

PROYECTO DE APRENDIZAJE LARAVEL

Sistema de Gestión de Servicios y Comunicación entre Usuarios

INTRODUCCIÓN

Este documento presenta un proyecto de aprendizaje diseñado para estudiantes en prácticas en nuestra empresa. El objetivo es desarrollar un sistema de publicación de servicios (como clases de música, viajes, etc.) con un sistema de comunicación entre usuarios similar a un marketplace. El proyecto combina aspectos prácticos de desarrollo web con Laravel y sigue los requerimientos técnicos de nuestra empresa.

OBJETIVOS DE APRENDIZAJE

- Configurar un entorno de desarrollo con VirtualBox y Vagrant
 - Implementar un sistema de autenticación y gestión de usuarios
 - Desarrollar un sistema de publicación de servicios
 - Crear un sistema de mensajería entre usuarios
 - Implementar paginación, ordenamiento y gestión de archivos
 - Aplicar buenas prácticas de desarrollo y control de versiones
-

CONFIGURACIÓN DEL ENTORNO

Requisitos técnicos:

- VirtualBox como software de virtualización
- Vagrant para gestión de entornos virtuales
- Laravel 8.x o superior
- MySQL como base de datos
- Git para control de versiones

Pasos para la configuración:

1. Instalar VirtualBox y Vagrant:

- Descargar e instalar VirtualBox: <https://www.virtualbox.org/>
- Descargar e instalar Vagrant: <https://www.vagrantup.com/>

2. Configurar Vagrant:

```
# Crear directorio para el proyecto  
mkdir laravel-marketplace
```

```

cd laravel-marketplace

# Inicializar Vagrant con una box de Ubuntu
vagrant init ubuntu/focal64

```

3. **Configurar Vagrantfile:** Editar el archivo Vagrantfile para configurar la máquina virtual:

```

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.hostname = "solucionesdelem.demo"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 2
  end
  config.vm.provision "shell", path: "provision.sh"
end

```

4. **Crear script de aprovisionamiento:** Crear un archivo provision.sh con el siguiente contenido:

```

#!/bin/bash

# Actualizar repositorios
apt-get update

# Instalar dependencias básicas
apt-get install -y git curl zip unzip

# Instalar PHP y extensiones necesarias
apt-get install -y php8.0-cli php8.0-common php8.0-curl php8.0-mbstring php8.0-mysql ph

# Instalar Composer
curl -sS https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer

# Instalar MySQL
apt-get install -y mysql-server

# Configurar MySQL
mysql -e "CREATE USER 'laravel'@'localhost' IDENTIFIED BY 'password';"
mysql -e "CREATE DATABASE laravel;"
mysql -e "GRANT ALL PRIVILEGES ON laravel.* TO 'laravel'@'localhost';"
mysql -e "FLUSH PRIVILEGES;"

# Instalar Node.js y NPM
curl -sL https://deb.nodesource.com/setup_14.x | bash -

```

```

apt-get install -y nodejs

# Crear proyecto Laravel
cd /vagrant
composer create-project laravel/laravel .

# Configurar permisos
chown -R vagrant:vagrant /vagrant

# Instalar dependencias NPM
npm install

echo "Instalación completada. Accede a http://192.168.33.10 para ver tu aplicación Lara

```

5. Iniciar la máquina virtual:

```

chmod +x provision.sh
vagrant up

```

6. Acceder a la máquina virtual:

```

vagrant ssh
cd /vagrant

```

7. Verificar instalación:

- Ejecutar `php artisan serve --host=0.0.0.0`
 - Acceder a `http://192.168.33.10:8000` en el navegador
 - Modificar el archivo hosts del sistema para asignar `solucionesdelem.demo` a 192.168.33.10
-

Módulo 1: Autenticación y Gestión de Usuarios (4 días)

Tareas: - Implementar registro y autenticación usando Laravel Fortify/Jetstream - Configurar perfiles de usuario con avatar - Implementar dashboard de usuario

Paquetes recomendados: - **Laravel Jetstream** para autenticación - **Spatie Laravel Permission** para gestión de roles - **Livewire** para componentes interactivos

Implementación:

```

# Instalar Laravel Jetstream con Livewire
composer require laravel/jetstream
php artisan jetstream:install livewire

# Instalar Spatie Laravel Permission
composer require spatie/laravel-permission

```

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
php artisan migrate
```

Módulo 2: Sistema de Publicación de Servicios (4 días)

Tareas: - Crear modelo de datos para Servicios - Implementar CRUD completo para Servicios - Añadir categorías y etiquetas - Implementar sistema de búsqueda y filtrado

Estructura de datos: - Tabla `services` con campos: id, title, description, user_id, price, duration, category_id, status, created_at, updated_at - Tabla `categories` con campos: id, name, slug, created_at, updated_at

Implementación:

