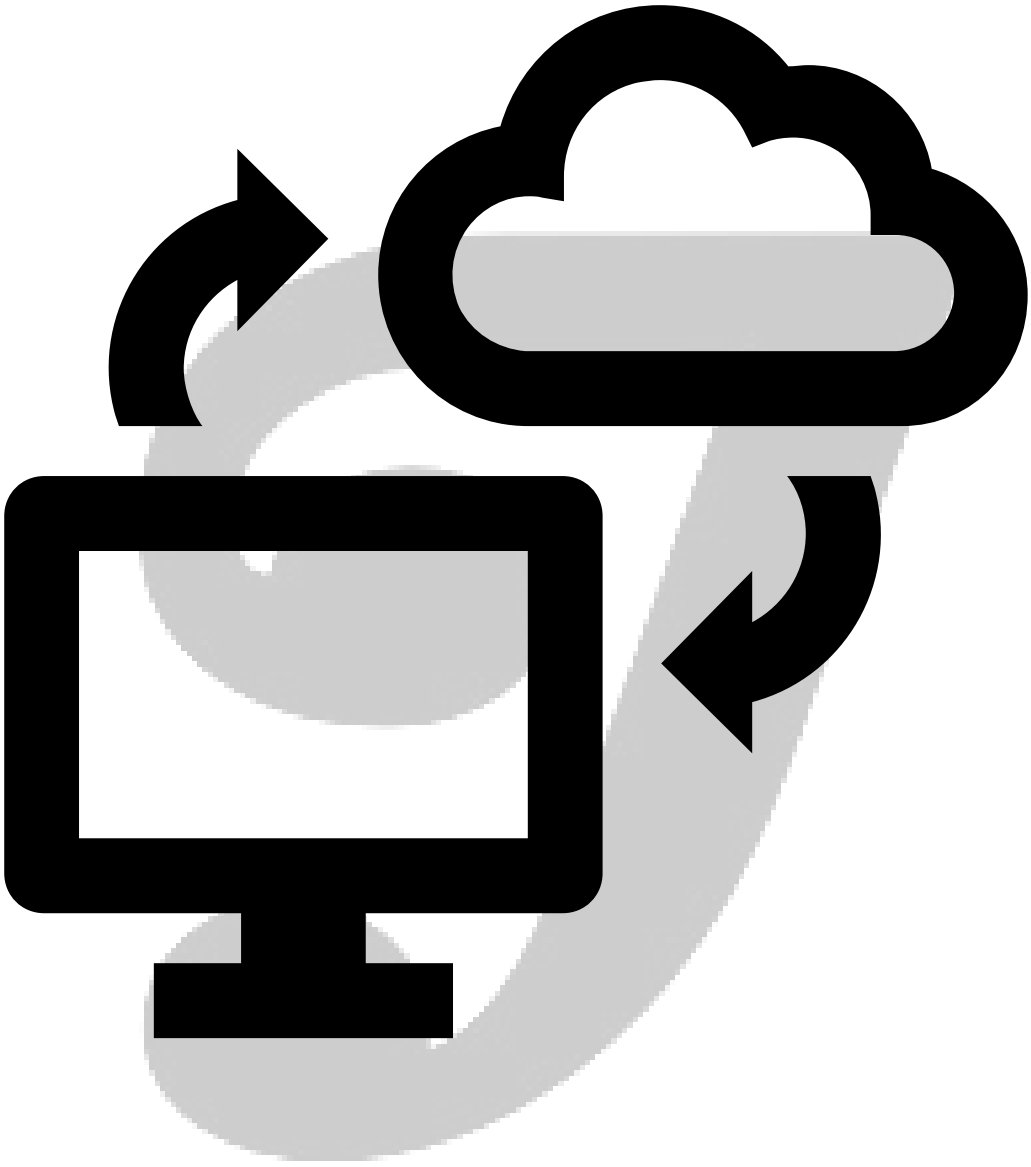


BLOG PERSONAL

Programación Cliente-Servidor

ESTUDIANTE FRANCISCO JAVIER ROSALES SANCHEZ
Francisco_1320114043@uptecamac.edu.mx

