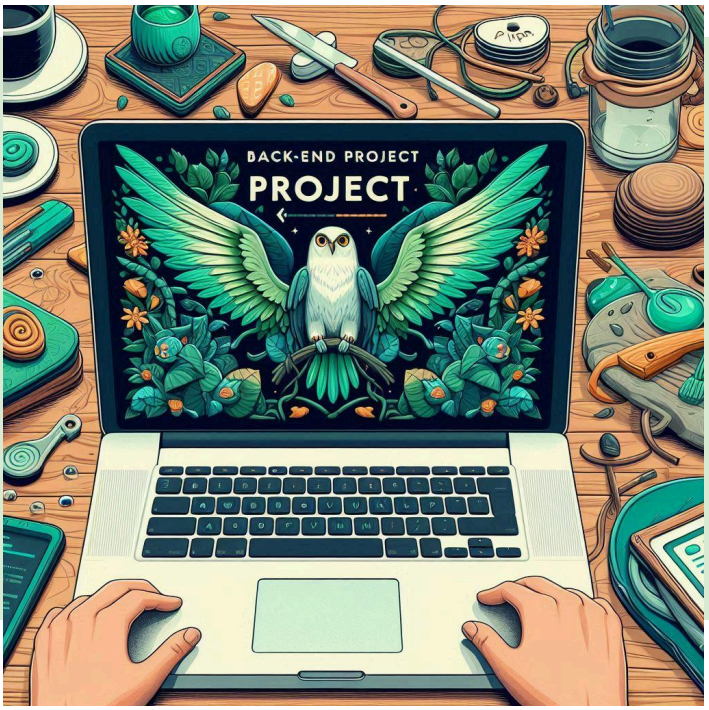


Producto 3. Desarrollo de una aplicación gestión de transfers con Laravel

Desarrollo back-end con PHP, framework MVC y gestor de contenidos

FP.064

Operatix 2.0

Omaima Khoyani Jabri
Joaquín Peraire Monfort
Genís Pascual Sarreta
Marc Gay Lachner

Índice

1. Instalar en el hosting AWS proporcionado por el consultor si no se ha hecho ya en el Producto 1, el entorno Laravel.

2. Importar la base de datos proporcionada utilizada en el producto 2.

    A. Se utilizará la base de datos completa que hemos utilizado en el producto 2, junto con los registros creados.

    B. Se podrá modificar la base de datos añadiendo campos o tablas nuevas solamente si se quiere añadir alguna función adicional al proyecto, no para facilitar las funciones básicas.