```
# Crear modelos y migraciones
php artisan make:model Service -mfsc
php artisan make:model Category -mfsc

# Livewire para componentes interactivos
php artisan make:livewire Services/Create
php artisan make:livewire Services/Index
php artisan make:livewire Services/Edit
```

Módulo 3: Sistema de Mensajería (6 días)

Tareas: - Implementar sistema de chats basado en publicaciones - Crear lógica para múltiples conversaciones - Implementar paginación y ordenamiento de mensajes - Permitir adjuntar archivos a los mensajes

Estructura de datos: - Tabla `conversations` con campos: id, service_id, user_id, created_at, updated_at - Tabla `messages` con campos: id, conversation_id, user_id, body, read_at, created_at, updated_at - Tabla `attachments` con campos: id, message_id, path, type, created_at, updated_at

Implementación:

```
# Crear modelos y migraciones
php artisan make:model Conversation -mfsc
php artisan make:model Message -mfsc
php artisan make:model Attachment -mfsc

# Componentes Livewire para chat
php artisan make:livewire Chat/Conversations
php artisan make:livewire Chat/Messages
php artisan make:livewire Chat/MessageForm
```

Requisitos específicos para el sistema de chat:

1. Chat entre usuario y anfitrión:

- Crear endpoint para enviar mensajes relacionados a un servicio
 - Guardar datos del emisor, receptor, fecha y contenido
2. **Múltiples conversaciones:**
 - Permitir que un usuario pueda tener múltiples conversaciones
 - Relacionar conversaciones con servicios específicos
 - Mostrar lista de conversaciones activas
 3. **Paginación y orden:**
 - Paginar mensajes en lotes de 15
 - Ordenar por fecha (más recientes primero)
 - Implementar lazy loading para cargar mensajes anteriores
 4. **Archivos adjuntos:**
 - Permitir adjuntar fotos y audios a los mensajes
 - Utilizar Laravel Storage para gestionar archivos
 - Mostrar previsualizaciones de archivos adjuntos
-

CRONOGRAMA DE TRABAJO (2 semanas)

Semana 1: Fundamentos y Sistema Básico

Días 1-2: Configuración y Familiarización

- Instalación y configuración del entorno con VirtualBox y Vagrant
- Exploración de la estructura de Laravel
- Primeros pasos con Git y control de versiones

Días 3-4: Sistema de Autenticación

- Implementación del sistema de autenticación con Jetstream
- Personalización de perfiles de usuario
- Configuración de permisos básicos

Día 5: Sistema de Servicios - Parte 1

- Creación de modelos y migraciones para servicios
- Relaciones entre servicios y usuarios
- Implementación de base de datos con seeders iniciales

Semana 2: Implementación del Sistema de Mensajería

Días 6-7: Sistema de Servicios - Parte 2

- Implementación de CRUD completo para servicios
- Diseño de interfaces con Livewire y Tailwind
- Implementación de filtros y búsqueda

Días 8-9: Sistema de Mensajería Básico

- Creación de modelos para conversaciones y mensajes
- Implementación de envío y recepción de mensajes
- Desarrollo de interfaces de chat

Días 10-11: Funcionalidades Avanzadas de Chat

- Implementación de paginación y ordenamiento
- Sistema de adjuntos para mensajes
- Notificaciones de mensajes nuevos

Días 12-14: Refinamiento y Pruebas

- Mejoras de interfaz de usuario
 - Depuración y corrección de errores
 - Documentación del código
 - Presentación final del proyecto
-

REQUERIMIENTOS TÉCNICOS DETALLADOS

1. Chat entre Usuario y Anfitrión

Endpoints necesarios: - POST /api/conversations - Crear nueva conversación - POST /api/messages - Enviar mensaje a una conversación

Controlador de ejemplo:

```
// Controlador para mensajes
public function store(Request $request)
{
    $request->validate([
        'conversation_id' => 'required|exists:conversations,id',
        'body' => 'required|string',
    ]);

    $message = Message::create([
        'conversation_id' => $request->conversation_id,
        'user_id' => auth()->id(),
        'body' => $request->body,
    ]);

    // Broadcast event for real-time updates
    broadcast(new MessageSent($message))->toOthers();

    return response()->json($message, 201);
}
```

2. Múltiples Chats

Lógica requerida: - Un usuario puede tener múltiples conversaciones - Cada conversación está relacionada con un servicio específico

Relaciones en los modelos:

```
// Modelo User
public function conversations()
{
    return $this->hasMany(Conversation::class);
}

// Modelo Conversation
public function service()
{
    return $this->belongsTo(Service::class);
}

public function messages()
{
    return $this->hasMany(Message::class);
}

public function participants()
{
    return $this->belongsToMany(User::class, 'conversation_user');
}
```

3. Paginación y Orden

Implementación en el controlador:

```
// Método para obtener mensajes paginados
public function getMessages(Conversation $conversation)
{
    $messages = $conversation->messages()
        ->with('user')
        ->orderBy('created_at', 'desc')
        ->paginate(15);

    return response()->json($messages);
}
```

Implementación en Livewire:

```
// Componente Livewire para mensajes
public function loadMessages()
{
```

```

    $this->messages = Message::where('conversation_id', $this->conversationId)
        ->with('user', 'attachments')
        ->orderBy('created_at', 'desc')
        ->paginate(15);
}

public function loadMore()
{
    $this->page++;
    $moreMessages = Message::where('conversation_id', $this->conversationId)
        ->with('user', 'attachments')
        ->orderBy('created_at', 'desc')
        ->paginate(15, ['*'], 'page', $this->page);

    $this->messages = $this->messages->merge($moreMessages);
}

```

4. Archivos Adjuntos

Método para subir archivos:

```

// Método para guardar archivos adjuntos
public function storeAttachment(Request $request)
{
    $request->validate([
        'message_id' => 'required|exists:messages,id',
        'file' => 'required|file|max:10240', // Max 10MB
    ]);

    $path = $request->file('file')->store('attachments', 'public');
    $type = $this->getFileType($request->file('file')->getMimeType());

    $attachment = Attachment::create([
        'message_id' => $request->message_id,
        'path' => $path,
        'type' => $type,
    ]);

    return response()->json($attachment, 201);
}

private function getFileType($mimeType)
{
    if (strpos($mimeType, 'image') !== false) {
        return 'image';
    } elseif (strpos($mimeType, 'audio') !== false) {

```

```

        return 'audio';
    }
    return 'file';
}

```

Configuración de Storage:

```
// Enlazar storage a la carpeta public
php artisan storage:link
```

GUÍA DE INSTALACIÓN Y CONFIGURACIÓN

Instalación de Paquetes Frontend

Basándonos en el package.json proporcionado, recomendamos:

```
# Actualizar dependencias existentes
npm update

# Instalar componentes adicionales (uno a uno para evitar conflictos)
npm install filepond --save # Para carga de archivos
npm install laravel-echo pusher-js --save # Para WebSockets (tiempo real)
npm install chart.js --save # Para gráficos y estadísticas (opcional)
```

Configuración de Herramientas de Desarrollo

```
# Instalar Laravel Debugbar (ayuda a depurar)
composer require barryvdh/laravel-debugbar --dev

# Configurar Debugbar
php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"

# Instalar IDE Helper (mejora el autocompletado en editores)
composer require barryvdh/laravel-ide-helper --dev
php artisan ide-helper:generate
php artisan ide-helper:models -N
```

Configuración de Webpack para Compilar Assets

```
// Añadir esto al archivo webpack.mix.js
mix.js('resources/js/app.js', 'public/js')
    .postCss('resources/css/app.css', 'public/css', [
        require('postcss-import'),
        require('tailwindcss'),
        require('autoprefixer'),
    ])
    .browserSync('solucionesdelem.demo'); // Para recargar automáticamente el navegador
```

```
// Para entorno de producción
if (mix.inProduction()) {
    mix.version();
}
```

Configuración de Storage para Archivos