Índice

Comunicación de Dispositivos de Red y las Arquitecturas

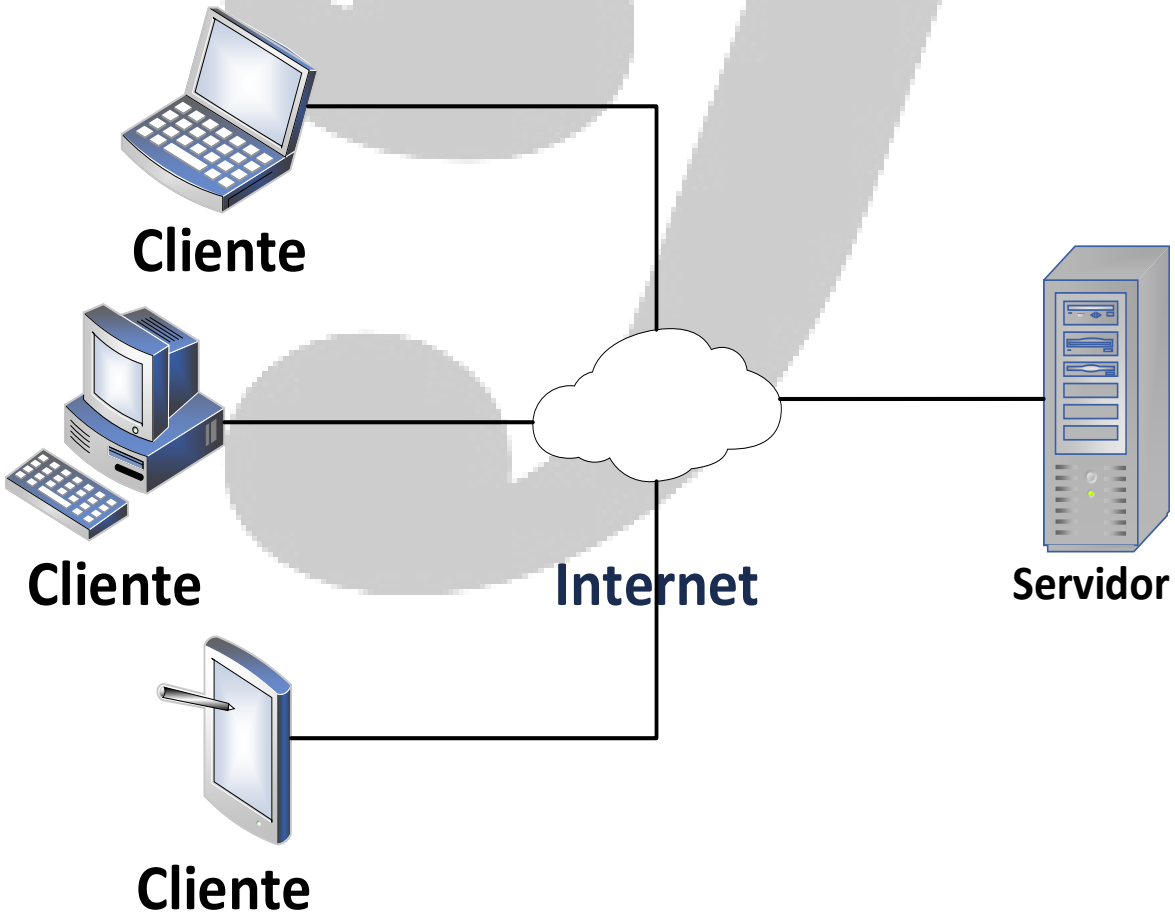
La arquitectura de red es el diseño de una red de comunicaciones. Es un marco para la especificación de los componentes físicos de una red y de su organización funcional y configuración, sus procedimientos y principios operacionales, así como los protocolos de comunicación utilizados en su funcionamiento.

