**Manual Técnico Digital Insight Assessment Tool (DIAT)**

**Descripción general del proyecto:**

Digital Insight Assessment Tool (DIAT) es una aplicación diseñada para medir la madurez digital de las empresas. Cuenta con parametrizaciones iniciales que permiten su adaptación y flexibilidad a futuros cambios. La aplicación genera formularios automáticamente, y presenta resultados configurables al usuario final, destacando áreas de mejora, acciones recomendadas, felicitaciones, y conclusiones. Además, DIAT ofrece visualización de los resultados mediante gráficos, facilitando la comprensión y análisis de los datos.

**Aspectos técnicos:**

DIAT es una aplicación web desarrollada con el framework Laravel, versión 8. Utiliza MySQL como motor de base de datos, gestionado a través de Eloquent. Para la visualización y el diseño de las páginas, emplea Blade, el motor de plantillas integrado en Laravel.

**Requisitos previos:**

El proyecto DIAT está construido sobre Laravel versión 8, por lo que los requisitos previos para su implementación son los correspondientes a esta versión del framework.

**PHP**: El proyecto requiere PHP versión ^7.3 o ^8.0.

**Composer**: Para gestionar las dependencias de PHP.

**Dependencias de Composer**:

* fruitcake/laravel-cors: Gestión de CORS en Laravel.
* guzzlehttp/guzzle: Cliente HTTP para solicitudes web.
* laravel/framework: Laravel Framework versión ^8.75.
* laravel/sanctum: Autenticación API con Laravel Sanctum.
* laravel/tinker: Consola interactiva de Laravel.
* laravel/ui: Generador de interfaces de usuario.
* laravelcollective/html: Manejo de formularios y HTML en Laravel.
* spatie/laravel-permission: Gestión de roles y permisos.
* symfony/http-client: Cliente HTTP de Symfony.
* symfony/mailgun-mailer, symfony/postmark-mailer: Proveedores de correo.

**Dependencias para desarrollo**:

* filp/whoops: Manejo de errores para desarrollo.
* facade/ignition: Herramienta de depuración para Laravel.
* phpunit/phpunit: Framework de pruebas.
* fakerphp/faker: Generador de datos falsos para pruebas.

1. **Instrucciones de instalación**:

Para el proceso de instalación

#### **1. Clonar el repositorio**

#### Clonar el repositorio desde GitHub:

git clone https://github.com/yicaritomont/DigitalAdvanceCheck.git

cd DigitalAdvanceCheck

#### **2. Instalar dependencias de PHP**

#### Tener Composer instalado. Luego, instalar las dependencias de PHP utilizando el siguiente comando:

composer install

#### **3. Configurar variables de entorno**

Copiar el archivo de ejemplo .env y renombrarlo a .env:

cp .env.example .env

Generar la clave de la aplicación de Laravel:

php artisan key:generate

#### **4. Configurar la base de datos**

En el archivo .env y configurar la conexión a la base de datos:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=nombre\_de\_la\_base\_de\_datos

DB\_USERNAME=tu\_usuario

DB\_PASSWORD=tu\_contraseña

Migrar las tablas a la base de datos y cargar la información predeterminada con:

php artisan migrate –seed

#### 

#### **5. Ejecutar el servidor (para ambientes locales)**

Finalmente, inicia el servidor de desarrollo de Laravel con:

php artisan serve

1. **Estructura del proyecto**:

El repositorio tiene la siguiente estructura de archivos:

**app/**: Contiene los controladores, modelos y lógica principal de la aplicación.

**bootstrap/**: Configuración inicial de Laravel.

**config/**: Archivos de configuración para la aplicación.

**database/**: Migraciones y archivos relacionados con la base de datos.

**public/**: Archivos accesibles públicamente (CSS, JS, imágenes).

**resources/**: Vistas y archivos front-end (Blade templates).

**routes/**: Define las rutas de la aplicación (web.php, api.php).

**storage/**: Almacena archivos generados por la aplicación, logs, caché.

**tests/**: Contiene pruebas unitarias e integraciones.

**composer.json**: Definiciones de dependencias PHP (Laravel).

**package.json**: Definiciones de dependencias de Node.js.

**README.md**: Archivo inicial de documentación.

1. **Instrucciones de uso**:

Una vez que DIAT se ha inicializado de manera local o en un servidor, la aplicación estará lista para su uso con la información predeterminada ya registrada en las semillas (seeders) de la base de datos. Esto asegura que la aplicación está preconfigurada con los datos necesarios para comenzar a operar:

* Usuario Administrador

Usuario: [administrador@gmail.com](mailto:administrador@gmail.com)

Contraseña: 123456Abc!

* Usuario General

Usuario: [correo@gmail.com](mailto:correo@gmail.com)

Contraseña: 123456Abc!

Los anteriores son usuarios por defecto para que se evidencie el panel de control en cada caso.

