



**Departamento Computación**  
**Licenciatura en Ciencia de la Computación**

**TRABAJO DE DIPLOMA**

# **SISTEMA DE INFORMACIÓN PARA EL REGISTRO CENTRAL COMERCIAL**

**Autor:** César Valdés Iglesias

**Tutores:** Ing. Jandro Daniel Velázquez Hernández

Dr. C. Abel Rodríguez Morffi

Santa Clara, Cuba, noviembre de 2021  
Copyright©UCLV



**Department of Computer Science**

**Bachelor in Computer Science**

**DIPLOMA THESIS**

# **INFORMATION SYSTEM FOR THE CENTRAL COMMERCIAL REGISTRY**

**Author:** César Valdés Iglesias

**Tutors:** Ing. Jandro Daniel Velázquez Hernández

Dr. C. Abel Rodríguez Morffi

Santa Clara, Cuba, November 2021  
Copyright©UCLV

Este documento es Propiedad Patrimonial de la Universidad Central “Marta Abreu” de Las Villas, y se encuentra depositado en los fondos de la Biblioteca Universitaria “Chiqui Gómez Lubian” subordinada a la Dirección de Información Científico Técnica de la mencionada casa de altos estudios.

Se autoriza su utilización bajo la licencia siguiente:

**Atribución- No Comercial- Compartir Igual**



Para cualquier información contacte con:

Dirección de Información Científico Técnica. Universidad Central “Marta Abreu” de Las Villas.

Carretera a Camajuaní. Km 5½. Santa Clara. Villa Clara. Cuba. CP. 54 830

Teléfonos.: +53 01 42281503-14190



Hago constar que el presente trabajo fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

---

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

---

Firma del tutor

---

Firma del tutor

---

Firma del jefe del Dpto

*“Para mí la programación es más que un importante arte práctico. También es un desafío gigantesco en los fundamentos del conocimiento.”*

**Grace Murray Hopper**

*“Sólo hay dos tipos de lenguajes: aquellos de los que la gente se queja y aquellos que nadie usa.”*

**Bjarne Stroustrup**

*“Cualquier tonto puede escribir código que un ordenador entiende. Los buenos programadores escriben código que los humanos pueden entender.”*

**Martin Fowler**

*A mi madre Mileidi Iglesias Cuellar quien me apoyó siempre en mi carrera como profesional, quien ayudó a seguir mi sueño de convertirme en un buen licenciado en Ciencias de la Computación. Por toda su dedicación, esfuerzo y apoyo incondicional durante toda mi carrera, por estar siempre presente en las buenas y en las malas y ser una madre maravillosa.*

*A mi padre Irán por su apoyo dado para que culminara mi carrera como licenciado*

*A mi abuela Teresa que con su forma estricta de ser apoyó mi carrera haciendo que no me saliera del buen camino.*

*A mi hermana Daniela y mi tío Joel por darme ánimos y aliento durante el proceso de confección del trabajo de diploma.*

*A mis compañeros de carrera que aun teniendo nuestras dudas acerca de la confección del trabajo de diploma nunca dejamos de apoyarnos para salir juntos adelante.*

*A todos los profesores de mi facultad que me impartieron docencia y contribuyeron a mi formación como profesional*

## **RESUMEN**

El Registro Central Comercial es una entidad subordinada al Ministerio del Comercio Interior que se encarga de la inscripción de establecimientos estatales o privados de comercio del mercado minorista y mayorista interno. Bienestar RCC es una nueva aplicación Web desarrollada para esta entidad para la gestión de los trámites registrales en línea, una solución alternativa a una aplicación existente en la actualidad SISREG, la cual se encuentra instalada en cada una de las sedes provinciales del RCC y en la sede nacional, donde, a través de la exportación e importación de ficheros, se realiza la consolidación de los datos. Con el fin de realizar el análisis de la información generada por el sistema Bienestar RCC se decidió la creación de un sistema informático que sirva como una herramienta de trabajo para mantener informados a los niveles superiores y favorezca la realización de análisis estadísticos de la red comercial y estudios geoespaciales.

