Marco de Análisis de Prompts para IA de Desarrollo Web

Criterios de Evaluación y Sistema de Puntuación (Escala de 1 a 5)

Categoría	Descripción	Criterios de Puntuación	Peso
Claridad	Nivel de claridad y ausencia de ambigüedad	1: Altamente ambiguo y confuso 2: Varios elementos poco claros 3: Mayormente claro con alguna ambigüedad 4: Claro con mínima ambigüedad 5: Perfectamente claro y sin ambigüedades	15%
Especificidad	Nivel de detalle y precisión en los requisitos	1: Extremadamente vago 2: Sólo información básica 3: Detalle moderado 4: Requisitos detallados 5: Especificaciones precisas y completas	20%
Precisión Técnica	Uso correcto de términos y conceptos técnicos	1: Errores técnicos graves 2: Varias imprecisiones técnicas 3: Problemas menores técnicos 4: Mayormente preciso 5: Totalmente preciso	15%
Provisión de Contexto	Información de fondo y contexto del proyecto	1: Sin contexto 2: Contexto mínimo 3: Contexto básico relevante 4: Buena información contextual 5: Contexto completo	10%
Calidad de la Estructura	Organización y flujo lógico del prompt	1: Estructura caótica 2: Organización deficiente 3: Estructura adecuada 4: Bien organizado 5: Estructura lógica excelente	10%
Calidad del Código Generado	Calidad del código resultante	1: Código no funcional 2: Funcional pero de baja calidad 3: Calidad aceptable 4: Buena calidad con problemas menores 5: Código excelente y listo para producción	30%

Plantilla de Documentación de Prompts

Campo	Contenido
ID del Prompt	P0019-SocialMediaPart
Fecha de Uso	11/03/2025
Modelo de IA	Claude Sonnet 3.7
Categoría de la Tarea	Frontend/Backend/Base de Datos/Autenticación
Tecnología	HTML/CSS/JavaScript/PHP
Texto Completo del Prompt	V.1 Prompt inicial: Dame el código para: Fase 5: Red social * Sistema de mensajería privada * Creación de foros y temas * Sistema de comentarios 1º.Iteración: Dame las rutas para: <?php namespace App\Http\Controllers; use App\Models\Comment; use App\Models\Forum;

```
use App\Models\Topic;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class CommentController extends Controller
{
    /**
     * Añadir comentario a un tema.
     */
    public function store(Request $request, Forum $forum, Topic $topic)
    {
        $request->validate([
            'content' => 'required|string',
        ]);
        // Verificar que el tema pertenece al foro
        if ($topic->forum_id != $forum->id) {
            return redirect()->route('forums.index');
        }
        $comment = new Comment();
        $comment->topic_id = $topic->id;
        $comment->user_id = Auth::id();
        $comment->content = $request->content;
        $comment->save();
        return redirect()->route('topics.show', [$forum, $topic])
            ->with('success', 'Comentario añadido correctamente');
    }
}
```

2º.Iteración: Dame las rutas para: <?php

```
namespace App\Http\Controllers;
use App\Models\Forum;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class ForumController extends Controller
{
    /**
     * Mostrar lista de foros disponibles según rol.
     */
    public function index()
    {
        $userRole = Auth::user()->role;
        // Mostrar foros para el rol del usuario
        $forums = Forum::where('role_access', $userRole)
            ->orWhere('role_access', 'all')
            ->orderBy('created_at', 'desc')
            ->get();
        return view('social.forums.index', compact('forums'));
    }
    /**
     * Mostrar foro específico con sus temas.
     */
    public function show(Forum $forum)
    {
        // Verificar que el usuario tenga acceso a este foro
        $userRole = Auth::user()->role;
        if ($forum->role_access != $userRole && $forum->role_access != 'all') {
            return redirect()->route('forums.index')
                ->with('error', 'No tienes permiso para acceder a este foro');
        }
        // Cargar los temas con relaciones eager loading
        $forum->load(['topics' => function ($query) {
            $query->orderBy('created_at', 'desc');
        }, 'topics.user']);
        return view('social.forums.show', compact('forum'));
    }
    /**
     * Crear nuevo foro (accesible por todos los usuarios).
     */
    public function store(Request $request)
    {
        $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
            'role_access' => 'required|in:cliente,vendedor,proveedor',
        ]);
        $forum = new Forum();
        $forum->user_id = Auth::id();
        $forum->title = $request->title;
        $forum->description = $request->description;
        $forum->role_access = $request->role_access;
        $forum->save();
        return redirect()->route('forums.show', $forum)
            ->with('success', 'Foro creado correctamente');
    }
    /**
     * Mostrar formulario para crear foro.
     */
    public function create()
    {
        return view('social.forums.create');
    }
}
```

