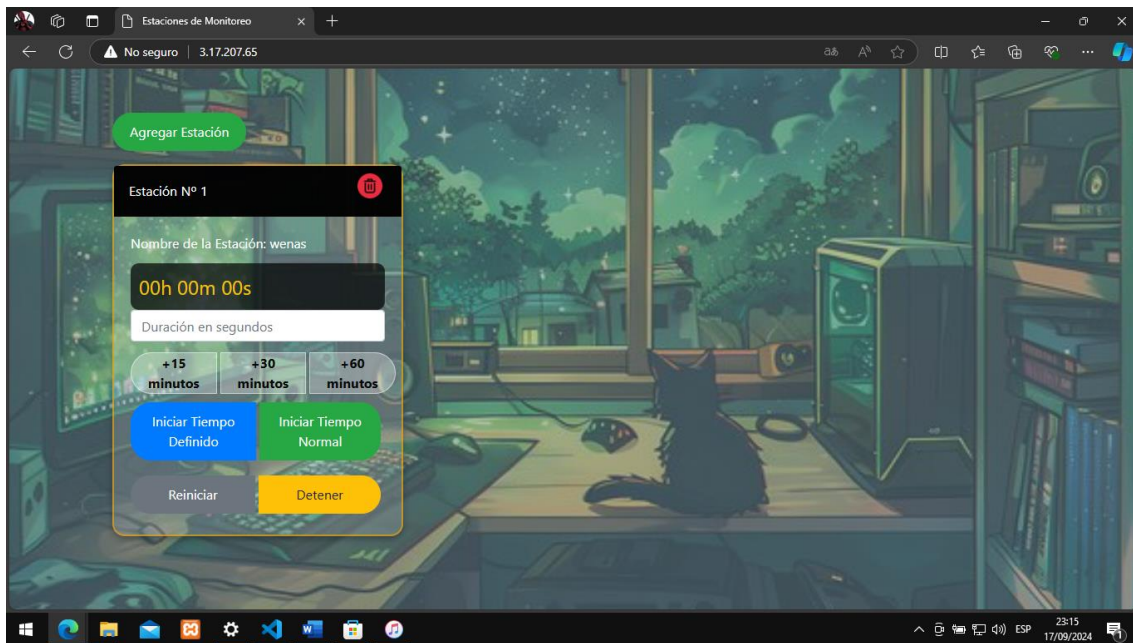
/*
 * Index File
 */
Typically this will be your index.php file, unless you've renamed it to
something else. If you are using mod_rewrite to remove the page set this
variable so that it is blank.

$config['index_page'] = 'index.php';

/*
 * URI PROTOCOL
 */
```

En la línea con el texto “\$config['base_url'] = ” debemos asignar la ip publica de nuestro servidor de AWS y guardamos

Luego de realizar los pasos nuestro servidor y pagina deberían quedar configurados y listos para su uso



En este caso nuestra dirección ip publica seria la **"3.17.207.65"** donde esta alojada la pagina del repositorio mencionado en el documento

Conclusión

El desarrollo del sistema de gestión de tiempo para estaciones ha sido cuidadosamente diseñado para proporcionar una solución eficiente y flexible en entornos donde es fundamental controlar el uso del tiempo de cada estación. Gracias a la arquitectura cliente-servidor, la implementación de tecnologías web estándar, y un backend robusto basado en CodeIgniter y MySQL, el sistema asegura un rendimiento óptimo y una fácil escalabilidad. La interfaz gráfica y el manejo intuitivo permiten tanto a los usuarios como a los administradores interactuar de manera fluida, asegurando un control preciso y eficiente del tiempo. En conjunto, este sistema proporciona una herramienta eficaz para mejorar la experiencia de usuario en entornos compartidos, facilitando la gestión de recursos y optimizando el rendimiento del negocio.