Imagen que contiene Logotipo

Descripción generada automáticamente

Trabajo Fin de Grado

***JOBWAVES***

Autor:

Carlos Oliva

Pablo Santorum

Tutor:

José Rodríguez de la Maya

Desarrollo de Aplicaciones Web - 2024

**Índice**

Contenido

[**1. Introducción** 4](#_Toc168244035)

[**1.1 Objetivos del Proyecto** 4](#_Toc168244036)

[**1.2 Alcance del Proyecto** 4](#_Toc168244037)

[**2. Descripción del Proyecto** 5](#_Toc168244038)

[**2.1 Descripción General** 5](#_Toc168244039)

[**2.2 Funcionalidades Principales** 5](#_Toc168244040)

[**3. Diseño del Sistema** 6](#_Toc168244041)

[**3.1 Arquitectura del Sistema** 6](#_Toc168244042)

[**3.2 Diagrama de Componentes** 7](#_Toc168244043)

[**4. Tecnologías y Herramientas Utilizadas** 8](#_Toc168244044)

[**4.1 Laravel** 8](#_Toc168244045)

[**4.1.1 Descripción y Justificación de Uso** 8](#_Toc168244046)

[**4.1.2 Principales Paquetes Utilizados** 8](#_Toc168244047)

[**4.2 Tailwind CSS** 9](#_Toc168244048)

[**4.3 Vite** 10](#_Toc168244049)

[**4.4 Otros (Composer, NPM, etc.)** 10](#_Toc168244050)

[**5. Desarrollo del Proyecto** 11](#_Toc168244051)

[**5.1 Configuración del Entorno de Desarrollo** 11](#_Toc168244052)

[**5.2 Estructura de Directorios del Proyecto** 13](#_Toc168244053)

[**5.3 Principales Componentes del Código** 14](#_Toc168244054)

[**5.3.1 Controladores** 14](#_Toc168244055)

[**5.3.2 Modelos** 15](#_Toc168244056)

[**5.3.3 Vistas** 16](#_Toc168244057)

[**6. Implementación de Funcionalidades** 18](#_Toc168244058)

[**6.1 Descripción de las Funcionalidades Implementadas** 18](#_Toc168244059)

[**6.2 Integración de Backend y Frontend** 20](#_Toc168244060)

[**7. Despliegue** 23](#_Toc168244061)

[**7.1 Proceso de Despliegue** 23](#_Toc168244062)

[**7.2 Herramientas Utilizadas para el Despliegue** 24](#_Toc168244063)

[**8. Referencias** 25](#_Toc168244064)

[**8.1 Bibliografía** 25](#_Toc168244065)

# **1. Introducción**

## **1.1 Objetivos del Proyecto**

El proyecto "**JobWaves**" tiene como objetivo la creación de una plataforma web dedicada a la búsqueda de empleo IT para los usuarios y las empresas. Las características de la web incluyen:

**Facilitar la búsqueda de empleo**: Proporcionar recursos que permitan a los usuarios encontrar ofertas de trabajo relevantes de manera rápida y sencilla, utilizando filtros específicos como categoría de trabajo, salario y más.

**Mejorar la gestión de procesos de selección**: Ofrecer a las empresas herramientas que faciliten la publicación de ofertas de empleo y la gestión de las solicitudes recibidas, incluyendo la posibilidad de filtrar candidatos y contactar directamente con los postulantes adecuados.

**Asegurar una experiencia de usuario óptima**: Diseñar una interfaz de usuario amigable que garantice una navegación fluida y una experiencia satisfactoria tanto a usuarios como empresas

## **1.2 Alcance del Proyecto**

El alcance del proyecto incluye el desarrollo completo de la plataforma "**JobWaves**", cubriendo tanto el frontend como el backend, y abarcando las siguientes características:

**Registro y Autenticación**: Proveer un sistema seguro de registro y autenticación para usuarios y empresas, permitiendo la creación de cuentas y el acceso a funcionalidades personalizadas.

**Publicación de Ofertas**: Facilitar a las empresas la creación y publicación de ofertas de empleo detalladas, incluyendo descripciones de trabajo, requisitos, beneficios, y más.

**Búsqueda y Aplicación**: Ofrecer a los usuarios herramientas avanzadas para buscar ofertas de empleo mediante diversos filtros y aplicar a ellas de manera directa desde la plataforma.

**Notificaciones**: Implementar un sistema de notificaciones que mantenga a los usuarios informados sobre el estado de sus aplicaciones y nuevas ofertas relevantes.

**Panel de Administración**: Proporcionar a las empresas un panel de administración para gestionar las ofertas de empleo y las solicitudes recibidas, permitiendo un seguimiento eficaz del proceso de selección.