En la telecomunicación, la especificación de un diseño de red puede incluir también una descripción detallada de los productos y servicios entregados a través de una red de comunicaciones, así como la tasa de facturación detallada y estructuras en las que se compensan los servicios.

El diseño de red de Internet se expresa de forma predominante por el uso de la familia de protocolos de Internet, en lugar de un modelo específico para la interconexión de redes o nodos en la red, o el uso de tipos específicos de enlaces de hardware.

En la computación distribuida, el concepto de arquitectura de red a menudo describe la estructura y la clasificación de una arquitectura de aplicaciones distribuidas, ya que los nodos que participan en una aplicación distribuida se conocen como una red. Por ejemplo, la arquitectura de aplicaciones de la red telefónica conmutada (RTC) se ha denominado la red inteligente avanzada.

Diagrama de componentes de la Arquitectura Cliente-Servidor



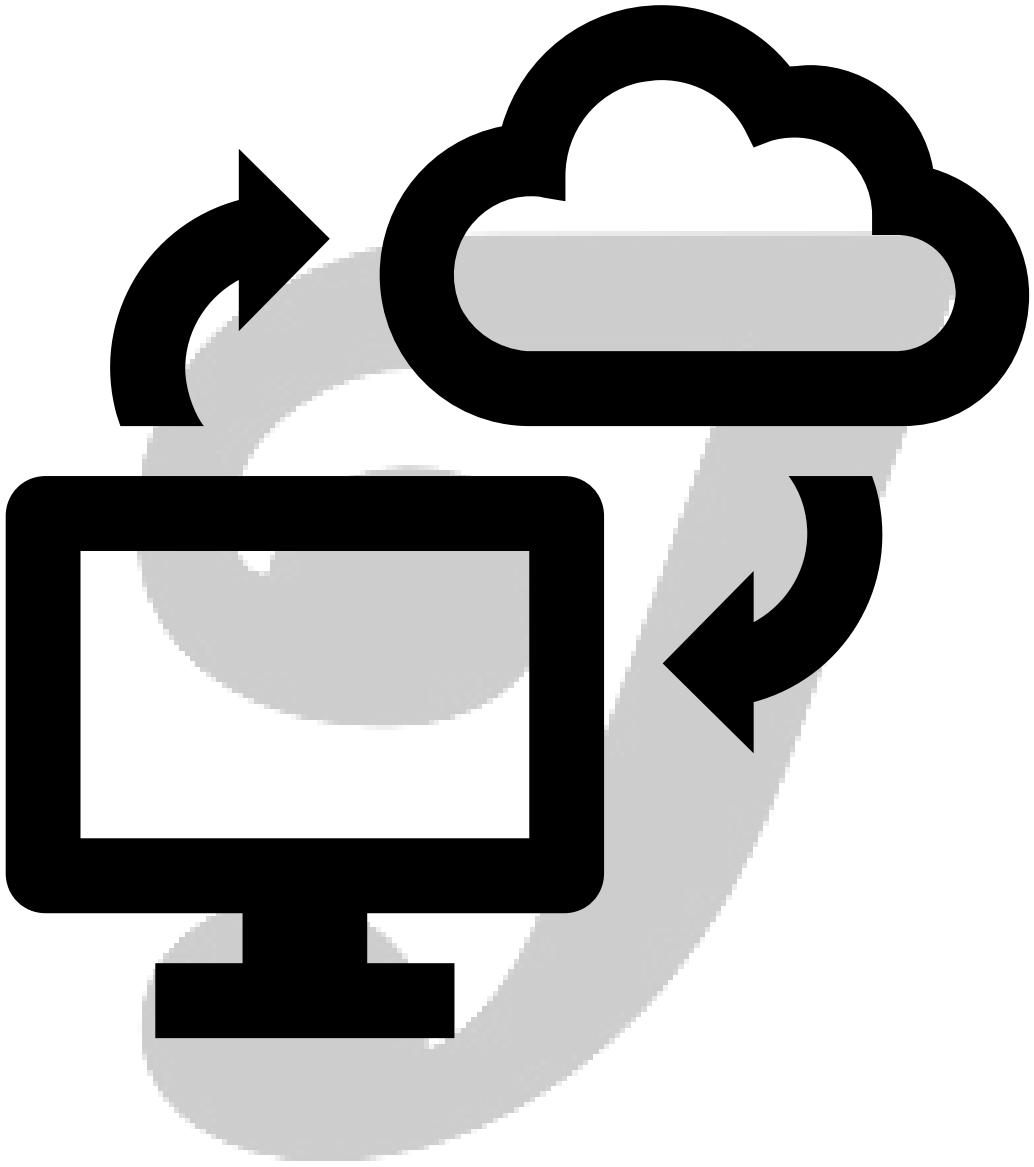
Cuadro Comparativo

IAAS	PAAS	SAAS	CLIENTE-SERVIDOR
Infraestructura como Servicio	Plataforma como Servicio	Software como servicio	Arquitectura
La mitad de todo es gestionado por el provedor	Runtine, Middleware, Sistema Operativo, Vitualización, Servidores, Almacenamiento, Networking son gestionados por el proveedor	Todo esta gestionado por el proovedor	Requieren de un servidor, si el servvidor cae las peticiones de los clientes no pueden ser satisfechas
