

20
24

DENTAL HUB MANUAL TÉCNICO

Desarrollado por:

Josías Isaac Alvarenga Romero
Luis Antonio Bonilla Elías
Alfonso Antonio Fernández Cotto
Karen Sofía Rivas Rivas
Diego Fernando Gómez Castro

CONTENIDO



03	Requerimientos
04	Librerías y Herramientas
05	Tecnologías
06	Pasos de Instalación
11	Arquitectura
12	Funcionalidad
13	Diseño de la base de datos
14	Unit Testing

REQUERIMIENTOS

Los requisitos mínimos de hardware para ejecutar nuestra aplicación son esenciales para garantizar una experiencia de usuario fluida y sin problemas. A continuación, se detallan los requisitos clave:

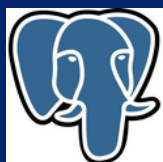
- Navegador Web (Chrome, Firefox, Safari, Edge)
- Mínimo 2 GB de Memoria RAM
- Procesador de al menos 1 GHz.
- Conexión a Internet Estable



Laravel v 11.20.0

Laravel es un marco PHP muy popular que ofrece una gran cantidad de características y herramientas para ayudar a los desarrolladores de software a crear aplicaciones web de alta calidad. Algunas ventajas por las cuales se eligió la implementación de esta son:

- Facilidad de uso
- Seguridad
- Escalabilidad
- Flexibilidad
- Productividad



PostgreSQL

PostgreSQL es un potente motor de bases de datos seleccionado por su libre uso y características de indexación que impulsa su rapidez.



Livewire v 3.5.4

Livewire es una librería de PHP que permite crear aplicaciones web interactivas y con muchas funcionalidades, sin tener que escribir código JavaScript complejo.



Alpine.js v 3.14.0

Alpine.js es una librería Javascript para el desarrollo de dinamismos en páginas web, con un enfoque sencillo y productivo.



Tailwind

Tailwind es un marco de diseño de código abierto utilizado en el desarrollo de aplicaciones web. Facilita el diseño responsivo adaptándose a las distintas resoluciones de dispositivos.



Filament v 3.2.102

Es un conjunto de herramientas convertidas en un único paquete para Laravel que nos permite desarrollar poderosos sistemas administrativos, todo esto de forma rápida, sencilla, flexible y eficiente.

Composer es un gestor de dependencias para PHP que facilita la administración de bibliotecas y paquetes en proyectos de desarrollo. Permite a los desarrolladores declarar las bibliotecas que su proyecto necesita y maneja automáticamente la instalación y actualización de estas dependencias, asegurando que se utilicen las versiones correctas y compatibles.

Esta herramienta es fundamental en el desarrollo moderno de aplicaciones PHP, ya que simplifica la gestión de dependencias y mejora la eficiencia del flujo de trabajo, permitiendo a los desarrolladores centrarse en la creación de código en lugar de preocuparse por la configuración de bibliotecas. Composer es ampliamente utilizado en la comunidad PHP y es esencial para frameworks populares como Laravel y Symfony.

Composer v 2.6.6

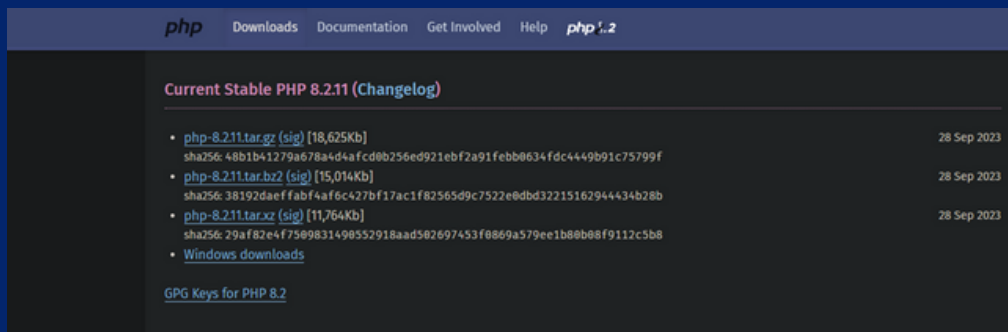
<https://getcomposer.org/download/>



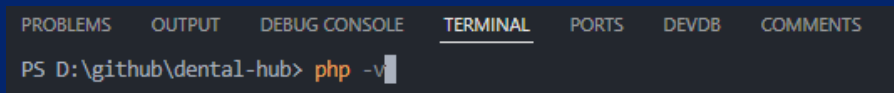
PASOS DE INSTALACIÓN

Instalar PHP

- Descarga e instala PHP desde el sitio web oficial de PHP siguiendo las instrucciones específicas para tu sistema operativo