3º.Iteración: Dame las rutas para: <?php

```
namespace App\Http\Controllers;
use App\Models\Message;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class MessageController extends Controller
{
    /**
     * Mostrar lista de conversaciones activas.
```

```
     */
    public function index()
    {
        $user = Auth::user();
        // Obtener usuarios con los que tiene conversaciones
        $conversations = Message::where('sender_id', $user->id)
            ->orWhere('receiver_id', $user->id)
            ->orderBy('created_at', 'desc')
            ->get()
            ->map(function ($message) use ($user) {
                return $message->sender_id == $user->id
                    ? $message->receiver_id
                    : $message->sender_id;
            })
            ->unique()
            ->values();
        $users = User::whereIn('id', $conversations)->get();
        return view('social.messages.index', compact('users'));
    }
    /**
     * Mostrar conversación con un usuario específico.
     */
    public function show(User $user)
    {
        $currentUser = Auth::user();
        // Actualizar mensajes como leídos
        Message::where('sender_id', $user->id)
            ->where('receiver_id', $currentUser->id)
            ->whereNull('read_at')
            ->update(['read_at' => now()]);
        // Obtener mensajes entre los dos usuarios
        $messages = Message::where(function ($query) use ($currentUser, $user) {
                $query->where('sender_id', $currentUser->id)
                    ->where('receiver_id', $user->id);
            })
            ->orWhere(function ($query) use ($currentUser, $user) {
                $query->where('sender_id', $user->id)
                    ->where('receiver_id', $currentUser->id);
            })
            ->orderBy('created_at')
            ->get();
        return view('social.messages.show', compact('user', 'messages'));
    }
    /**
     * Enviar un mensaje a un usuario.
     */
    public function store(Request $request, User $user)
    {
        $request->validate([
            'content' => 'required|string|max:1000',
        ]);
        $message = new Message();
        $message->sender_id = Auth::id();
        $message->receiver_id = $user->id;
        $message->content = $request->content;
        $message->save();
        return redirect()->route('messages.show', $user)->with('success', 'Mensaje enviado');
    }
}
```

4º.Iteración: Dame las rutas para: <?php

```
namespace App\Http\Controllers;
use App\Models\Forum;
use App\Models\Topic;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class TopicController extends Controller
{
    /**
     * Mostrar tema específico con sus comentarios.
     */
    public function show(Forum $forum, Topic $topic)
    {
        // Verificar que el tema pertenece al foro
        if ($topic->forum_id != $forum->id) {
            return redirect()->route('forums.index');
        }
        // Cargar comentarios con usuarios
        $topic->load(['comments' => function ($query) {
            $query->orderBy('created_at');
        }, 'comments.user', 'user']);
        return view('social.topics.show', compact('forum', 'topic'));
    }
    /**
     * Crear nuevo tema en un foro.
     */
    public function store(Request $request, Forum $forum)
    {
        $request->validate([
            'title' => 'required|string|max:255',
            'content' => 'required|string',
        ]);
        $topic = new Topic();
        $topic->forum_id = $forum->id;
        $topic->user_id = Auth::id();
        $topic->title = $request->title;
        $topic->content = $request->content;
        $topic->save();
        return redirect()->route('topics.show', [$forum, $topic])
            ->with('success', 'Tema creado correctamente');
    }
    /**
     * Mostrar formulario para crear tema.
     */
```

```
    public function create(Forum $forum)
    {
        return view('social.topics.create', compact('forum'));
    }
}
```

5º.Iteración: Cuando entro a esta ruta me da error 404: Route::get('/forums/{forum}/topics/create', [TopicController::class, 'create'])
->name('topics.create');

6º.Iteración: Añade al controlador un metodo para crear conversaciones privadas entre usuarios:

```
<?php
namespace App\Http\Controllers;
use App\Models\Message;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class MessageController extends Controller
{
    /**
     * Mostrar lista de conversaciones activas.
     */
    public function index()
    {
        $user = Auth::user();
        // Obtener usuarios con los que tiene conversaciones
        $conversations = Message::where('sender_id', $user->id)
            ->orWhere('receiver_id', $user->id)
            ->orderBy('created_at', 'desc')
            ->get()
            ->map(function ($message) use ($user) {
                return $message->sender_id == $user->id
                    ? $message->receiver_id
                    : $message->sender_id;
            })
            ->unique()
            ->values();
        $users = User::whereIn('id', $conversations)->get();
        return view('social.messages.index', compact('users'));
    }
    /**
     * Mostrar conversación con un usuario específico.
     */
    public function show(User $user)
    {
        $currentUser = Auth::user();
        // Actualizar mensajes como leídos
        Message::where('sender_id', $user->id)
            ->where('receiver_id', $currentUser->id)
            ->whereNull('read_at')
            ->update(['read_at' => now()]);
        // Obtener mensajes entre los dos usuarios
        $messages = Message::where(function ($query) use ($currentUser, $user) {
                $query->where('sender_id', $currentUser->id)
                    ->where('receiver_id', $user->id);
            })
            ->orWhere(function ($query) use ($currentUser, $user) {
                $query->where('sender_id', $user->id)
                    ->where('receiver_id', $currentUser->id);
            })
            ->orderBy('created_at')
            ->get();
        return view('social.messages.show', compact('user', 'messages'));
    }
    /**
     * Enviar un mensaje a un usuario.
     */
    public function store(Request $request, User $user)
    {
        $request->validate([
            'content' => 'required|string|max:1000',
        ]);
        $message = new Message();
        $message->sender_id = Auth::id();
        $message->receiver_id = $user->id;
        $message->content = $request->content;
        $message->save();
        return redirect()->route('messages.show', $user)->with('success', 'Mensaje enviado');
    }
}
```