```
# Crear enlace simbólico para storage
php artisan storage:link

# Configurar en .env
FILESYSTEM_DRIVER=public
```

RECURSOS DE APRENDIZAJE Y HERRAMIENTAS

Frameworks y Bibliotecas Principales

Backend (PHP)

- **Laravel 8+ Framework:** <https://laravel.com/docs>
 - Instalación: `composer create-project laravel/laravel proyecto-nombre`
- **Jetstream:** <https://jetstream.laravel.com>
 - Instalación: `composer require laravel/jetstream`
- **Livewire:** <https://laravel-livewire.com>
 - Instalación (con Jetstream): `php artisan jetstream:install livewire`
 - O independiente: `composer require livewire/livewire`

Paquetes de Spatie (Recomendados)

- **Laravel Permission:** <https://spatie.be/docs/laravel-permission>
 - Instalación: `composer require spatie/laravel-permission`
- **Laravel Media Library:** <https://spatie.be/docs/laravel-medialibrary>
 - Instalación: `composer require spatie/laravel-medialibrary`
- **Laravel Activitylog:** <https://spatie.be/docs/laravel-activitylog>
 - Instalación: `composer require spatie/laravel-activitylog`

Frontend

- **Tailwind CSS:** <https://tailwindcss.com>
 - Incluido con Jetstream
- **Alpine.js:** <https://alpinejs.dev>
 - Incluido con Jetstream + Livewire
- **Sweetalert2:** <https://sweetalert2.github.io>

- Instalación: `npm install sweetalert2`
- **Filepond:** <https://pqina.nl/filepond>
 - Instalación: `npm install filepond`
- **Flatpickr:** <https://flatpickr.js.org>
 - Instalación: `npm install flatpickr`

Herramientas de Desarrollo

- **Laravel Debugbar:** <https://github.com/barryvdh/laravel-debugbar>
 - Instalación: `composer require barryvdh/laravel-debugbar --dev`
- **Laravel IDE Helper:** <https://github.com/barryvdh/laravel-ide-helper>
 - Instalación: `composer require barryvdh/laravel-ide-helper --dev`
- **Laravel Telescope:** <https://laravel.com/docs/telescope>
 - Instalación: `composer require laravel/telescope --dev`

Plataformas de Aprendizaje

Sitios de Cursos y Tutoriales

- **Laracasts:** <https://laracasts.com>
 - Curso recomendado: “Laravel 8 From Scratch” (gratuito)
 - Curso “Livewire Basics”
- **Laravel Daily:** <https://laraveldaily.com>
 - YouTube: <https://www.youtube.com/c/LaravelDaily>
 - Tutoriales prácticos sobre Laravel y Livewire
- **Codecourse:** <https://codecourse.com>
 - Tutoriales sobre Laravel, Livewire y componentes
- **DigitalOcean Tutorials:** <https://www.digitalocean.com/community/tutorials?q=laravel>
 - Tutoriales gratuitos de alta calidad

Recursos Específicos Recomendados

- **Laravel Livewire Screencasts:** <https://laravel-livewire.com/screencasts>
 - Tutoriales oficiales del creador de Livewire
- **Alpine.js Essentials:** <https://alpinejs.dev/start-here>
 - Guía oficial para aprender Alpine.js
- **Tailwind CSS Screencasts:** <https://tailwindcss.com/screencasts>
 - Tutoriales oficiales de Tailwind CSS

Canales de YouTube

- **Laravel Daily:** <https://www.youtube.com/c/LaravelDaily>
 - Tutoriales cortos y prácticos
- **Andre Madarang:** <https://www.youtube.com/c/drehimself>
 - Proyectos completos con Laravel

- **Bitfumes:** <https://www.youtube.com/c/Bitfumes>
 - Tutoriales detallados de Laravel
- **Caleb Porzio:** <https://www.youtube.com/calebporzio>
 - Creador de Livewire y Alpine.js

Comunidad y Soporte

- **Laravel.io Forums:** <https://laravel.io/forum>
- **Laracasts Forums:** <https://laracasts.com/discuss>
- **Stack Overflow:** <https://stackoverflow.com/questions/tagged/laravel>
- **Laravel News:** <https://laravel-news.com>
- **Laravel Discord:** <https://discord.com/invite/mPZNm7A>
- **Laravel Spain Slack:** <https://laraveles.com/slack> (comunidad española)

Documentación y Cheatsheets

- **Laravel Cheatsheet:** <https://laravel.com/docs/master/cheatsheet>
 - **Laravel Livewire Cheatsheet:** <https://laravel-livewire.com/docs/2.x/cheatsheet>
 - **Tailwind CSS Cheatsheet:** <https://tailwindcomponents.com/cheatsheet/>
 - **MySQL Cheatsheet:** <https://devhints.io/mysql>
 - **Git Cheatsheet:** <https://education.github.com/git-cheat-sheet-education.pdf>
-

EVALUACIÓN Y ENTREGABLES

Entregables

- Repositorio Git con el código completo
- Base de datos con migraciones y seeders
- Documentación de endpoints de API
- Script de instalación para la máquina virtual

Criterios de Evaluación

- Funcionalidad del sistema de publicación de servicios
 - Implementación correcta del sistema de mensajería
 - Calidad del código (convenciones PSR)
 - Uso de migraciones, seeders y factories
 - Diseño y usabilidad de la interfaz
-

EJERCICIO DE PRUEBA FINAL

Para verificar el aprendizaje, el estudiante deberá crear: 1. Un servicio nuevo (ej. “Aprende a tocar el piano”) 2. Dos usuarios de prueba (ej. “DANIEL485”)

y “MARIA_1234”) 3. Iniciar una conversación entre los usuarios sobre el servicio 4. Demostrar la funcionalidad de múltiples conversaciones 5. Mostrar la paginación y ordenamiento de mensajes 6. Adjuntar un archivo a un mensaje

El estudiante deberá presentar este ejercicio al finalizar el periodo de desarrollo. No se espera que el estudiante complete todas las funcionalidades avanzadas si el tiempo no es suficiente - lo más importante es que comprenda los conceptos fundamentales y logre implementar correctamente las funcionalidades básicas.