# **2. Descripción del Proyecto**

## **2.1 Descripción General**

"**JobWaves**" es una aplicación web diseñada para servir como un puente entre buscadores de empleo y empleadores. La plataforma está destinada a facilitar el proceso de búsqueda de empleo para los usuarios y el proceso de selección de candidatos para las empresas. Los usuarios pueden registrarse, crear un perfil, buscar ofertas de empleo y aplicar directamente a través de la plataforma. Las empresas pueden publicar ofertas de trabajo, revisar las aplicaciones recibidas y gestionar el proceso de selección desde el mismo lugar. La plataforma está diseñada para ser intuitiva y eficiente, proporcionando una experiencia de usuario de alta calidad.

## **2.2 Funcionalidades Principales**

**Registro y Autenticación**: Los usuarios pueden registrarse con su correo electrónico y una contraseña. El sistema de autenticación asegura que solo los usuarios registrados puedan acceder a ciertas funcionalidades.

**Búsqueda de Empleo**: Los usuarios pueden buscar ofertas de empleo utilizando filtros como categoría de trabajo, salario y más. Esta funcionalidad incluye una barra de búsqueda avanzada y opciones de filtrado dinámicas para mejorar la experiencia de usuario.

**Aplicación a Ofertas**: Los usuarios pueden enviar sus aplicaciones a las ofertas de empleo directamente desde la plataforma. Esta funcionalidad permite adjuntar documentos como currículum y cartas de presentación, y proporciona un seguimiento del estado de las aplicaciones enviadas.

**Publicación de Ofertas de Empleo**: Las empresas pueden crear y publicar ofertas de trabajo, incluyendo detalles como título del trabajo, descripción, salario y categoría de trabajo. La plataforma ofrece una interfaz amigable para la gestión de ofertas, permitiendo a las empresas editar y actualizar las publicaciones según sea necesario.

**Gestión de Solicitudes**: Las empresas pueden revisar las solicitudes de empleo, filtrar candidatos, contactar a los postulantes adecuados, y gestionar el proceso de selección de manera eficiente.

**Notificaciones**: El sistema envía notificaciones a los usuarios sobre el estado de sus aplicaciones y nuevas ofertas relevantes, mejorando la comunicación y el seguimiento del proceso de selección.

# **3. Diseño del Sistema**

El diseño del sistema es una fase crítica en el desarrollo de la aplicación, ya que define cómo se estructurarán los componentes del sistema, cómo interactuarán entre sí y cómo se gestionarán los datos. En esta sección, se detallarán la arquitectura del sistema, el diagrama de componentes y el diagrama de clases.

## **3.1 Arquitectura del Sistema**

La arquitectura del sistema para "**JobWaves**" se basa en una arquitectura en capas, que proporciona una separación clara entre las distintas preocupaciones de la aplicación. Las principales capas de la arquitectura son:

**Capa de Presentación**: Esta capa incluye la interfaz de usuario (UI) que interactúa con los usuarios. Utiliza tecnologías como HTML, Tailwind CSS, JavaScript, y frameworks como Laravel además de Livewire, herramienta de desarrollo full stack, para construir interfaces de usuario dinámicas.

**Capa de Aplicación**: También conocida como capa lógica o de negocio, esta capa maneja la lógica de negocio de la aplicación. Incluye controladores y servicios que procesan las solicitudes de los usuarios y las respuestas de los modelos de datos.

**Capa de Persistencia de Datos**: Esta capa maneja la interacción con la base de datos. Utiliza modelos de Eloquent ORM para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos MySQL.

**Capa de Infraestructura**: Esta capa proporciona servicios comunes como autenticación, notificaciones, y otras funcionalidades de soporte que pueden ser reutilizadas en toda la aplicación.

**Arquitectura de Tres Capas**

La arquitectura de tres capas se puede representar de la siguiente manera:

**1. Capa de Presentación**

Responsabilidad: Interactuar con el usuario final.

Tecnologías: HTML, Tailwind CSS, JavaScript, Laravel, Livewire.

**2. Capa de Aplicación**

Responsabilidad: Procesar la lógica de negocio y coordinar las interacciones entre la capa de presentación y la capa de datos.

Tecnologías: Laravel Framework (PHP).

**3. Capa de Datos**

Responsabilidad: Gestionar el almacenamiento y la recuperación de datos.

Tecnologías: MySQL, Eloquent ORM.

**Componentes Principales**

Servidor Web: Utilizamos Apache o Nginx para servir las solicitudes HTTP.