Este sistema muestra el análisis de los datos a través de gráficas interactivas, lo que posibilita la toma de decisiones. Utiliza una interfaz de programación de aplicaciones (API) para proveer los servicios accedidos desde un navegador Web y puede también ser utilizada para compartir información hacia dispositivos móviles. Esta API implementa el estilo arquitectónico REST y se implementó utilizando PHP como lenguaje de programación mediante el marco de trabajo Laravel.

**Palabras Clave:** análisis de datos, estadísticas, toma de decisiones, interfaz de programación de aplicaciones, REST.



## **ABSTRACT**

The Central Commercial Registry is an entity subordinate to the Ministry of Internal Commerce that is responsible for the registration of state or private commercial establishments in the domestic retail and wholesale market. Bienestar RCC is a new Web application developed for this entity for the management of registration procedures online, an alternative solution to an application that currently exists SISREG, which is installed in each of the provincial headquarters of the RCC and in the national headquarters, where, through the export and import of files, the data is consolidated. In order to perform the analysis of the information generated by the Bienestar RCC system, it was decided to create a computer system that serves as a work tool to keep the higher levels informed and favors the performance of statistical analysis of the commercial network and geospatial studies.

This system shows the analysis of the data through interactive graphics, which enables decision-making. It uses an application programming interface (API) to provide the services accessed from a Web browser and can also be used to share information to mobile devices. This API implements the REST architectural style and was implemented using PHP as the programming language using the Laravel framework.

**Keywords:** data analysis, statistics, decision making, application programming interface, REST.

## TABLA DE CONTENIDOS

INTRODUCCIÓN .....	1
CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.....	4
1.1. <i>Desarrollo basado en API</i> .....	4
1.1.1. API REST .....	5
1.1.2. Características de REST .....	7
1.1.3. Ventajas que ofrece REST para el desarrollo .....	8
1.2. <i>Framework Laravel</i> .....	9
1.2.1. Características de Laravel .....	10
1.2.2. Ecosistema de Laravel .....	12
1.3. <i>Framework Vue.js</i> .....	14
1.3.1. ¿Qué es Vue.js?.....	15
1.3.2. ¿Por qué elegir Vue.js? .....	15
1.3.3. ¿Cuándo usar Vue.js? .....	17
1.4. <i>ApexCharts</i> .....	17
1.5. <i>Conclusiones parciales del capítulo.</i> .....	18
CAPÍTULO 2: CONCEPCIÓN DEL SISTEMA PARA EL ANÁLISIS DE LA INFORMACIÓN GENERADA POR LA APLICACIÓN BIENESTAR RCC.....	19
2.1. <i>Arquitectura del sistema</i> .....	19
2.2. <i>Descripción de la fuente de datos</i> .....	20
2.3. <i>Configuración del Ambiente de Desarrollo</i> .....	23
2.4. <i>Diagrama de Clases UML</i> .....	23
2.5. <i>Implementación del API REST</i> .....	32
2.5.1 Creación de las Rutas de la API RESTful .....	32
2.5.2 Diseño e implementación de la clase controladora.....	36
2.5.3 WSO2 y seguridad de la API.....	43
2.6. <i>Conclusiones parciales del capítulo.</i> .....	47
CAPÍTULO 3 IMPLEMENTACIÓN DEL SISTEMA PARA EL ANÁLISIS DE LA INFORMACIÓN GENERADA POR LA APLICACIÓN BIENESTAR RCC.....	48

3.1.	<i>Herramientas utilizadas en el ambiente de desarrollo.....</i>	48
3.2.	<i>Estructura del proyecto .....</i>	49
3.3.	<i>Implementación de los ficheros vue.....</i>	52
3.4.	<i>Interfaz de usuario .....</i>	55
3.4.1.	<i>Autenticación .....</i>	55
3.4.2.	<i>Pantalla principal del sistema .....</i>	56
3.5.	<i>Conclusiones parciales del capítulo.....</i>	60
CONCLUSIONES .....		61
RECOMENDACIONES.....		62
REFERENCIAS BIBLIOGRÁFICAS .....		63