7º.Iteración: Dame: social.messages.create

8º.Iteración: Añade un boton o un campo para crear una conversacion:

```
@extends('layouts.app')
@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-10">
            <div class="card">
                <div class="card-header">Mensajes</div>
                <div class="card-body">
                    @if(session('success'))
                        <div class="alert alert-success">
                            {{ session('success') }}
                        </div>
                    @endif
                    @if(count($users) > 0)
                        <div class="list-group">
                            @foreach($users as $user)
                                <a href="{{ route('messages.show', $user) }}" class="list-group-item list-group-item-action">
                                    <div class="d-flex justify-content-between align-items-center">
                                        <div>
                                            @if($user->profile_picture)
                                                <img src="{{ asset($user->profile_picture) }}" class="img-thumbnail rounded-circle" width="50">
```

```
                                @else
                                    <img src="{{ asset('img/default-user.png') }}" class="img-thumbnail rounded-circle" width="50">
                                @endif
                                <span class="ml-2">{{ $user->name }}</span>
                                <small class="text-muted ml-2">({{ ucfirst($user->role) }})</small>
                            </div>
                            @php
                                $unreadCount = App\Models\Message::where('sender_id', $user->id)
                                    ->where('receiver_id', Auth::id())
                                    ->whereNull('read_at')
                                    ->count();
                            @endphp
                            @if($unreadCount > 0)
                                <span class="badge badge-primary badge-pill">{{ $unreadCount }}</span>
                            @endif
                        </div>
                    </a>
                @endforeach
            </div>
        @else
            <p class="text-center">No tienes conversaciones activas</p>
        @endif
    </div>
</div>
</div>
</div>
@endsection
```

9º.Iteración: Añade iconos o algún campo en la barra de navegación para los usuarios registrados para poder acceder a los foros y las conversaciones privadas:

10º.Iteración: Modifica este campo para que tome el rol del usuario y no le deje seleccionar:

```
<select id="role_access" class="form-control @error('role_access') is-invalid @enderror" name="role_access" required>
    <option value="cliente" {{ old('role_access') == 'cliente' ? 'selected' : '' }}>Clientes</option>
    <option value="vendedor" {{ old('role_access') == 'vendedor' ? 'selected' : '' }}>Vendedores</option>
    <option value="proveedor" {{ old('role_access') == 'proveedor' ? 'selected' : '' }}>Proveedores</option>
</select>
```

11º.Iteración: La vista no manda al receptor, debe enviarse en la variable $user:

```
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-10">
            <div class="card">
                <div class="card-header d-flex justify-content-between align-items-center">
                    <div>
                        Conversación con {{ $user->name }}
                        <small class="text-muted">({{ ucfirst($user->role) }})</small>
                    </div>
                    <a href="{{ route('messages.index') }}" class="btn btn-sm btn-secondary">Volver</a>
                </div>
                <div class="card-body">
                    <div class="messages-container">
                        @foreach($messages as $message)
                            <div class="message {{ $message->sender_id == Auth::id() ? 'sent' : 'received' }}">
                                <div class="message-content {{ $message->sender_id == Auth::id() ? 'bg-primary text-white' : 'bg-light' }}">
                                    {{ $message->content }}
                                    <small class="message-time d-block">
                                        {{ $message->created_at->format('d/m/Y H:i') }}
                                    </small>
                                </div>
                            </div>
                        @endforeach
                    </div>
                    <form action="{{ route('messages.store', $user) }}" method="POST" class="mt-4">
                        @csrf
                        <div class="form-group">
                            <textarea name="content" rows="3" class="form-control @error('content') is-invalid @enderror" placeholder="Escribe tu mensaje..."></textarea>
                            @error('content')
                                <span class="invalid-feedback">{{ $message }}</span>
                            @enderror
                        </div>
                        <button type="submit" class="btn btn-primary">Enviar</button>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

pero da error:

```
public function store(Request $request, User $user)
    {
        $request->validate([
            'content' => 'required|string|max:1000',
        ]);
        $message = new Message();
        $message->sender_id = Auth::id();
        $message->receiver_id = $user->id;
        $message->content = $request->content;
        $message->save();
        return redirect()->route('messages.show', $user)->with('success', 'Mensaje enviado');
    }
```

12º.Iteración: Para solucionar el problema de que user es null envialo por el siguiente formulario:

```
<form action="{{ route('messages.store', $user) }}" method="POST" class="mt-4">
    @csrf
    <div class="form-group">
        <textarea name="content" rows="3"
            class="form-control @error('content') is-invalid @enderror"
            placeholder="Escribe tu mensaje..."></textarea>
        @error('content')
            <span class="invalid-feedback">{{ $message }}</span>
        @enderror
    </div>
    <button type="submit" class="btn btn-primary">Enviar</button>
</form>
```

13º.Iteración: me da error: SQLSTATE[23000]: Integrity constraint violation: 1048 Column 'receiver_id' cannot be null

	V.1.1 Desarrolla el código necesario para la Fase 5: Red Social del CMS del Mercado Local San Mateo, asegurando que se implemente en Laravel siguiendo buenas prácticas de arquitectura MVC. Esta fase debe incluir: 1. Sistema de Mensajería Privada: ○ Listado de conversaciones activas del usuario autenticado. ○ Vista de mensajes entre dos usuarios con actualización de estado de lectura. ○ Envío de mensajes con validaciones adecuadas. ○ Opción para iniciar una nueva conversación desde una lista de usuarios disponibles. 2. Foros y Temas: ○ Listado de foros accesibles según el rol del usuario (cliente, vendedor o proveedor). ○ Creación de nuevos foros con asignación automática del rol del usuario creador. ○ Visualización de un foro con sus temas y acceso restringido según rol. ○ Creación de temas dentro de un foro con validaciones adecuadas. 3. Sistema de Comentarios: ○ Visualización de un tema con todos sus comentarios. ○ Posibilidad de añadir comentarios a los temas existentes. Especificaciones Técnicas: ● Utilizar Laravel 8+ con autenticación basada en `Auth::user()`. ● Implementar rutas en `web.php` organizadas dentro de middleware `auth`. ● Los controladores deben seguir la estructura CRUD estándar (`index`, `show`, `create`, `store`). ● Implementar middleware para restringir acceso a foros según rol. ● Validaciones en controladores utilizando `request->validate()`. ● Uso de Eloquent con relaciones adecuadas (`hasMany`, `belongsTo`). ● Creación de vistas en Blade (`resources/views/social/...`) con Bootstrap para diseño responsivo. ● Incluir iconos en la barra de navegación para acceso a foros y mensajes privados. ● Solución al problema común de conflicto entre rutas dinámicas y estáticas (`/forums/{forum}/topics/create` vs `/forums/{forum}/topics/{topic}`). ● Enviar `receiver_id` correctamente en el formulario de mensajes para evitar errores de integridad en la base de datos. Documentos Adjuntos: especificaciones del proyecto y estructura de la base de datos si es necesario.	
Long. del Prompt	V.1 Prompt inicial: 124 caracteres Conjunto: 16898 caracteres	V.1.1 1934 caracteres
Documentos incluidos	Sí, documentos con las especificaciones del proyecto y base de datos	
Ejemplos Incluidos	No	
Calidad de la Respuesta	V.1 5	V.1.1 5
P. Ponderada	V.1 4,33	V.1.1 5
Prob. de Código	Referencias a funciones, rutas y vistas no implementadas. Funcionamiento no alineado con los requerimientos pedidos. Parámetros mal pasados en las rutas.	
Iteraciones	13	
Mejoras Recomendadas	Mayor especificidad para reducir iteraciones en la implementación.	

Tabla de Comparación de Versiones de Prompt

Versión	Camb. Clave	Cla.	Esp.	Prec. Téc.	Ctx.	Estruc.	Cal. Cód.	Punt. Pond.	Mej.
1.0	Inicial	4	3	5	4	5	5	4,33	-
1.1	Se agregaron especificaciones técnicas para evitar iteraciones.	5	5	5	5	5	5	5,00	0,67

Leyenda de Abreviaturas

- **Camb. Clave** = Cambios Clave
- **Cla.** = Claridad
- **Esp.** = Especificidad
- **Prec. Téc.** = Precisión Técnica
- **Ctx.** = Contexto
- **Estruc.** = Estructura
- **Cal. Cód.** = Calidad del Código
- **Punt. Pond.** = Puntuación Ponderada
- **Mej.** = Mejora