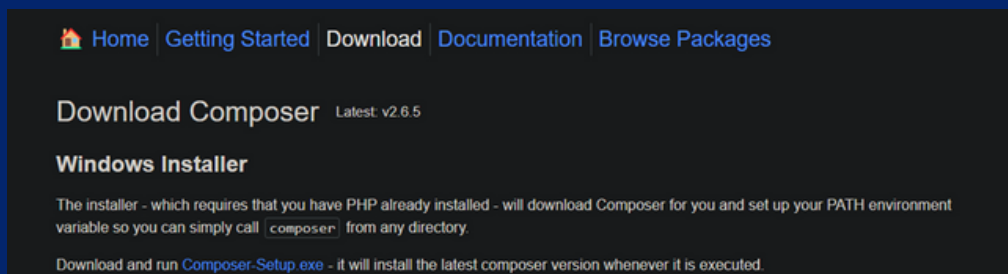


- Verifica la instalación de PHP ejecutando el siguiente comando en tu terminal o símbolo del sistema:

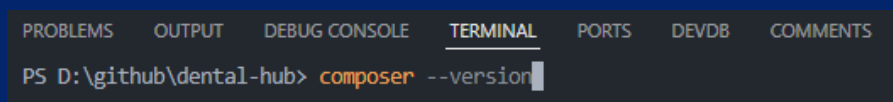


Instalar Composer

- Descarga e instala Composer desde getcomposer.org siguiendo las instrucciones para tu sistema operativo.



- Verifica la instalación de Composer ejecutando:

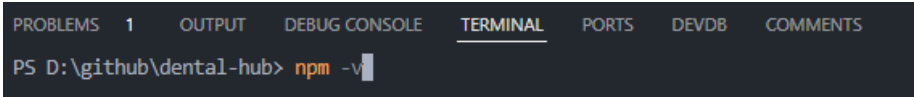
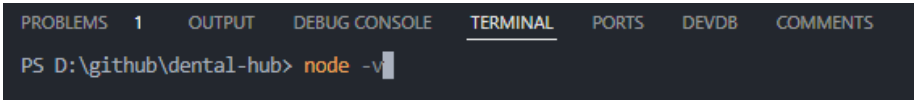


Instalar Node.js

- Descarga e instala Node.js desde nodejs.org, lo cual también instalará npm (Node Package Manager).

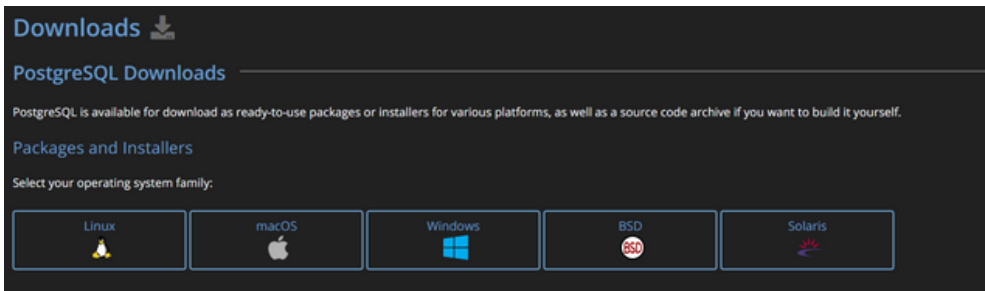


- Verifica la instalación de Composer ejecutando:



Instalar PostgreSQL

- Descarga e instala PostgreSQL desde <https://www.postgresql.org/download/> siguiendo las instrucciones específicas para tu sistema operativo.