## LISTA DE FIGURAS

Figura 1: Arquitectura del sistema.....	19
Figura 2: Diagrama de recursos UML para la sección de los nomencladores.....	24
Figura 3: Diagrama de recursos UML para la sección de análisis de datos .....	27
Figura 4: Diagrama recursos UML sección clientes.....	30
Figura 5: Archivo Kernel.php.....	33
Figura 6: Estructura del proyecto para desarrollo de la API, definición de rutas.....	34
Figura 7: Fichero de rutas api.php .....	35
Figura 8: Listado de rutas de la API. ....	36
Figura 9: Diagrama de clases: clase controladora .....	37
Figura 10: Estructura del proyecto, Carpeta app/Http/Controllers .....	39
Figura 11: Implementación de la clase NegocioController.php .....	40
Figura 12: Implementación de la clase getGraphActividades .....	41
Figura 13: Ejemplo de petición al recurso <i>graph-act</i> .....	41
Figura 14: Ejemplo de respuesta del recurso <i>graph-act</i> .....	42
Figura 15: WSO2 - diagrama del API gateway .....	43
Figura 16: Función generar token .....	44
Figura 17: Ejemplo de respuesta de token válido .....	45
Figura 18: Función para obtener los datos del usuario autenticado.....	46
Figura 19: Función para listar los usuarios del sistema .....	46
Figura 20: Estructura del proyecto Vue.js .....	49
Figura 21: Estructura de la carpeta /src .....	50
Figura 22: Contenido de la carpeta components.....	51
Figura 23: Estructura de la carpeta views .....	52
Figura 24: Estructura de un archivo .vue .....	53

Figura 25: Implementación del archivo auth.js .....	55
Figura 26: Pantalla de Inicio de sesión .....	56
Figura 27: Vista principal de la sección del tablero de control .....	56
Figura 28: Panel de filtros.....	57
Figura 29: Sección del comercio minorista .....	58
Figura 30: Clasificación de las instalaciones por particularidades .....	58
Figura 31: Listado de los usuarios asociados a los clientes (entidades o UEB) .....	59
Figura 32: Tablero de control del proceso registral de la red comercial .....	60

## **LISTA DE TABLAS**

Tabla 1: Lista de Recursos de la sección de los nomencladores .....	24
Tabla 2: Respuestas de los recursos de la sección de los nomencladores .....	25
Tabla 3: Lista de recursos para la sección de análisis de datos .....	27
Tabla 4: Respuestas de los recursos para la sección de análisis de datos .....	29
Tabla 5: Lista de recursos para la sección clientes .....	30
Tabla 6: Respuestas de los recursos de la sección clientes .....	31

## **INTRODUCCIÓN**

La necesidad de los gobiernos por agilizar, optimizar, flexibilizar, transparentar y abaratar procesos y/o actividades del sistema público, ha motivado a utilizar en forma acelerada y sustancial las tecnologías de información y comunicación (TIC) para el desarrollo de aplicaciones cada vez más complejas, indefectiblemente apoyadas por arquitecturas dedicadas, especialmente diseñadas para trabajar de la manera más óptima, integrando sistemas, utilizando las mejores herramientas de gestión y desarrollando modelos adecuados a las necesidades de Gobierno, creando plataformas compatibles que resuelven temas como la interoperabilidad, compatibilidad, acceso, seguridad, entre otras (Palvia, Palvia y Sharma, 2007).

El Gobierno Electrónico (GE) es la transformación de todo el gobierno como un cambio de paradigma en la gestión gubernamental, es un concepto de gestión que fusiona la utilización intensiva de las TIC, con modalidades de gestión, planificación y administración, como una nueva forma de gobierno. Bajo este punto de vista, el GE basa y fundamenta su aplicación en la Administración Pública, teniendo como objetivo contribuir al uso de las TIC para mejorar los servicios e información ofrecida a los ciudadanos y organizaciones, mejorar y simplificar los procesos de soporte institucional y facilitar la creación de canales que permitan aumentar la transparencia y la participación ciudadana (Toledo, 2016).