3. Desarrollar el producto 2 al Framework Laravel mejorando el aspecto gráfico.

4. Los pasos a seguir para mejorar el producto son:

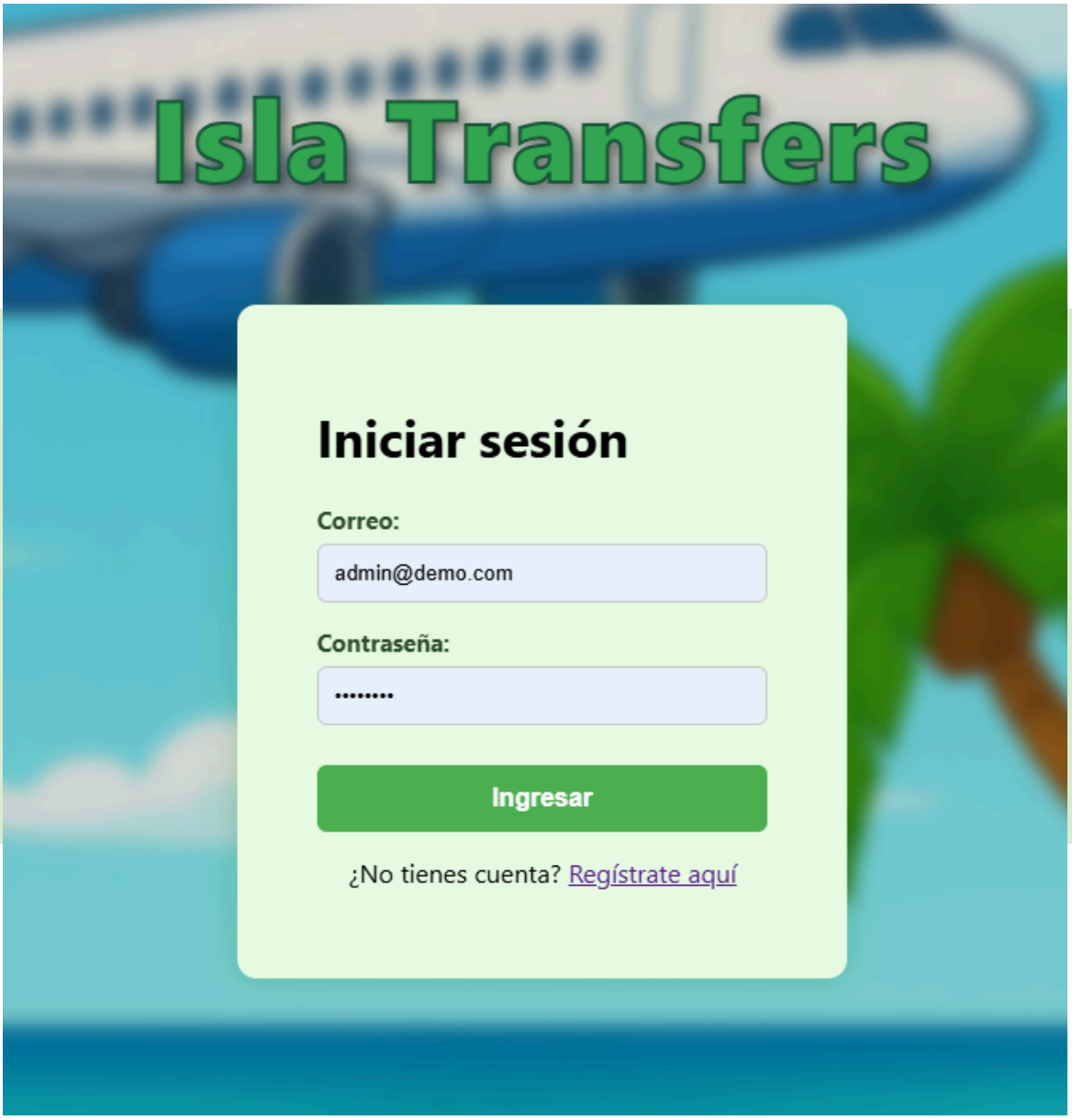
    A. Los usuarios corporativos los dará de alta el administrador y tendrán un usuario propio con acceso a un panel. En ese panel podrán realizar reservas con destino/origen (o ambos) el hotel. En este caso el hotel recibirá una comisión por el servicio que puede ser variable según lo pactado entre el hotel y la empresa de transfers. En el panel también podrán ver las comisiones que recibirán mes a mes.

    B. El administrador ha de poder ver un listado de las reservas realizadas por cada hotel y calcular la comisión total a pagar cada mes.

5. Se creará un “REST WebService” o procedimiento que devuelva un JSON con información agregada de las reservas realizadas a distintas zonas de la isla. En este sentido cada hotel está asignado a una zona. El resultado debe ser un listado de zona, número de traslados y % del total. El objetivo es mostrar en el producto 5 un listado de las reservas realizadas por zonas.

6. La aplicación debe responder a dominio.com/producto3 ya que en el producto 5 instalaremos en la raíz del dominio la web de la empresa.