Servidor de Aplicaciones: Laravel gestiona la lógica de negocio y la interacción con la base de datos.

Base de Datos: MySQL se utiliza para el almacenamiento de datos.

## **3.2 Diagrama de Componentes**

El diagrama de componentes muestra los módulos principales del sistema y sus interacciones. Es una representación de alto nivel que facilita la comprensión de cómo se divide la aplicación en partes funcionales y cómo estas partes se comunican entre sí.

**Componentes Principales**

**1. Frontend**

Descripción: Maneja la interacción con el usuario final.

Componentes:

* Vistas (HTML, Blade)
* Scripts (JavaScript)
* Estilos (CSS, Tailwind CSS)

**2. Backend**

Descripción: Gestiona la lógica de negocio y las operaciones de datos.

Componentes:

* Controladores (VacancyController, ProfileController, CandidateController, NotificationController, Controller)
* Servicios (Autenticación, Notificaciones)
* Repositorios (Para la gestión de datos)

**3. Base de Datos**

Descripción: Almacena datos persistentes.

Componentes:

* Esquemas de base de datos
* Migraciones (Para la creación y actualización de tablas)
* Modelos (Candidate, Category, User, Salary, Vacancy)

**4. Servicios Externos**

Descripción: Servicios de terceros utilizados para funcionalidades específicas.

Componentes:

* Servicio de Envío de Correos (Para notificaciones por correo)
* APIs de Terceros

# **4. Tecnologías y Herramientas Utilizadas**

## **4.1 Laravel**

### **4.1.1 Descripción y Justificación de Uso**

Laravel es un framework de código abierto para el desarrollo de aplicaciones web con PHP. Laravel se ha elegido para este proyecto debido a sus características avanzadas. Algunas de las características clave de Laravel incluyen:

* **Simplicidad**: Laravel sigue el sistema de desarrollo MVC (Model-View-Controller), que ayuda a mantener el código organizado y limpio.
* **Enrutamiento Intuitivo**: Ofrece una forma simple y expresiva de definir rutas, lo que facilita el manejo de las solicitudes HTTP.
* **ORM Eloquent**: Proporciona un mapeo objeto-relacional que simplifica las interacciones con la base de datos.
* **Seguridad**: Incluye medidas de seguridad integradas, como la protección contra ataques de inyección SQL y CSRF (Cross-Site Request Forgery).
* **Herramientas de Desarrollo**: Laravel incluye herramientas como Artisan (para automatizar tareas comunes) y un sistema de migraciones (para gestionar la base de datos).

**Justificación de Uso**:

Se ha decidido usar Laravel por su capacidad para acelerar el desarrollo, facilitar el mantenimiento y proporcionar un conjunto completo de funcionalidades listas para usar. Además, su estructura modular y su soporte para paquetes adicionales permiten una gran flexibilidad y extensibilidad en el desarrollo de aplicaciones.

### **4.1.2 Principales Paquetes Utilizados**

En el desarrollo de "JobWaves", se han utilizado varios paquetes de Laravel que amplían sus funcionalidades básicas. Los principales paquetes utilizados son:

* **laravel/sanctum**: Utilizado para la autenticación de usuarios mediante tokens API. Sanctum permite la creación de tokens API para usuarios sin la complejidad de OAuth.

Funcionalidades Clave: Autenticación de usuario, manejo de tokens API, seguridad.

* **livewire/livewire**: Facilita la creación de interfaces de usuario dinámicas y reactivas sin necesidad de escribir mucho JavaScript. Livewire permite construir componentes de interfaz que pueden reaccionar a cambios en el servidor en tiempo real.

Funcionalidades Clave: Interactividad, simplicidad en la gestión del estado de la UI, compatibilidad con Blade.

* **laravel/tinker:** Permite interactuar con tu aplicación Laravel desde la línea de comandos. Esto es útil para probar y depurar código rápidamente.

Funcionalidades Clave: Interactividad con la aplicación, depuración, prueba de código.

* **laravel/breeze:** Proporciona una implementación mínima y sencilla de autenticación de usuarios, incluyendo vistas, controladores y rutas preconfiguradas.

Funcionalidades Clave: Autenticación básica, simplicidad, facilidad de integración.

* **laravel/sail (require-dev):** Proporciona un entorno de desarrollo local basado en Docker para aplicaciones Laravel. Sail es útil para tener un entorno de desarrollo consistente.

Funcionalidades Clave: Entorno de desarrollo local, Docker, consistencia en el entorno.

## **4.2 Tailwind CSS**

Tailwind CSS es un framework de CSS de utilidad primero que proporciona clases de bajo nivel que se pueden combinar para construir cualquier diseño directamente con HTML. Tailwind se centra en permitir una personalización completa y rápida del diseño.