1. **Explicación del código**:

Al estar desarrollado bajo el framework se utilizaron los componentes incluidos dentro del mismo como:

* Controladores
* Modelos
* Vistas
* Migraciones
* Semillados

En DIAT, se implementó un CRUD completo para cada opción desarrollada, con todas las funcionalidades correspondientes. Cada CRUD se incluyó dentro de un directorio específico a nivel de vistas, y los controladores fueron optimizados para utilizar solo los métodos esenciales (index, create, update, y edit). En la mayoría de los modelos, se añadió un campo denominado "status" para gestionar el estado de los registros, lo que permite marcar un registro como activo o inactivo sin necesidad de eliminarlo, preservando la información.

A nivel de menú y sus opciones, se implementó un firewall de autenticación (auth) para verificar los roles de los usuarios. DIAT cuenta con dos roles principales:

1. **Administrador**: Responsable de la parametrización y puesta en marcha de la aplicación. Este rol tiene acceso a la visualización general de los resultados de las mediciones.

2. **Usuario**: Este es el rol de los usuarios finales, quienes pueden actualizar su información principal y realizar hasta tres mediciones de madurez digital.

Se realizaron modificaciones en los controladores de **Auth** y **Login** para gestionar la visualización del panel de control y las secciones del menú, adaptándose al rol del usuario en sesión.

1. **Contribuciones**:

Para realizar cualquier contribución adicional es importante que se lleven a cabo los siguientes pasos :

* Fork del repositorio, realizar un fork del repositorio GitHub para una copia personal del mismo.
* Clonar el fork en el equipo local.
* Crear una rama nueva, nombrar la rama según la funcionalidad que se va a incluir o editar.
* Realizar commits de los cambios realizados, incluir un mensaje descriptivo de los ajustes o nuevas funcionalidades ejecutadas.
* Enviar los cambios a través de un push y finalmente solicitar pull request del fork hacia el repositorio base.

A nivel de estilo de codificación, se deben seguir los lineamientos del estándar **PSR-12** para el código PHP, lo que incluye el uso de 4 espacios para la indentación (no tabulaciones). Los nombres de las clases deben estar en **CamelCase**, mientras que los nombres de las funciones deben seguir el formato **CamelCase**. Para mantener un código entendible, es necesario que, al añadir nuevas funcionalidades o realizar modificaciones, se asegure de incluir comentarios y documentación claros y concisos.

1. **Despliegue**:

Antes de realizar el despliegue, verificar que el servidor está configurado con lo siguiente:

**Servidor web**: Apache o Nginx.

**PHP**: Laravel requiere al menos PHP 7.3. Asegurarse de tener todas las extensiones requeridas instaladas (como OpenSSL, PDO, Mbstring, Tokenizer, XML, Ctype, JSON, BCMath).

**Base de datos**: MySQL.

**Composer**: Para manejar las dependencias del proyecto.

**Configuración del entorno**

Archivo .env: configurar correctamente el archivo .env para el entorno de producción:

APP\_ENV=production

APP\_DEBUG=false

APP\_KEY: Genera la clave de la aplicación

**Despliegue de la aplicación**

Transferencia de archivos, puede usar scp, rsync o FTP para transferir los archivos de la aplicación al servidor, o simplemente realice un clone del mismo.

**Instalación de dependencias**

Navegar al directorio de proyecto en el servidor y ejecutar:

composer install --optimize-autoloader --no-dev

Esto instalará las dependencias sin los paquetes de desarrollo y optimizará el autoload.

**Migraciones y Seeders**

Ejecutar las migraciones para crear las tablas en la base de datos:

php artisan migrate --force

Si es necesario, también puedes ejecutar seeders para poblar la base de datos:

php artisan db:seed --force

**Configuración de permisos**

Asegurarse de que los permisos de los directorios storage y bootstrap/cache sean correctos.

chmod -R 775 storage

chmod -R 775 bootstrap/cache

**Configuración del servidor web**

Si usa Apache, asegúrese de que el archivo .htaccess esté presente en el directorio raíz.

Para Nginx, el bloque de servidor debería estar configurado para apuntar al archivo public/index.php de tu aplicación Laravel. Ejemplo de configuración Nginx:

server {

listen 80;

server\_name tu\_dominio.com;

root /ruta/a/tu/proyecto/public;

index index.php index.html index.htm;

location / {

try\_files $uri $uri/ /index.php?$query\_string;

}

location ~ \.php$ {

include snippets/fastcgi-php.conf;

fastcgi\_pass unix:/var/run/php/php7.4-fpm.sock; # Cambia según tu versión de PHP

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

location ~ /\.ht {

deny all;

}

}

**Mejores prácticas de despliegue**

Caching: Considere habilitar el caching de la configuración y las rutas para mejorar el rendimiento:

php artisan config:cache

php artisan route:cache

php artisan view:cache