7.Enlaces


Isla Transfers
Iniciar sesión
Correo:
admin@demo.com
Contraseña:
Ingresar
¿No tienes cuenta? Regístrate aquí

Resolución del producto:

1. Instalar en el hosting AWS proporcionado por el consultor si no se ha hecho ya en el Producto 1, el entorno Laravel.

- Paso 1: Para instalar el hosting de Amazon Web Services proporcionado por el consultor en el producto 1, lo que vamos a hacer es acceder al programa Filezilla.
- Paso 2: Introducimos las claves proporcionadas por el consultor en el menú de grupos del Campus Virtual y, una vez dentro de ese menú haremos clic en la opción que lleva el nombre de nuestro grupo, en este caso Operatix 2.0.
- Paso 3: Una vez dentro, veremos la opción 'anuncios' y allí encontraremos el mensaje del consultor con las claves personalizadas para nuestro proyecto y equipo.
- Paso 4: Introducimos las claves en el programa Filezilla e importamos allí nuestro repositorio de GitHub del producto 3.

2. Importar la base de datos proporcionada utilizada en el producto 2.

A. Se utilizará la base de datos completa que hemos utilizado en el producto 2, junto con los registros creados.

Para importar la base de datos que hemos utilizado en el proyecto anterior, los pasos a seguir son muy sencillos y son los siguientes:

- Paso 1: Entramos en nuestro localhost:8082 donde tenemos alojado nuestro contenedor con Phpmyadmin.
- Paso 2: Dentro de la interfaz principal, a la izquierda encontraremos una serie de bases de datos con tablas.
- Paso 3: Seleccionamos la que lleva el nombre de la que usamos en el producto 2 y entramos en ella.
- Paso 4: Una vez entramos, una de las opciones de la barra de navegación situada en la parte superior de la pantalla es directamente 'Exportar'.
- Paso 5: Hacemos clic en 'Exportar' y directamente se nos descarga en el navegador el archivo .sql con nuestra base de datos configurada y los datos insertados en las tablas hasta ahora.

- Paso 6: Abriendo el repositorio para el producto 3, vamos a acceder de nuevo al localhost:8082 donde está el phpmyadmin del producto 3.
- Paso 7: En el menú vertical de la izquierda seleccionamos la opción 'Crear Nueva'.
- Paso 8: La llamamos de nuevo 'isla_transfers' y entramos en ella.
- Paso 9: Dentro de la base de datos, seleccionamos la opción 'importar' en el menú situado en la parte superior de la pantalla.
- Paso 10: En la interfaz de importar, la primera opción que tenemos es la de cargar un archivo de configuración de bases de datos donde vamos a cargar el archivo .sql que hemos descargado previamente.
- Paso 11: Aceptamos sin tocar nada más de la interfaz de importación y observamos como la base de datos se crea con éxito y comprobamos que las tablas y datos guardados son los correctos.

B. Se podrá modificar la base de datos añadiendo campos o tablas nuevas solamente si se quiere añadir alguna función adicional al proyecto, no para facilitar las funciones básicas.

3. Desarrollar el producto 2 al Framework Laravel mejorando el aspecto gráfico.

- Paso 1: Lo primero que debemos hacer para poder migrar con éxito nuestro proyecto de PHP nativo a Laravel es generar dos archivos que son de vital importancia para cualquier proyecto realizado con Laravel que son: el web.php (dentro de la carpeta 'Routes' para a posteriori escribir en él todas las rutas de archivos de vistas y controladores; y el index.php alojado dentro de la carpeta 'Public' para que, al redireccionar el 'Dockerfile' apunte directamente a ese index.php y no al de PHP nativo.
- Paso 2: Para seguir con una migración correcta, en el siguiente paso vamos a comprobar que efectivamente ya estamos trabajando con Laravel con lo que accederemos a nuestro localhost:8080.
- Paso 3: Una vez comprobado que localhost:8080 apunta correctamente al index.php de la carpeta 'Public', iniciamos la migración de archivos de nuestra arquitectura MVC.
- Paso 4: Lo más sencillo bajo nuestro punto de vista es migrar los archivos de los controladores y los modelos primero, puesto que el código que se

debe modificar es poco y relativamente sencillo, sólo habrá que cambiar algunas referencias cuando se empiecen a enrutar los archivos en el web.php.