**Características Clave:**

* **Utilidades**: Proporciona una gran cantidad de clases de utilidad para el diseño, lo que permite un control preciso y una personalización total.
* **Responsive Design**: Facilita la creación de diseños responsivos con clases de utilidad específicas para diferentes puntos de interrupción.
* **Personalización**: Permite la personalización del tema de la aplicación ajustando configuraciones como colores, tamaños y tipografías en el archivo de configuración.
* **Optimización**: Genera archivos CSS mínimos al eliminar las clases no utilizadas.

**Justificación de Uso:**

Se usa Tailwind CSS por su flexibilidad y eficiencia en la creación de interfaces de usuario personalizadas. Permite a los desarrolladores crear diseños únicos sin necesidad de sobrescribir estilos predefinidos, y resulta en un código CSS más limpio y mantenible. Además, mejora el rendimiento de la aplicación.

## **4.3 Vite**

Vite es una herramienta de construcción de frontend rápida y moderna que ofrece una experiencia de desarrollo ágil y eficiente. Fue creada para abordar las limitaciones de las herramientas de construcción tradicionales, como Webpack, ofreciendo una compilación más rápida y una mejor experiencia de desarrollo.

**Características Clave:**

* **Desarrollo Rápido**: Vite utiliza una técnica de servidor de desarrollo basado en módulos ES nativos, lo que permite tiempos de carga increíblemente rápidos.
* **Hot Module Replacement (HMR):** Proporciona reemplazo en caliente de módulos, lo que permite que las actualizaciones del código se reflejen instantáneamente en la aplicación sin necesidad de recargar toda la página.
* **Optimización de Producción**: Vite utiliza Rollup bajo el capó para la compilación de producción, garantizando un rendimiento óptimo y un tamaño de paquete reducido.
* **Compatibilidad con Múltiples Frameworks**: Soporta React, Vue, Svelte, y otros frameworks modernos de frontend.

**Justificación de Uso:**

Vite ha sido seleccionado por su capacidad para mejorar significativamente la experiencia de desarrollo mediante la reducción de los tiempos de compilación y la actualización en directo de módulos. Esto resulta en un flujo de trabajo más eficiente y productivo. Además, su enfoque en la optimización de la producción garantiza que las aplicaciones sean rápidas y ligeras en el entorno de producción.

## **4.4 Otros (Composer, NPM, etc.)**

Composer es un gestor de dependencias para PHP. Permite declarar las bibliotecas de las que depende el proyecto y las gestiona (instala/actualiza) automáticamente.

**Características Clave:**

* **Gestión de Paquetes**: Facilita la instalación, actualización y gestión de paquetes y bibliotecas de PHP.
* **Autoloading**: Proporciona un sistema de autoloading automático para clases PHP, simplificando la carga de clases y reduciendo el código manual.
* **Control de Versiones**: Permite especificar versiones exactas o rangos de versiones de paquetes, asegurando la compatibilidad y estabilidad del proyecto.

**Justificación de Uso:**

Composer es esencial para gestionar las dependencias del proyecto Laravel, permitiendo integrar y mantener bibliotecas de terceros de manera eficiente. Su uso garantiza que todas las dependencias estén correctamente instaladas y actualizadas, facilitando el desarrollo y el mantenimiento del proyecto.

NPM (Node Package Manager) es el gestor de paquetes predeterminado para Node.js. Se utiliza para gestionar las dependencias del frontend del proyecto, incluyendo bibliotecas y herramientas de construcción.

**Características Clave:**

* **Gestión de Paquetes**: Facilita la instalación, actualización y gestión de paquetes y bibliotecas de JavaScript.
* **Scripts de NPM**: Permite definir scripts personalizados para automatizar tareas comunes, como la compilación de CSS y JavaScript, pruebas, y despliegue.
* **Extenso Ecosistema**: Ofrece acceso a una vasta cantidad de paquetes y bibliotecas que pueden ser fácilmente integradas en el proyecto.

**Justificación de Uso:**

NPM se utiliza para gestionar las dependencias del frontend, como Tailwind CSS y Vite, y para ejecutar scripts de construcción y desarrollo. Esto asegura que todas las herramientas y bibliotecas necesarias estén disponibles y correctamente configuradas, optimizando el flujo de trabajo de desarrollo.

# **5. Desarrollo del Proyecto**

El desarrollo del proyecto "JobWaves" abarca desde la configuración inicial del entorno de desarrollo hasta la implementación y organización del código. Esta sección proporciona una visión detallada de estos aspectos clave.