Verifica la instalación de PostgreSQL ejecutando:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> psql -v
```

Instalar Laravel

- Instala Laravel globalmente usando Composer:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> composer global require laravel/installer
```

Instalar Livewire

- Instala Livewire usando Composer:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> composer require livewire/livewire
```

Instalar Filament

- Instala Filament usando Composer:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> composer require filament/filament
```


Pasos para clonar y ejecutar un proyecto de Filament

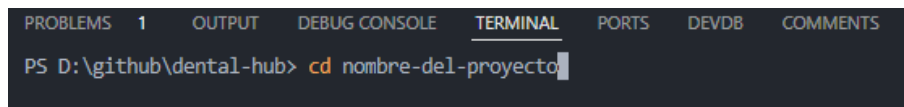
Clonar el repositorio:

A terminal window with a dark background. The prompt is 'bash'. The command 'git clone URL_DEL_REPOSITORIO' is entered. A 'Copy code' button is visible in the top right corner.

```
bash
git clone URL_DEL_REPOSITORIO
```

Sustituye URL_DEL_REPOSITORIO por la URL real del repositorio de Git donde se encuentra el proyecto de React Native.

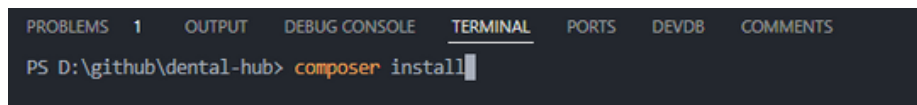
- Navegar al Directorio del Proyecto:

A terminal window with a dark background. The prompt is 'PS D:\github\dental-hub>'. The command 'cd nombre-del-proyecto' is entered. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, DEVDB, and COMMENTS.

```
PS D:\github\dental-hub> cd nombre-del-proyecto
```

Navega al directorio del proyecto clonado utilizando el nombre del directorio que se creó después de clonar el repositorio.

- Ejecutar comando composer install

A terminal window with a dark background. The prompt is 'PS D:\github\dental-hub>'. The command 'composer install' is entered. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, DEVDB, and COMMENTS.

```
PS D:\github\dental-hub> composer install
```

El comando composer install instala las dependencias del proyecto

Configuración

Copia el archivo .env.example y renómbralo a .env:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> cp .env.example .env
```

Configura la conexión a la base de datos y otras variables de entorno en el archivo .env

```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:veBGUMPd30MP+9Dd/nMUGFY+MEU4OLfr1kILICCS6M=
4 APP_DEBUG=true
5 APP_TIMEZONE=UTC
6 APP_URL=http://localhost
7
8 APP_LOCALE=en
9 APP_FALLBACK_LOCALE=en
10 APP_FAKER_LOCALE=en_US
11
12 APP_MAINTENANCE_DRIVER=file
13 # APP_MAINTENANCE_STORE=database
14
15 BCRYPT_ROUNDS=12
16
17 LOG_CHANNEL=stack
18 LOG_STACK=single
19 LOG_DEPRECATED_CHANNELS=null
20 LOG_LEVEL=debug
21
22 DB_CONNECTION=sqlite
23 DB_HOST=127.0.0.1
24 DB_PORT=3306
25 DB_DATABASE=laravel
26 DB_USERNAME=root
27 DB_PASSWORD=
```

- Genera una nueva clave de aplicación:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> php artisan key:generate
```

- Ejecuta las migraciones para crear las tablas de la base de datos:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB COMMENTS
PS D:\github\dental-hub> php artisan migrate
```

- Ejecutar el proyecto con el comando php artisan serve