- Paso 5: Finalmente, vamos a migrar los archivos que configuran las vistas de nuestro proyecto, que en nuestro caso están divididas en 3: Admin, Clientes y Reservas para mejor organización y aclaración de procesos.
- Paso 6: Las carpetas de vistas, por recomendación de Laravel, van a ir dentro de la carpeta llamada 'Resources' y dentro de ella encontraremos una carpeta que directamente se llama 'Views'.
- Paso 7: Alojamos en la carpeta 'Views' anteriormente mencionada nuestras carpetas que por el momento contienen nuestras vistas en PHP nativo.
- Paso 8: Otra de las recomendaciones de Laravel para simplificar la lectura del código, es pasar nuestras vistas de .php a .blade.php para que puedan usar las plantillas de Laravel y a nosotros nos facilita el trabajo porque se vuelven sencillos archivos de HTML5 los cuales modificamos el estilo con el CSS3 que creamos para el producto 2.
- Paso 9: Si todo ha salido como debe, ahora mismo lo que deberíamos hacer es empezar a generar todas las rutas en el 'web.php' que conectan nuestros controladores con nuestras vistas con cada una de las interfaces y menús del producto 2.
- Paso 10: Una vez finalizadas las rutas correctamente, deberíamos tener el mismo producto 2 pero en Laravel en lugar de en PHP nativo.

4. **Los pasos a seguir para mejorar el producto son:**

   A. Los usuarios corporativos los dará de alta el administrador y tendrán un usuario propio con acceso a un panel. En ese panel podrán realizar reservas con destino/origen (o ambos) el hotel. En este caso el hotel recibirá una comisión por el servicio que puede ser variable según lo pactado entre el hotel y la empresa de transfers. En el panel también podrán ver las comisiones que recibirán mes a mes.

   Para que los usuarios administrativos los de de alta el Administrador, debemos incluir, además de la vista pertinente y las rutas necesarias en web.php, esta función dentro de AdminController:

```php
// registrar cliente corporativo
public function registrarCorporativo(Request $request)
{
    $request->validate([
        'nombre'      => 'required|string|max:255',
        'apellido1'   => 'required|string|max:255',
        'apellido2'   => 'required|string|max:255',
        'direccion'   => 'required|string|max:255',
        'codigoPostal'=> 'required|string|max:10',
        'ciudad'      => 'required|string|max:100',
        'pais'        => 'required|string|max:100',
        'email'       => 'required|email|unique:transfer_viajeros,email',
        'password'    => 'required|string|min:8|confirmed',
    ]);

    try {
        $cliente = new Cliente();
        $cliente->nombre        = $request->nombre;
        $cliente->apellido1     = $request->apellido1;
        $cliente->apellido2     = $request->apellido2;
        $cliente->direccion     = $request->direccion;
        $cliente->codigoPostal  = $request->codigoPostal;
        $cliente->ciudad        = $request->ciudad;
        $cliente->pais          = $request->pais;
        $cliente->email         = $request->email;
        $cliente->password      = Hash::make($request->password);
        $cliente->tipo_cliente  = 'corporativo';
        $cliente->save();

        return redirect()->route('admin.corporativos.form')
                         ->with('success', '✅ Usuario corporativo registrado correctamente.');
    } catch (\Exception $e) {
        return back()->withInput()
            ->with('error', '❌ Error al registrar el usuario corporativo: ' . $e->getMessage());
    }
}
```

B. El administrador ha de poder ver un listado de las reservas realizadas por cada hotel y calcular la comisión total a pagar cada mes.