## **5.1 Configuración del Entorno de Desarrollo**

Los pasos para configurar el entorno de desarrollo de "JobWaves" son los siguientes:

**1. Instalación de PHP y Composer:**

**PHP**: Es necesario tener PHP 8.1 o superior instalado en el sistema. PHP es el lenguaje de programación en el que está basado Laravel.

**Composer**: Es un gestor de dependencias para PHP que permite instalar y gestionar bibliotecas y paquetes necesarios para el proyecto.

*# Instalación de Composer*

*php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"*

*php composer-setup.php*

*php -r "unlink('composer-setup.php');"*

**2. Instalación de Node.js y NPM:**

**Node.js:** Se requiere Node.js para ejecutar scripts de desarrollo y herramientas de construcción.

**NPM (Node Package Manager):** Viene incluido con Node.js y se utiliza para gestionar las dependencias del frontend.

*# Instalación de Node.js y NPM*

*sudo apt install nodejs*

*sudo apt install npm*

**3. Clonación del Repositorio del Proyecto:**

Clonar el repositorio de GitHub que contiene el código fuente de "JobWaves".

*git clone* [*https://github.com/colidom/jobwaves.git*](https://github.com/colidom/jobwaves.git)

*cd jobwaves*

**4. Instalación de Dependencias:**

Utilizar Composer para instalar las dependencias de PHP especificadas en el archivo composer.json.

*composer install*

Utilizar NPM para instalar las dependencias de JavaScript especificadas en el archivo package.json.

*npm install*

**5. Configuración del Entorno:**

Copiar el archivo de ejemplo .env.example a .env y configurar las variables de entorno necesarias, como las credenciales de la base de datos, claves API, etc.

*cp .env.example .env*

*php artisan key:generate*

**6. Configuración de la Base de Datos:**

Crear una base de datos MySQL y actualizar el archivo .env con las credenciales de conexión.

Ejecutar las migraciones para crear las tablas necesarias en la base de datos.

*php artisan migrate*

**7. Compilación de Activos:**

Compilar los activos de frontend utilizando Vite

*npm run dev*

**8. Servidor de Desarrollo:**

Iniciar el servidor de desarrollo de Laravel para ejecutar la aplicación localmente

*php artisan serve*

## **5.2 Estructura de Directorios del Proyecto**

La estructura de directorios de un proyecto Laravel está organizada para facilitar la gestión y el mantenimiento del código. A continuación, se describe la estructura de directorios de "JobWaves" y la función de cada uno:

Descripción de los Directorios Principales:

* **app/:** Contiene el núcleo de la aplicación, incluyendo controladores, modelos y middleware.
  + **Console/:** Comandos personalizados de Artisan.
  + **Exceptions/:** Manejo de excepciones personalizadas.
  + **Http/:** Contiene controladores, middleware y solicitudes HTTP.
    - **Controllers/:** Controladores que gestionan la lógica de las solicitudes HTTP.
    - **Middleware/:** Middleware que intercepta las solicitudes HTTP para aplicar filtros o reglas adicionales.
  + **Models/:** Modelos Eloquent que interactúan con la base de datos.
* **bootstrap/:** Archivos de arranque y configuración inicial de la aplicación.
* **config/:** Archivos de configuración de la aplicación.
* **database/:** Archivos relacionados con la base de datos, como migraciones, fábricas y seeders.
  + **factories/:** Definiciones de fábricas para crear datos de prueba.
  + **migrations/:** Archivos de migración para definir la estructura de la base de datos.
  + **seeders/:** Seeders para poblar la base de datos con datos iniciales.
* **public/:** Contiene archivos públicos accesibles desde el navegador, incluyendo el punto de entrada principal (index.php).
* **resources/:** Recursos de vista y frontend.
  + **css/:** Archivos CSS.
  + **js/:** Archivos JavaScript.
  + **lang/:** Archivos de localización.
  + **views/:** Vistas Blade de Laravel.
* **routes/:** Definiciones de rutas de la aplicación.
* **storage/:** Archivos generados por la aplicación, como logs y cachés.
* **tests/:** Pruebas automatizadas.
* **vendor/:** Dependencias instaladas por Composer.

## **5.3 Principales Componentes del Código**

El código de "JobWaves" está organizado en varios componentes clave que incluyen controladores, modelos y vistas. A continuación, se describen estos componentes en detalle:

### **5.3.1 Controladores**

Los controladores son clases que agrupan la lógica para manejar solicitudes HTTP y retornar respuestas. Aquí se define cómo interactúan con los modelos y las vistas. Un ejemplo de controlador es **VacancyController.php**:

<?php

namespace App\Http\Controllers;

use App\Models\Vacancy;

use Illuminate\Http\Request;

class VacancyController extends Controller

{

    /\*\*

     \* Display a listing of the resource.