```
PS D:\github\dental-hub> php artisan serve

INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
```

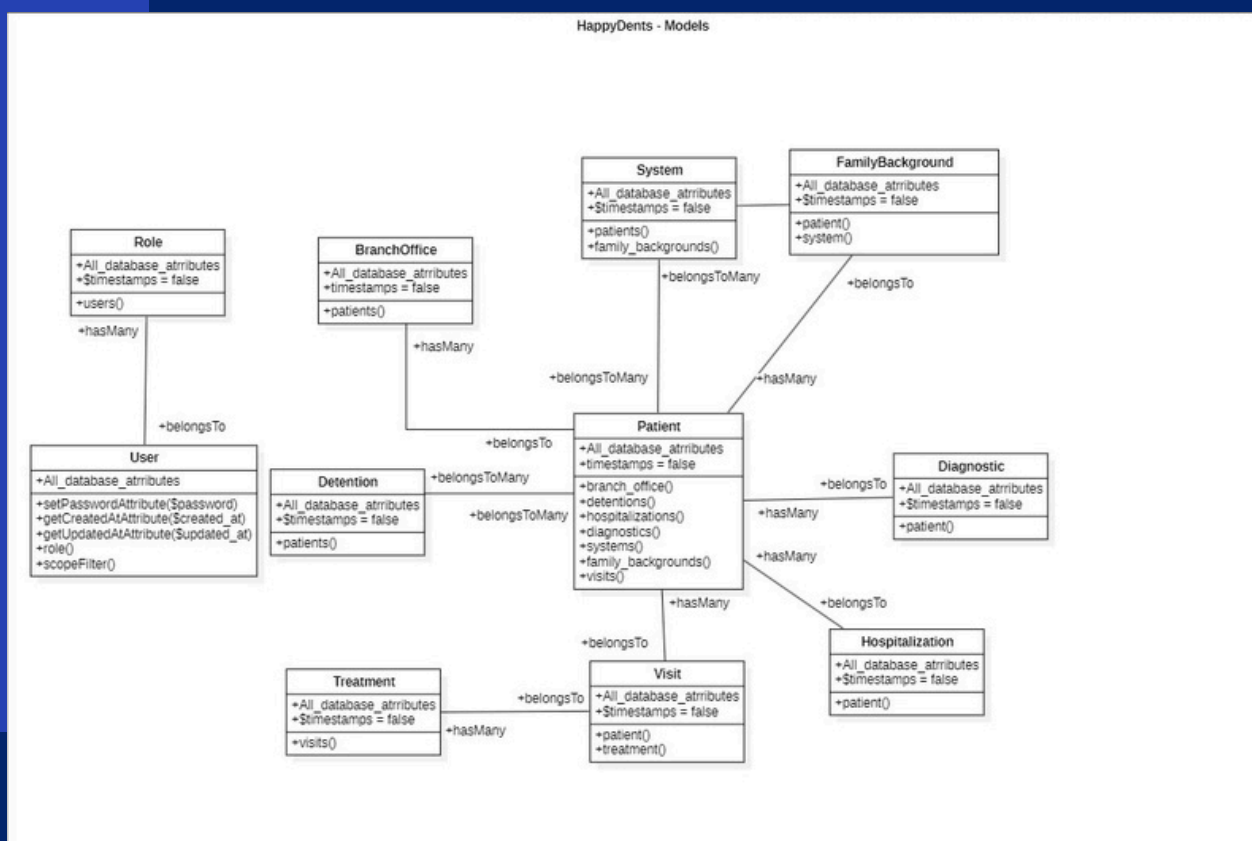
ARQUITECTURA

El siguiente diagrama de clases refleja la estructura fundamental de nuestro sistema clínico. En él se destacan los principales elementos y las relaciones que interactúan en el desarrollo de la aplicación.

Primero, tenemos a los Estados y Municipios, que forman la base geográfica de la estructura. Cada estado puede tener múltiples municipios, los cuales a su vez albergan tanto clínicas como pacientes. Las clínicas cuentan con varias sucursales, cada una de las cuales gestiona su propio personal, pacientes y citas. Las sucursales, además, están ligadas a los municipios en los que operan.

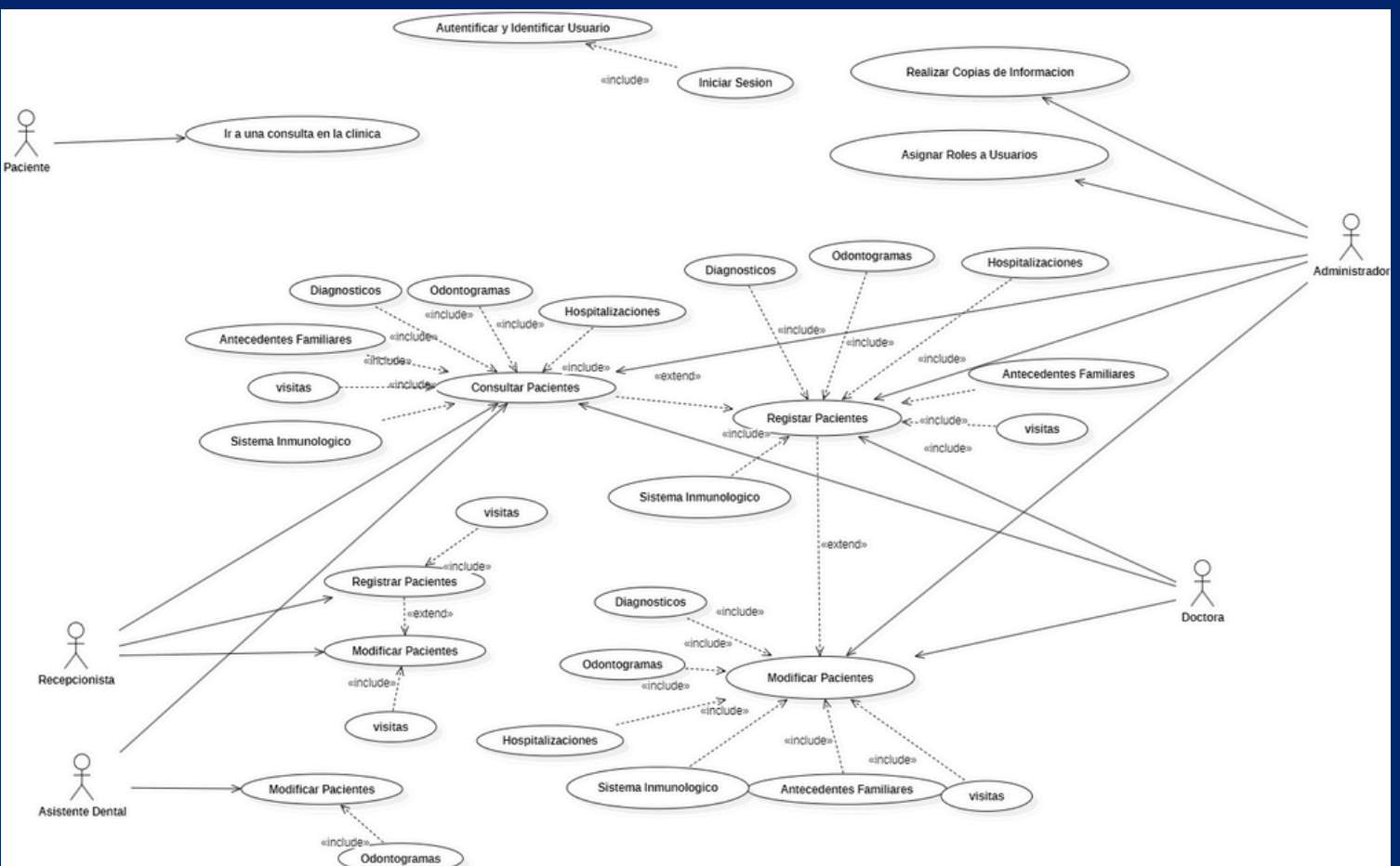
Un actor clave en el sistema es el Usuario. Los usuarios, que pueden tener distintos Roles, como administradores o personal médico, acceden al sistema y gestionan citas y pacientes. Las citas son gestionadas por el personal y están vinculadas a los Servicios Dentales, que son los tratamientos ofrecidos a los pacientes.

El sistema también contempla la gestión de los Pacientes, quienes son registrados con información detallada, como contactos de emergencia, historial médico y registros dentales. Estos registros permiten que el personal médico lleve un control detallado de la salud de los pacientes, facilitando el diagnóstico y el tratamiento.



DENTAL
HUB