Para poder visualizar este nueva función, hemos incluido en AdminController, la nueva función llamada verResumenComisiones:

```php
public function verResumenComisiones(Request $request)
{
    $mes = $request->input('mes', now()->format('m'));
    $anio = $request->input('anio', now()->format('Y'));

    $reservas = DB::table('transfer_reservas')
        ->join('tranfer_hotel', 'transfer_reservas.id_hotel', '=', 'tranfer_hotel.id_hotel')
        ->select(
            'tranfer_hotel.nombre as nombre',
            DB::raw('COUNT(transfer_reservas.id_reserva) as total_reservas'),
            DB::raw('REPLACE(tranfer_hotel.Comision, " euros", "") as comision_unitaria'),
            DB::raw('COUNT(transfer_reservas.id_reserva) * REPLACE(tranfer_hotel.Comision, " euros", "") as total_comision')
        )
        ->whereMonth('transfer_reservas.fecha_reserva', '=', $mes)
        ->whereYear('transfer_reservas.fecha_reserva', '=', $anio)
        ->groupBy('transfer_reservas.id_hotel', 'tranfer_hotel.nombre', 'tranfer_hotel.Comision')
        ->get();

    return view('Admin.resumen_comisiones', compact('reservas', 'mes', 'anio'));
}
```

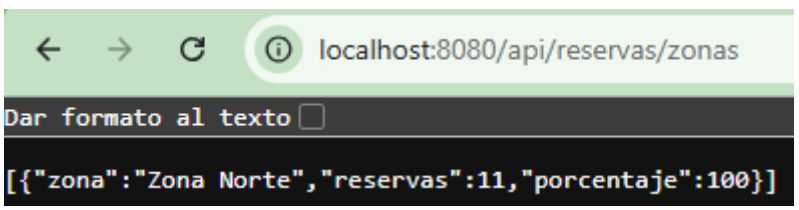
Además de crear la vista resumen_comisiones.blade.php, donde se visualiza toda esta información:

```blade
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Resumen de Comisiones</title>
    <link rel="stylesheet" href="{{ asset('css/styles.css') }}">
</head>
<body>
    <div class="panel-container">
        <h2>💼 Resumen de Comisiones por Hotel</h2>

        <form method="GET" action="{{ route('admin.comisiones.resumen') }}">
            <label for="mes">Mes:</label>
            <select name="mes" id="mes">
                @for ($m = 1; $m <= 12; $m++)
                    <option value="{{ sprintf('%02d', $m) }}" {{ $mes == $m ? 'selected' : '' }}>
                        {{ DateTime::createFromFormat('!m', $m)->format('F') }}
                    </option>
                @endfor
            </select>

            <label for="anio">Año:</label>
            <input type="number" name="anio" id="anio" value="{{ $anio }}" min="2020" max="{{ date('Y') }}">

            <button type="submit">Filtrar</button>
        </form>

        <table class="styled-table">
            <thead>
                <tr>
                    <th>🏨 Hotel</th>
                    <th>📊 Total Reservas</th>
                    <th>💸 Comisión por Reserva</th>
                    <th>💰 Total Comisión</th>
                </tr>
            </thead>
```

```
            </thead>
            <tbody>
                @php $totalGlobal = 0; @endphp
                @forelse ($reservas as $r)
                    <tr>
                        <td>{{ $r->nombre }}</td>
                        <td>{{ $r->total_reservas }}</td>
                        <td>{{ number_format((float)$r->comision_unitaria, 2) }} €</td>
                        <td><strong>{{ number_format((float)$r->total_comision, 2) }} €</strong></td>
                        @php $totalGlobal += (float)$r->total_comision; @endphp
                    </tr>
                @empty
                    <tr>
                        <td colspan="4">No hay datos para el periodo seleccionado.</td>
                    </tr>
                @endforelse
                @if(count($reservas))
                    <tr style="font-weight: bold;">
                        <td colspan="3" style="text-align: right;">Total Global</td>
                        <td>{{ number_format($totalGlobal, 2) }} €</td>
                    </tr>
                @endif
            </tbody>
        </table>

        <div class="volver-menu">
            <a href="{{ route('admin.home') }}">← Volver al Panel de Administración</a>
        </div>
    </div>
</body>
</html>
```

5. Se creará un "REST WebService" o procedimiento que devuelva un JSON con información agregada de las reservas realizadas a distintas zonas de la isla. En este sentido cada hotel está asignado a una zona. El resultado debe ser un listado de zona, número de traslados y % del total. El objetivo es mostrar en el producto 5 un listado de las reservas realizadas por zonas.