     \*/

    public function index()

    {

        $this->authorize('viewAny', Vacancy::class); // Previene el acceso a todo lo relacionado con este modelo

        return view('vacancies.index');

    }

    /\*\*

     \* Show the form for creating a new resource.

     \*/

    public function create()

    {

        $this->authorize('create', Vacancy::class); // Previene el acceso a todo lo relacionao con este modelo

        return view('vacancies.create');

    }

    /\*\*

     \* Display the specified resource.

     \*/

    public function show(Vacancy $vacancy)

    {

        return view('vacancies.show', [

            'vacancy' => $vacancy

        ]);

    }

    /\*\*

     \* Show the form for editing the specified resource.

     \*/

    public function edit(Vacancy $vacancy)

    {

        $this->authorize('update', $vacancy);

        return view('vacancies.edit', [

            'vacancy' => $vacancy

        ]);

    }

}

Este controlador maneja la visualización de todas las vacantes, la creación de nuevas vacantes y la validación de los datos enviados.

### **5.3.2 Modelos**

Los modelos en Laravel representan las entidades de la base de datos y contienen la lógica de negocio. El siguiente es un ejemplo de modelo para **Vacancy.php**:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Vacancy extends Model

{

    use HasFactory;

    protected $casts = ['end\_date' => 'date'];

    protected $fillable = [

        'title',

        'salary\_id',

        'category\_id',

        'company\_name',

        'end\_date',

        'job\_description',

        'image',

        'published',

        'user\_id'

    ];

    public function category()

    {

        return $this->belongsTo(Category::class);

    }

    public function salary()

    {

        return $this->belongsTo(Salary::class);

    }

    public function candidates()

    {

        return $this->hasMany(Candidate::class)->orderBy('created\_at', 'DESC');

    }

    public function recruiter()

    {

        return $this->belongsTo(User::class, 'user\_id');

    }

}

El modelo Vacancy utiliza la clase Model de Eloquent y define los campos fillable para la asignación masiva.

### **5.3.3 Vistas**

Las vistas en Laravel se encargan de mostrar la interfaz de usuario. Laravel utiliza Blade como su motor de plantillas. Un ejemplo de vista para mostrar una lista de candidatos (index.blade.php):

<x-app-layout>

    <x-slot name="header">

        <h2 class="font-semibold text-xl text-gray-200 dark:text-gray-900 leading-tight">

            {{ \_\_('Candidatos') }}

        </h2>

    </x-slot>

    <div class="py-12">

        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

            <div class="bg-white dark:bg-gray-50 overflow-hidden shadow-sm sm:rounded-lg">

                <div class="p-6 text-gray-700 dark:text-gray-900 border-b border-gray-200">

                    <h1 class="text-2xl font-bold text-center my-10">

                        Oferta de empleo: {{ $vacancy->title }}

                    </h1>

                    <div class="md:flex md:justify-center p-5">

                        <ul class="divide-y divide-gray-200 w-full">

                            @forelse ($vacancy->candidates as $candidate)

                                <li class="p-3 flex items-center">

                                    <div class="flex-1">

                                        <p class="text-xl font-medium text-gray-800">{{ $candidate->user->name }}</p>

                                        <p class="text-sm text-gray-600">{{ $candidate->user->email }}</p>

                                        <p class="text-sm font-medium text-gray-600">Inscrito hace:

                                            <span

                                                class="font-normal">{{ $candidate->created\_at->diffForHumans() }}</span>

                                        </p>

                                    </div>

                                    <div>

                                        <a class="inline-flex items-center shadow-sm px-3 py-2.5 border

                                            border-gray-300 text-sm leading-5 font-medium rounded-full

                                            text-gray-700 bg-white hover:bg-gray-50"

                                            href="{{ asset('storage/cv/' . $candidate->cv) }}" target="\_blank"

                                            rel="noreferrer noopener">

                                            Ver CV

                                        </a>

                                    </div>

                                </li>

                            @empty

                                <p class="p-3 text-center text-sm text-gray-600">Aún no se han registrado candidatos en

                                    esta oferta de empleo.</p>

                            @endforelse

                        </ul>

                    </div>

                </div>

            </div>

        </div>

</x-app-layout>

# **6. Implementación de Funcionalidades**

La implementación de funcionalidades es el núcleo del desarrollo de cualquier proyecto. En "JobWaves", se han implementado varias funcionalidades esenciales para proporcionar una experiencia de usuario completa y eficiente. Esta sección detalla estas funcionalidades, proporciona ejemplos de código y explica cómo se ha logrado la integración entre el backend y el frontend.