Además, el diagrama pone énfasis en la interacción entre pacientes y el personal clínico a través de la gestión de citas y diagnósticos. Estas funcionalidades son esenciales para asegurar la correcta administración de los servicios dentales y garantizar la salud de los pacientes.

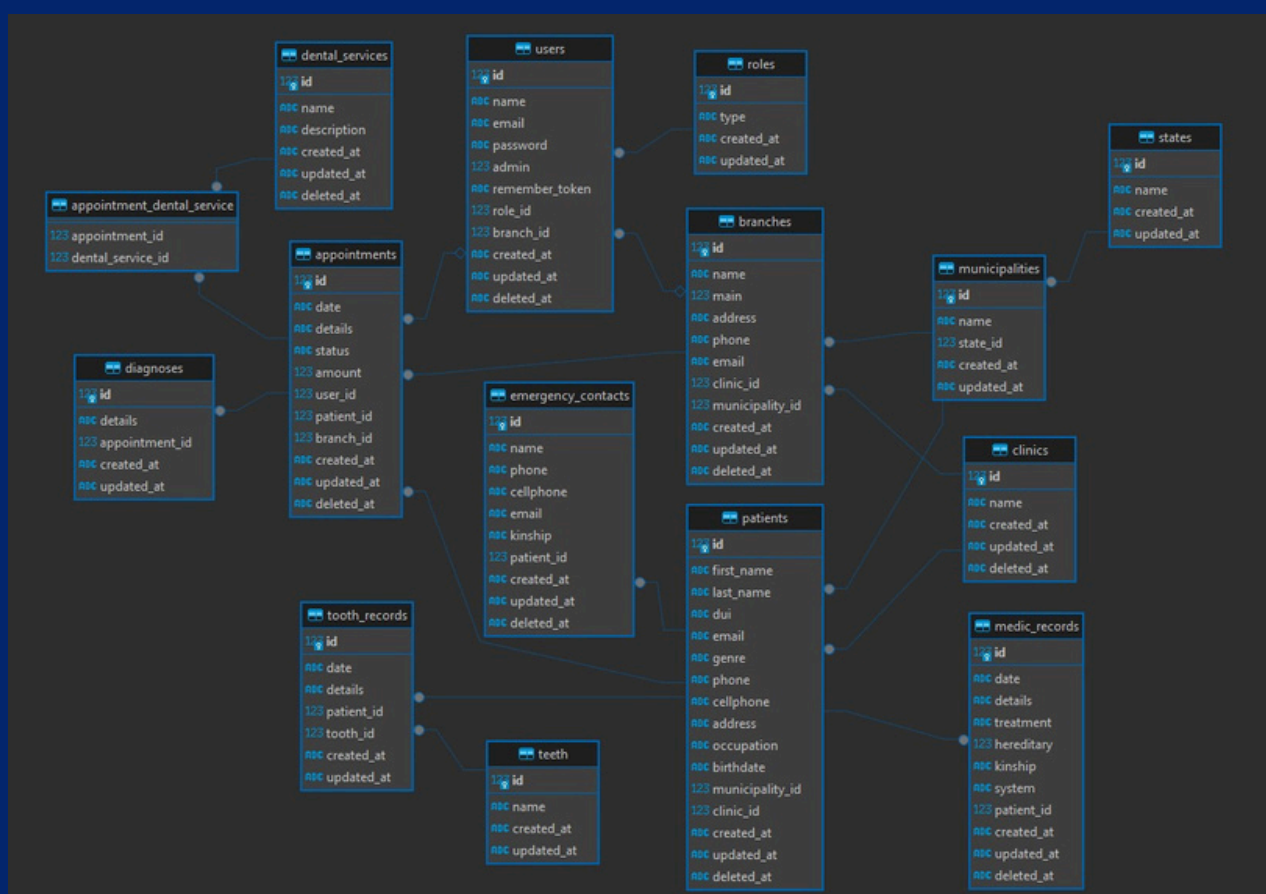


DISEÑO DE LA BASE DE DATOS

DENTAL
HUB

A continuación se proporciona un desglose detallado de todas las tablas que conforman la estructura, detallando sus campos respectivos y subrayando la importancia de las claves primarias y foráneas que garantizan la integridad y la coherencia de los datos almacenados. Además, hemos destacado las relaciones que establecen vínculos entre estas tablas, permitiendo una comprensión completa de cómo los distintos conjuntos de datos interactúan y se relacionan entre sí.

Esta estructuración es fundamental para tener una base sólida para futuras optimizaciones y para el mantenimiento continuo de una base de datos eficiente y confiable en el entorno de nuestro sistema.



Se han implementado pruebas automatizadas de integración para la aplicación 'Dental Hub' realizada con Filament y Livewire, utilizando PestPHP como framework de pruebas. Estas pruebas cubren componentes críticos del sistema relacionados con la administración de modelos clave como citas, sucursales, clínicas, pacientes y usuarios. Todas las pruebas requieren autenticación previa de un usuario y utilizan una base de datos en memoria SQLite para aislar el entorno de pruebas.



Grupos de pruebas

Las pruebas se han dividido en los siguientes grupos:

- **AppointmentsTest:** Validaciones relacionadas con la administración de citas.
- **BranchesTest:** Pruebas para la gestión de sucursales.