Aplicaciónes, Datos, Runtine, Middleware y Sistema Operativo son gestionados por el cliente	Aplicaciónes y Datos son gestionados por el cliente	No hay nada gestionado por el cliente	La arquitectura cliente-servidor no es robusta, la arquitectura no es escalable

Propuesta técnica “Cliente-Servidor”

Levantar un blog web como propuesta considerando gustos personales “hobbies” para hacer esto se requiere de un lenguaje de programación, los requerimientos comunes de un servidor, así como un dominio.

Con respecto al lenguaje se usa laravel, contara con controladores, vistas, y un sistema de envió de mensajes al propietario para comentarios, etc.

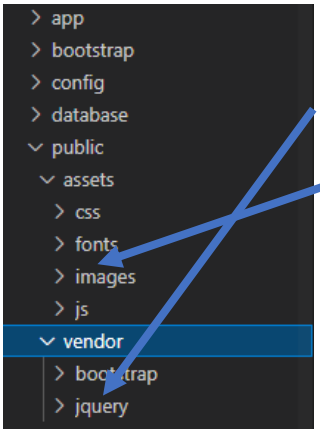


MANUAL CLIENTE-SERVIDOR

Programación Cliente-Servidor

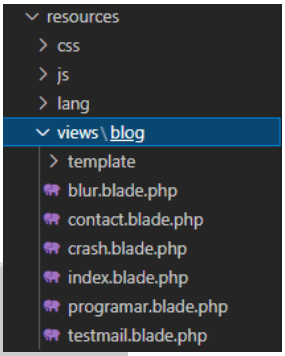
ESTUDIANTE FRANCISCO JAVIER ROSALES SANCHEZ
Francisco_1320114043@uptecamac.edu.mx

Distribución de Pantallas



Los principales archivos se encuentran en hobbies/public/ en las carpetas assets y vendor, las imágenes, carpetas de Bootstrap y jQuery.

Las vistas o pantallas se encuentran en resources/views/blog, hay un template con un layout que sirve de fondo para los detalles de las pantallas de blur, crash y programación.