## **6.1 Descripción de las Funcionalidades Implementadas**

Las funcionalidades implementadas en "JobWaves" están diseñadas para cubrir los aspectos clave del flujo de trabajo de búsqueda de empleo y gestión de usuarios. A continuación, se describen las principales funcionalidades:

* **Registro y Autenticación de Usuarios:**
  + **Descripción**: Permite a los usuarios crear una cuenta y acceder al sistema mediante autenticación.
  + **Funcionalidad**: Los usuarios pueden registrarse proporcionando un nombre, correo electrónico y contraseña. Una vez registrados, pueden iniciar sesión utilizando sus credenciales. La autenticación está gestionada mediante tokens de acceso proporcionados por Laravel Sanctum.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* **Publicación y Gestión de Ofertas de Empleo:**
  + **Descripción**: Permite a las empresas publicar nuevas ofertas de empleo y gestionar las existentes.
  + **Funcionalidad**: Las empresas pueden crear, editar y eliminar ofertas de empleo. Cada oferta incluye detalles como el título del trabajo, descripción, salario y categoría.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Búsqueda y Aplicación a Ofertas de Empleo:**
  + **Descripción**: Facilita a los usuarios la búsqueda de ofertas de empleo y la aplicación a las mismas.
  + **Funcionalidad**: Los usuarios pueden buscar ofertas de empleo utilizando filtros como categoria, salario y palabras clave. Pueden aplicar a las ofertas directamente desde la plataforma, y gestionar sus aplicaciones desde su perfil.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Notificaciones:**
  + **Descripción**: Mantiene a los usuarios informados sobre eventos importantes, como nuevas ofertas de empleo y actualizaciones en sus aplicaciones.
  + **Funcionalidad**: Las notificaciones se muestran en tiempo real y pueden ser marcadas como leídas o eliminadas por los usuarios.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Teams

Descripción generada automáticamente

* **Panel de Administración:**
  + **Descripción**: Proporciona herramientas de gestión a los administradores del sistema.
  + **Funcionalidad**: Los administradores pueden gestionar usuarios y ofertas de empleo, revisar estadísticas y monitorear la actividad del sistema desde un panel de control centralizado.

## **6.2 Integración de Backend y Frontend**

**Rutas**

El archivo **web.php** define las rutas de la aplicación, conectando las URL con los controladores:

<?php

use App\Http\Controllers\CandidateController;

use App\Http\Controllers\HomeController;

use App\Http\Controllers\NotificationController;

use App\Http\Controllers\ProfileController;

use App\Http\Controllers\VacancyController;

use Illuminate\Support\Facades\Route;

/\*

|--------------------------------------------------------------------------

| Web Routes

|--------------------------------------------------------------------------

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider and all of them will

| be assigned to the "web" middleware group. Make something great!

|

\*/

Route::get('/', HomeController::class)->name('home');

// Vacancies

Route::get('/dashboard', [VacancyController::class, 'index'])->middleware(['auth', 'verified', 'rol.recruiter'])->name('vacancies.index');

Route::get('/vacancies/create', [VacancyController::class, 'create'])->middleware(['auth', 'verified'])->name('vacancies.create');

Route::get('/vacancies/{vacancy}/edit', [VacancyController::class, 'edit'])->middleware(['auth', 'verified'])->name('vacancies.edit');

Route::get('/vacancies/{vacancy}', [VacancyController::class, 'show'])->name('vacancies.show');

Route::get('/candidates/{vacancy}', [CandidateController::class, 'index'])->name('candidates.index');

// Notifications

Route::get('/notifications', NotificationController::class)->middleware(['auth', 'verified', 'rol.recruiter'])->name('notifications');

// Profile

Route::middleware('auth', 'verified')->group(function () {

    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');

    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');

    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

});

require \_\_DIR\_\_ . '/auth.php';

**Configuración de Vite**

El archivo **vite.config.js** configura Vite para manejar los recursos del frontend, como CSS y JavaScript:

import { defineConfig } from 'vite';

import laravel from 'laravel-vite-plugin';

export default defineConfig({

    plugins: [

        laravel({

            input: [

                'resources/css/app.css',

                'resources/js/app.js',

            ],

            refresh: true,

        }),

    ],

});

**Dependencias del Frontend**

El archivo **package.json** lista las dependencias del proyecto, incluyendo herramientas como Vite y Tailwind CSS:

{

    "private": true,

    "type": "module",

    "scripts": {

        "dev": "vite",

        "build": "vite build"

    },

    "devDependencies": {

        "@tailwindcss/forms": "^0.5.2",

        "alpinejs": "^3.4.2",

        "autoprefixer": "^10.4.2",

        "axios": "^1.6.4",

        "laravel-vite-plugin": "^1.0.0",

        "postcss": "^8.4.31",

        "tailwindcss": "^3.1.0",

        "vite": "^5.0.0"

    }

}

**Flujo de Trabajo**

**Definición de Rutas:** En el archivo web.php, se define una ruta para el recurso candidates que utiliza el CandidateController para manejar las solicitudes.