- **ClinicTest:** Verificación de la funcionalidad de las clínicas.
- **PatientsTest:** Pruebas enfocadas en la administración de pacientes.
- **UsersTest:** Validaciones de gestión de usuarios.

Cobertura de pruebas

Las pruebas automatizadas se han centrado en los siguientes escenarios para cada grupo de pruebas:

1. Renderizado del Index (Listado):

- Se probó que las vistas del listado de cada recurso se rendericen correctamente y que la tabla de datos cargue de manera adecuada.

2. Visualización Detallada (Show):

- Se verificó que la vista individual de un modelo específico cargue correctamente, mostrando la información detallada del recurso.

3. Creación y Validación de Registros (Create):

- Se implementaron pruebas para asegurar que los datos se validen correctamente al crear un nuevo registro. Se validó la creación exitosa y la correcta inserción de datos en la base de datos.

4. Renderizado y Edición (Edit):

- Se probó que la página de edición de un recurso existente se renderice correctamente y que los datos se muestren en el formulario de edición.

5. Actualización de Registros (Update):

- Se asegura que la página de edición tenga acceso al recurso existente y que pueda realizar los cambios correctamente en la base de datos.

6. Soft-Delete y Restauración:

- Se verificó que la funcionalidad de 'soft-delete' funcione correctamente, permitiendo que los registros sean eliminados temporalmente y posteriormente restaurados.

Configuración del Entorno de Pruebas

- **Framework de Pruebas:** PestPHP