Dentro de la gestión gubernamental, el Registro Central Comercial (RCC) es una entidad subordinada al Ministerio del Comercio Interior que se encarga de la inscripción de aquellos establecimientos que, en el mercado interno, realizan actividades de comercio mayorista, venta minorista de mercancías, gastronomía, servicios personales, técnicos comerciales, de uso doméstico, de alojamiento y de recreación, ya sea del sector estatal, cooperativo, mixto o privado. Todas aquellas personas que ejercen operaciones vinculadas al comercio, ya sea de carácter minorista o mayorista, estatal o privado, están en la obligación de inscribirse en el Registro Central Comercial.

Para el procesamiento de la información los especialistas del RCC utilizan la aplicación de escritorio SISREG. Dicha aplicación se encuentra instalada en cada una de las sedes provinciales del RCC y en la sede nacional, donde, a través de la exportación e importación de ficheros, se realiza la consolidación de los datos. Dentro de cualquier entidad cobra gran importancia

mantener informado a los niveles superiores de lo acontecido dentro de ella; con esta información se facilita la toma de decisiones. Llevar a cabo tal objetivo es una tarea difícil cuando se solicita alguna información al RCC por no existir una herramienta que brinde toda la información que se solicita. En estos casos, el especialista del RCC exporta los informes generados por el sistema SISREG a formato Excel, realiza el análisis de los datos y envía el fichero resultante a la entidad que lo solicita, obteniendo resultados ineficientes y demora en la toma de decisiones por contar con información desactualizada.

Bienestar RCC es una nueva aplicación Web desarrollada en el Registro Central Comercial para los trámites registrales en línea, que constituye una nueva versión de SISREG; esta herramienta informática ofrece múltiples bondades al usuario, además de incorporar innovación al Ministerio del Comercio Interior, alineado con lo establecido en la Estrategia Económico Social del país; sin embargo, este no brinda toda la información que necesitan los organismos para la toma de decisiones a nivel gubernamental.

De lo expuesto anteriormente se puede identificar el siguiente problema: Se necesita acceder a la información generada por la aplicación Bienestar RCC desde aplicaciones implementadas en cualquier lenguaje de programación, que sirvan como una herramienta de trabajo para mantener informados a los niveles superiores y favorezcan la realización de análisis estadísticos de la red comercial y estudios geoespaciales.

Esta información constituirá un servicio añadido a las entidades interesadas en realizar procesos inversionistas y que necesitan conocer el mercado y una caracterización de los establecimientos afines a la actividad en que incursionarán. Los análisis deberán proporcionar el conocimiento de forma inmediata de cuáles son las zonas poblacionales que no tienen una cobertura suficiente en cuanto a la gastronomía popular, venta de productos normados o de demanda diaria, y hacerles llegar esa información a las direcciones estatales de comercio.

Por todo lo anterior, el **objetivo** de este trabajo es desarrollar un sistema informático para el análisis de la información generada por el sistema automatizado Bienestar RCC que utilice una interfaz de programación de aplicaciones para proveer los servicios accedidos desde un navegador Web y que pueda usarse para compartir información hacia dispositivos móviles.

De esta manera, se definen los **objetivos específicos** siguientes:



1. Determinar el tipo de Interfaz de Programación de Aplicaciones (API) conveniente para comunicar la información almacenada en el sistema Bienestar RCC con aplicaciones implementadas en cualquier lenguaje de programación.
2. Realizar un estudio sobre la utilización de marcos de trabajo para el desarrollo de la API y de la aplicación Web para el análisis de los datos.
3. Definir la arquitectura de la aplicación para el análisis de la información generada por el sistema Bienestar RCC.
4. Implementar una API para la comunicación entre el software Bienestar RCC y la aplicación Web.
5. Implementar la aplicación Web utilizando los marcos de trabajo de desarrollo seleccionados.