Distribución de Controladores

Se creo un solo controlador que se encarga de obtener nombre, correo y el mensaje escrito para mandarla al correo escrito en la clase ***"MailController"***.

```php
class MailController extends Controller
{
    public function getMail(Request $datos){
        $Nombre = $datos -> name;
        $Mensaje = $datos -> message;
        $Correo = $datos -> subject;
        $data = ['name' => $Nombre, 'sms' => $Mensaje, 'email' => $Correo];
        Mail::to('raziel.rosfo.1904@gmail.com')->send(new TestMail($data));
        return redirect('contacto');
    }
}
```

TestMail

```php
public $data;
public $Nombre;
public $Correo;
public $Mensaje;

public function __construct($data)
{
    $this->data = $data;

}

/**
 * Build the message.
 *
 * @return $this
 */
public function build()
{
    return $this->from(env('frank.pruebas.0402@gmail.com'), env('MAIL_FROM_NAME'))
                ->view('blog/testmail')
                ->subject('Frank-Hobbies')
                ->with($this->data);
```

El **TestMail** utilizado para definir las variables y crear una ***public function*** en la que indica el formato del mensaje y el correo de procedencia de este, el correo se especifica para evitar problemas con los puntos del correo, todo se retorna junto con la variable "data" a través de la función "build".

Template “Details”

El layout de detalles sirve de fondo para base de las views de Crash, Blur y Programación. Se indica su espacio de modificación con un **@yield(‘Barra’)** siendo *“Barra”* un nombre de identificador.

```
57                  <li class="nav-item active">
58                    @yield('Barra')
59                  </li>
```

Para otras modificaciones se nombran de una manera distinta y se colocan en el espacio en el que se modificara, para cambiar la imagen de fondo, el titulo y comentarios agregados.

```
95                      @yield('Imagen')
96                    </div>
97                    <div class="down-content">
98                      @yield('Titulo')
99                    </div>
100                 </div>
101               </div>
102               <div class="col-lg-12">
103                 <div class="sidebar-item comments">
104                   @yield('Coment')
```

Web.php

En la sección de las rutas el archivo **web.php** es el que redirige a todo empezando por el índex principal y las rutas del resto de views utilizadas como la de los detalles de páginas o la de contacto, así como la de envió del correo electrónico mandando a llamar la clase ***“MailController”*** para la que se importa el controlador ***“MailController”*** con el comando **“use”** y la ruta del Controlador, así como el nombre de la función a la que debe de entrar en este caso ***“getMail”***.

```
1   <?php
2
3   use Illuminate\Support\Facades\Route;
4   use App\Http\Controllers\MailController;
5   /*
6   |--------------------------------------------------------------------------
7   | Web Routes
8   |--------------------------------------------------------------------------
9   |
10  | Here is where you can register web routes for your application. These
11  | routes are loaded by the RouteServiceProvider within a group which
12  | contains the "web" middleware group. Now create something great!
13  |
14  */
15  Route::post('/mail', [MailController::class, 'getMail']);
16
17  Route::get('/', function () {
18      return view('blog.index');
19  })->name('inicio');
20
21  Route::get('/contacto', function () {
22      return view('blog.contact');
23  })->name('contacto');
24
25  Route::get('/programar', function () {
26      return view('blog.programar');
27  })->name('programar');
28
29  Route::get('/crash', function () {
30      return view('blog.crash');
31  })->name('crash');
32
33  Route::get('/blur', function () {
34      return view('blog.blur');
35  })->name('blur');
```

Index (Principal)

En la pantalla principal o ***“Índex”*** es más que diseño en el que cambia el uso de rutas con **web.php** ya que se nombran de diferente manera al ser llamadas como, por ejemplo:

```
51              <div class="collapse navbar-collapse" id="navbarResponsive">
52                <ul class="navbar-nav ml-auto">
53                  <li class="nav-item active">
54                    <a class="nav-link" href="/">Inicio
55                      <span class="sr-only">(current)</span>
56                    </a>
57                  </li>
58                  <li class="nav-item">
59                    <a class="nav-link" href="contacto">Contacto</a>
```