- **Autenticación:** Todas las pruebas requieren que un usuario esté autenticado. Se utiliza un usuario predefinido para simular la autenticación.
- **Base de Datos:** Las pruebas utilizan una base de datos SQLite en memoria para garantizar la aislación de los datos y evitar interferencias con otras pruebas o datos de producción.

Aunque el entorno de producción utiliza **PostgreSQL** como sistema gestor de base de datos, las pruebas automatizadas se ejecutan con **SQLite**. Esto es posible gracias al **ORM Eloquent** de Laravel, que abstrae las interacciones con la base de datos, asegurando que mientras no se utilicen RAW SQL queries, haya completa compatibilidad y consistencia en el funcionamiento entre diferentes gestores de bases de datos. De esta manera, las pruebas realizadas en SQLite son representativas y equivalentes a cómo se comportaría el sistema en un entorno de producción con PostgreSQL.

Resultados Obtenidos

Todas las pruebas ejecutadas han pasado satisfactoriamente, con las siguientes consideraciones:

- **Autenticación:** En caso de que la autenticación falle (por ejemplo, si un usuario no está autenticado), las pruebas relacionadas con la creación, edición, o eliminación de modelos fallarán debido a la falta de permisos.
- **Cambios en los Modelos:** Si se realizan cambios en los campos de los modelos, es probable que las pruebas fallen hasta que se ajusten los tests correspondientes a los nuevos esquemas o validaciones.

En general, los resultados de las pruebas indican que las funcionalidades críticas del sistema están correctamente implementadas, y que los datos son gestionados de manera segura y precisa en la base de datos.