Este documento se ha estructurado en Introducción, tres capítulos, Conclusiones, Recomendaciones y Referencias bibliográficas. En el primer capítulo se hace un estudio de los referentes teóricos que sustentan el desarrollo del trabajo, especificando el uso de los marcos de trabajo utilizados en la aplicación, las razones de la selección, ventajas y características esenciales para el desarrollo de APIs, así como cualidades e importancia del uso de estas para el desarrollo de sistemas informáticos robustos.

En el segundo capítulo se describen la arquitectura del sistema propuesto, los conceptos importantes para la construcción del software, se define el ambiente de desarrollo, diseño e implementación de la API, donde se incluye la definición e implementación de los recursos disponibles a consumir por las diferentes aplicaciones y la descripción y documentación de cada funcionalidad.

En el tercer capítulo se detallan las herramientas usadas y las interioridades de la programación para la implementación del sistema informático, se describe la estructura del proyecto, así como la estructura interna de los archivos según el framework VueJS, además se definen los plugins utilizados y la interfaz de usuario.

## CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se describen algunos de los principales conceptos, métodos, tecnologías y aplicaciones existentes en la actualidad, relacionados con la posible solución del problema de investigación planteado. Se llega a conclusiones sobre los métodos, tecnologías y software a desarrollar.

### *1.1. Desarrollo basado en API*

El desarrollo de aplicaciones avanza constantemente y presenta novedades prácticamente desde que nació y lo hace a una gran velocidad. Los desarrolladores están saturados ante opciones nuevas, lenguajes, arquitecturas, bibliotecas o frameworks (Pantoja y Pardo, 2016). Ante ese panorama de continuo avance, se encuentran corrientes que, por sus ventajas y versatilidad, están adquiriendo cada vez un mayor protagonismo, como es el caso del desarrollo basado en API (Jin, Sahni y Shevat, 2018).

Desarrollo backend tradicional:

El desarrollo del lado del servidor tradicional se basaba en construir sistemas que devolvieran al cliente código HTML (Musciano y Kennedy, 2000), capaz de ser interpretado directamente por el navegador. De esta manera, cuando se programaba en PHP (Group, 2016), Python (Luca, 2019), .NET (Schmelzer, 2019), etc. lo normal era que se entregase al navegador todo lo necesario para «pintar» la página a su usuario. El punto débil de esta alternativa se basa principalmente en:

- Hoy no se consume el servicio web solo a través de Internet, sino también por medio de aplicaciones para móviles, etc. Si se produce HTML desde el lado del backend, es muy probable que se tenga que programar varias veces ese backend para cada sistema al que se lo quiere aportar.
- Ante la cantidad de alternativas que aparecen con tanta velocidad, la parte que menos cambia es la del backend, por lo que es interesante que se pueda desarrollar esa capa de manera que se consiga adaptar a cualquier tipo de biblioteca del lado del frontend.

Cuando se desarrolla un servicio backend construyendo una API, se asegura que se pueda consumir desde cualquier sistema o cliente. El API no devuelve más que datos, que están

desacoplados a cualquier modo de visualización. Si se está implementando una web, se consume el API desde cualquiera de los frameworks o bibliotecas Javascript populares, como AngularJS, Polymer o ReactJS. También se podrán consumir los servicios del API desde aplicaciones desarrolladas en Java para Android o Swift/Objective C para iOS, por ejemplo.

Una interfaz de programación de aplicaciones (API) es una conexión entre computadoras o entre programas de computadora. Es un tipo de interfaz de software que ofrece un servicio a otras piezas de software (Reddy, 2011). Un documento o estándar que describe cómo construir o usar dicha conexión o interfaz se denomina especificación de API. Se dice que un sistema informático que cumple con este estándar implementa o expone una API.