Para acceder a esta funcionalidad que hemos insertado en este Producto, deberemos acceder al siguiente link: http://localhost:8080/api/reservas/zonas



Para conseguir esto, el primer paso ha tenido que ser crear manualmente el repositorio de Providers (situado en src/app), que haciendo la instalación de Laravel, no se había creado correctamente. Ahí, nos aseguramos de tener dos archivos, que son AppServiceProvider.php y RouteServicePRovider.php, que se encargarán de gestionar los servicios Rest.

AppServiceProvider.php

```php
<?php

namespace app\Providers;

use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     */
    public function register(): void
    {
        //
    }

    /**
     * Bootstrap any application services.
     */
    public function boot(): void
    {
        //
    }
}
```

RouteServiceProvider.php

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Route;

class RouteServiceProvider extends ServiceProvider
{
    /**
     * Register services.
     */
    public function register(): void
    {
        //
    }

    /**
     * Bootstrap services.
     */
    public function boot(): void
    {
        Route::middleware('api')
            ->prefix('api')
            ->group(base_path('routes/api.php'));
    }
}
```

Finalmente, dentro de la carpeta routes, también debemos asegurarnos de tener el archivo api.php, donde incluiremos la ruta de ApiController.php:

api.php

Producto2_Operatix2.0 > Producto2_Operatix2.0 > Producto1 > Producto1 > src > routes > api.php

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ApiController;

// Ruta API para obtener resumen de reservas por zona
Route::get('/reservas/zonas', [ApiController::class, 'reservasPorZona']);
```

Este archivo anteriormente mencionado, está ubicado con el resto de Controladores en la ruta src/app/Http/Controllers y aquí podemos observar su lógica:

ApiController.php

Producto2_Operatix2.0 > Producto2_Operatix2.0 > Producto1 > Producto1 > src > app > Http > Controllers > ApiController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class ApiController extends Controller
{
    public function reservasPorZona()
    {
        // Total de todas las reservas
        $total = DB::table('transfer_reservas')->count();

        // Agregado por zona
        $zonas = DB::table('transfer_reservas')
            ->join('tranfer_hotel', 'transfer_reservas.id_hotel', '=', 'tranfer_hotel.id_hotel')
            ->join('transfer_zona', 'tranfer_hotel.id_zona', '=', 'transfer_zona.id_zona')
            ->select(
                'transfer_zona.nombre_zona as zona',
                DB::raw('COUNT(transfer_reservas.id_reserva) as reservas')
            )
            ->groupBy('transfer_zona.nombre_zona')
            ->get();

        // Añadir % del total a cada zona
        $zonas = $zonas->map(function ($zona) use ($total) {
            $zona->porcentaje = $total > 0 ? round(($zona->reservas / $total) * 100, 1) : 0;
            return $zona;
        });

        return response()->json($zonas);
    }
}

```

6. La aplicación debe responder a dominio.com/producto3 ya que en el producto 5 instalaremos en la raíz del dominio la web de la empresa.

Para que la aplicación responda a dominio.com/producto 3, simplemente deberemos modificar el archivo .env, para que este apartado quede de la siguiente manera:

```
APP_URL=http://localhost:8080/producto3
```

También tenemos que tener en cuenta, usar siempre en nuestras vistas Blade {{ asset() }} y {{url() }} siempre.

7. Enlaces : https://youtu.be/GebKvLU92uU

Enlace al repositorio de Github: https://github.com/Bassleader777/Producto3

Enlace al video de Youtube:

Enlace al Google Sites:
https://sites.google.com/d/1IVM9kFFg4yZi_tdM7_h9iWMeylVckQgk/p/1ODLnepU18CrtNlAF_5STQJzHMFv_sJol/edit