Crash, Blur, Programación

```
@extends('blog.template.details')
@section('Barra')
<a class="nav-link" href="crash">Crash
    <span class="sr-only">(current)</span>
</a>
@endsection
@section('Imagen')
<img src="assets/images/Image2.jpg" alt="">
@endsection
@section('Titulo')
<span>Crash Team Racing</span>
<a href="crash"><h4>Un clasico de los Video
<p>Crash Team Racing es la cuarta entrega de
<br><br>Durante las carreras, los potenciado
@endsection
@section('Coment')
```

Las pantallas son similares a nivel estructura lo único que cambia es el contenido lo que son nombres y descripción a nivel de la Interfaz por lo que gracias a eso se extiende desde un "***template***" nombrado "***details***" para facilitar esta edición en las 3 views.

```
@extends('blog.template.details')
@section('Barra')
<a class="nav-link" href="blur">Blur
    <span class="sr-only">(current)</span>
</a>
@endsection
@section('Imagen')
<img src="assets/images/Image3.jpg" alt="">
@endsection
@section('Titulo')
<span>Blur</span>
<a href="blur"><h4>Pasatiempo o Misión</h4>
<p>Blur es un videojuego arcade desarrollado
<br><br>En el modo de un jugador, el mismo
@endsection
@section('Coment')
```

```
@extends('blog.template.details')
@section('Barra')
<a class="nav-link" href="programar">Programar
    <span class="sr-only">(current)</span>
</a>
@endsection
@section('Imagen')
<img src="assets/images/Image1.jpg" alt="">
@endsection
@section('Titulo')
<span>Programar</span>
<a href="programar"><h4>Programar es amar y llo
<p>La programación es el proceso utilizado para
@endsection
@section('Coment')
```

El templete de "details" muestra las secciones que se nombraron para agregar las cuales son ***"Barra", "Imagen", "Coment" y "Titulo"*** con las cuales so nombre asignar una personalización diferente en cada vista.

Contacto

```
<form action="/mail" method="post">
  @csrf
  <div class="row">
    <div class="col-md-6 col-sm-12">
      <fieldset>
        <input name="name" type="text" id="name" placeholder="Nombre">
      </fieldset>
    </div>
    <div class="col-md-6 col-sm-12">
      <fieldset>
        <input name="subject" type="text" id="subject" placeholder="Correo">
      </fieldset>
    </div>
    <div class="col-lg-12">
      <fieldset>
        <textarea name="message" rows="6" id="message" placeholder="Mensaje"></textarea>
      </fieldset>
    </div>
    <div class="col-lg-12">
      <fieldset>
        <button type="submit" id="form-submit" class="main-button">Enviar</button>
      </fieldset>
    </div>
  </div>
</form>
```

La vista contacto la función que tiene es dar información personal como un correo, más aparte un formulario de envió de correo para poder contactar con el dueño del sitio y/o dar un comentario sobre la página. El formulario obtiene los datos nombre, correo, el mensaje a enviar y el botón enviar manda a llamar a llamar la ruta ***"/mail"*** vista en ***"Web.php"*** la cual manda a llamar al controlador ***"MailController"*** y finalmente enviara a la vista ***"TestMail".***

Vista (TestMail)

El mensaje final o lo que dirá el correo se crea en una vista nombrada TestMail en el que se mostrará el contenido del mensaje de acuerdo a las variables obtenidas en el controlador ***"MailController"*** y finalmente se enviará el mensaje al correo asignado el archivo de variables ***"env".***

Archivo (env)

El archivo fue modificado y agregado las variables de uso para ser subido en donde el ***"APP_URL"*** se modifica al subir y se agrega el url en el que estará, así como el tipo de Mailer y el Host que se modificó e igualmente se subieron las variables usadas.

```
APP_NAME=Frank-Hobbies
APP_ENV=local
APP_KEY=base64:4jTE89ZoRd5hDbz/egejG0fW7yQR47AjI6nJZ4dyJHg=
APP_DEBUG=true
APP_URL=http://localhost
```

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=frank.pruebas.0402@gmail.com
MAIL_PASSWORD=jigowzmxfdzkuonc
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=frank.pruebas.0402@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```