El término API puede referirse a la especificación o a la implementación. A diferencia de una interfaz de usuario, que conecta una computadora a una persona, una API conecta computadoras o piezas de software entre sí. No está destinado a ser utilizado directamente por una persona (el usuario final) que no sea un programador informático que lo está incorporando en el software. Una API a menudo se compone de diferentes partes que actúan como herramientas o servicios que están disponibles para el programador. Se dice que, un programa o un programador que usa una de estas partes, llama a esa parte de la API. Las llamadas que componen la API también se conocen como subrutinas, métodos, solicitudes o endpoints. Una especificación de API define estas llamadas, lo que significa que explica cómo usarlas o implementarlas.

Al crear aplicaciones, una API (interfaz de programación de aplicaciones) simplifica la programación al abstraer la implementación subyacente y exponer solo los objetos o acciones que el desarrollador necesita. Si bien una interfaz gráfica para un cliente de correo electrónico puede proporcionar al usuario un botón que realiza todos los pasos para buscar y resaltar nuevos correos electrónicos, una API para la entrada / salida de archivos puede brindarle al desarrollador una función que copie un archivo de una ubicación a otra sin requerir que el desarrollador comprenda las operaciones del sistema de archivos que ocurren detrás de escena (Clarke, 2004).

#### 1.1.1. API REST

Dentro de las alternativas de construcción de un API, REST es una que simplifica mucho el funcionamiento, dado que se elimina todo lo relacionado al estado de la aplicación.

En primer lugar, se debe de entender qué se considera exactamente una API RESTful. REST significa Representational State Transfer y es un estilo arquitectónico para la comunicación de red entre aplicaciones, que se basa en un protocolo sin estado (normalmente HTTP) para la interacción (De Souza, 2020).

En las API RESTful, se usan los verbos HTTP como acciones y los extremos son los recursos sobre los que se actúa. Se usan los verbos HTTP por su significado semántico:

- GET: recuperar recursos
- POST: crear recursos
- PUT: actualizar recursos
- DELETE: eliminar recursos

En el ecosistema backend, la mayoría de los lenguajes nos permiten construir API REST de manera fácil, mediante diversas bibliotecas o frameworks.

- PHP. Existen frameworks potentes como Symfony o Laravel capaces de acelerar mucho el desarrollo de un API. Además, hay muchos microframeworks que tienen como principal objetivo el desarrollo de API REST, desechando todo lo pesado y no tan útil de un framework convencional, como Lumen o Slim.
- NodeJS. El desarrollo con NodeJS de un API REST es especialmente indicado, dado que permite construir el backend con el mismo lenguaje con el que se construye el frontend, evitando el esfuerzo de pasar de un lenguaje a otro cuando se desarrollan ambas capas de un proyecto web. Además, dadas las características de esta plataforma, también se hace muy adecuada para el desarrollo RESTful dado que Node, al no ser bloqueante, puede atender a más usuarios de manera concurrente. En NodeJS, hay frameworks potentes para el desarrollo de APIs y aplicaciones en general, como Express o algunos enfocados al desarrollo de APIs de manera más particular como SailsJS.
- .NET. En esta tecnología también existen diversas alternativas sencillas y rápidas para implementar un API. El propio Microsoft ofrece ASP.NET Web API, un complemento esencial para construir servicios HTTP especialmente pensados para REST.

REST cambió por completo la ingeniería de software a partir del 2000. Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por Roy Fielding, el padre de la especificación HTTP y uno de los referentes internacionales en todo lo relacionado con la Arquitectura de Redes. En el campo de las APIs, REST es, a día de hoy, la clave del desarrollo de servicios de aplicaciones (Castillo *et al.*, 2011).

En la actualidad la mayoría de los proyectos o aplicaciones disponen de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook, etc. entre otras cientos de empresas generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet. REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST (Liew, 2018).

#### 1.1.2. Características de REST

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita

acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

- Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
- Uso de hipermedios: hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.
- Para cualquier API REST es obligatorio disponer del principio HATEOAS (Hypermedia As The Engine Of Application State – Hipermedia Como Motor del Estado de la Aplicación) para ser una verdadera API REST. Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente (Alonso, 2015).