**Controlador**: El CandidateController maneja las solicitudes entrantes, procesa los datos y retorna las vistas correspondientes. Por ejemplo, el método index recupera todos los candidatos y los pasa a la vista index.blade.php.

**Vistas**: Las vistas Blade (index.blade.php) renderizan el HTML y pueden incluir CSS y JavaScript para mejorar la experiencia del usuario. Estos recursos son gestionados y compilados por Vite, configurado en vite.config.js.

**Compilación de Recursos:** Vite se encarga de compilar los archivos CSS y JavaScript definidos en resources/css/app.css y resources/js/app.js, asegurando que el frontend está correctamente integrado con el backend.

**Gestión de Dependencias**: Las dependencias del frontend, como Tailwind CSS para el diseño y Axios para las solicitudes HTTP, se gestionan a través de package.json.

Este flujo de trabajo asegura una integración fluida entre el backend en Laravel y el frontend manejado por Vite, proporcionando una estructura eficiente y organizada para el desarrollo de aplicaciones web.

# **7. Despliegue**

## **7.1 Proceso de Despliegue**

El proceso de despliegue a nivel local implica configurar el entorno de desarrollo en tu máquina local. Aquí se detallan los pasos necesarios:

**1. Clonar el Repositorio:**

Clona el repositorio del proyecto desde el sistema de control de versiones (por ejemplo, Git).

*git clone https://github.com/tuusuario/tu-proyecto.git*

*cd tu-proyecto*

**2. Instalar Dependencias:**

Usa Composer para instalar las dependencias de PHP.

*composer install*

Usa npm para instalar las dependencias de JavaScript.

*npm install*

**3. Configurar Variables de Entorno:**

Copia el archivo .env.example a .env y configura las variables de entorno según sea necesario.

*cp .env.example .env*

*nano .env*

Genera una clave de aplicación única.

*php artisan key:generate*

**4. Migrar y Poblar la Base de Datos:**

Ejecuta las migraciones para crear las tablas necesarias en tu base de datos local.

*php artisan migrate*

*php artisan db:seed*

**5. Iniciar el Servidor de Desarrollo:**

Inicia el servidor de desarrollo de Laravel.

*php artisan serve*

Compila y observa los cambios en los archivos frontend.

*npm run dev*

Estos comandos pondrán en funcionamiento el servidor web de desarrollo y la compilación de recursos frontend, permitiéndote ver y probar tu aplicación localmente.

## **7.2 Herramientas Utilizadas para el Despliegue**

Para desplegar la aplicación localmente, se utilizan las siguientes herramientas:

* **Git**: Para clonar el repositorio del proyecto.
* **Composer**: Para gestionar las dependencias de PHP.
* **npm**: Para gestionar las dependencias de JavaScript.
* **PHP**: Lenguaje de programación en el que está escrita la aplicación.
* **Laravel** **Artisan**: Herramienta de línea de comandos de Laravel para tareas de desarrollo y mantenimiento.
* **Laravel** **Mix**: Para compilar y gestionar los recursos frontend.

# **8. Referencias**

## **8.1 Bibliografía**

La bibliografía incluye sitios web, tutoriales, foros y otros materiales accesibles en internet que han sido útiles durante el desarrollo del proyecto "JobWaves".

**Documentación de Laravel -** <https://laravel.com/docs/11.x>: La documentación oficial del framework Laravel, que proporciona guías detalladas y referencias para todas las funcionalidades de Laravel.

**Documentación de Tailwind CSS** - <https://tailwindcss.com/docs/>: La documentación oficial para Tailwind CSS, que explica cómo utilizar las clases utilitarias para construir interfaces.

**Documentación de Vite** - <https://vitejs.dev/guide/>: La documentación oficial de Vite, que detalla cómo configurar y utilizar esta herramienta

**Stack Overflow** - <https://stackoverflow.com/>: Foro de preguntas y respuestas para desarrolladores, útil para resolver problemas específicos de código

**GitHub** - <https://github.com/>: Plataforma para alojar, revisar código y gestionar proyectos.