#### 1.1.3. Ventajas que ofrece REST para el desarrollo

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- Visibilidad, fiabilidad y escalabilidad: La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin

excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles para trabajar.

- La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Con el auge del desarrollo de aplicaciones para dispositivos móviles y los marcos de JavaScript, el uso de una API RESTful es una de las mejores opciones para crear una única interfaz entre sus datos y su cliente (Miguel y Hernández, 2021).

### *1.2. Framework Laravel*

Laravel es uno de los frameworks más populares de código abierto para PHP que permite desarrollar aplicaciones y servicios web con PHP. Fue creado en el año 2011 y su filosofía es la de desarrollar código PHP de forma elegante y simple para crear código de forma sencilla y permitiendo múltiples funcionalidades. Laravel es tal vez el framework de PHP más usado en la actualidad debido a la facilidad de uso que proporciona y la versatilidad para desarrollar no sólo aplicaciones sencillas sino grandes proyectos lo suficientemente robustos para cubrir necesidades muy específicas sin dejar de lado la calidad. Una de las ventajas de este framework es que puede ser utilizado para construir APIs gracias a todos los componentes que implementa. Como framework resulta bastante moderno y ofrece muchas utilidades potentes a los desarrolladores, que permiten agilizar el desarrollo de las aplicaciones web. Laravel pone énfasis en la calidad del código, la facilidad de mantenimiento y escalabilidad, lo que permite realizar proyectos desde pequeños a grandes o muy grandes. Además, permite y facilita el trabajo en equipo y promueve las mejores prácticas (Chen *et al.*, 2017).

### 1.2.1. Características de Laravel

El framework Laravel trabaja con una arquitectura avanzada de carpetas, de modo que promueve la separación de los archivos con un orden correcto y definido, que guiará a todos los integrantes del equipo de trabajo y será un estándar a lo largo de los distintos proyectos. Por supuesto, dispone también de una arquitectura de clases también muy adecuada, que promueve la separación del código por responsabilidades. Su estilo arquitectónico es MVC (Laravel, 2015). Contiene además un amplio conjunto de características, que sirven para realizar la mayoría de las aplicaciones web. Entre ellas se encuentran:

- Un sistema de rutas, mediante las cuales es fácil crear y mantener todo tipo de URLs amistosas a usuarios y buscadores, rutas de API, etc.
- Un sistema de abstracción de base de datos, con un ORM potente pero sencillo de manejar, mediante el que podemos tratar los datos de la base de datos como si fueran simples objetos.
- Un sistema para creación de colas de trabajo, de modo que es posible enviar tareas para ejecución en background y aumentar el rendimiento de las aplicaciones.
- Varias configuraciones para envío de email, con proveedores diversos.
- Un sistema de notificaciones a usuarios, mediante email, base de datos y otros canales.
- Una abstracción del sistema de archivos, mediante el cual podemos escribir datos en proveedores cloud, y por supuesto en el disco del servidor, con el mismo código.
- Gestión de sesiones.
- Sistema de autenticación, con todo lo necesario como recordatorios de clave, confirmación de cuentas, recordar un usuario que ha iniciado sesión (log), etc.
- La posibilidad de acceder a datos en tiempo real y recibir notificaciones cuando éstos acceden a la base de datos.

Laravel se ha consolidado en los últimos años como el framework PHP más utilizado. Esto se debe a varios factores. Obviamente, es fundamental la cantidad de funcionalidades ya listas que ofrece, sin embargo, este motivo no es el más importante, dado que muchos otros frameworks



ofrecen un soporte también muy completo. Sí es verdad que la cantidad de módulos ha ido incrementándose continuamente y a día de hoy es posible que supere a otras alternativas populares, pero su adopción por parte de la comunidad ha sido muy relevante incluso antes de ello.

Facilidad de uso: Desde el conocimiento de varios frameworks populares, más o menos complejos, podemos decir que Laravel es uno de los más sencillos de utilizar. Existen frameworks tan potentes como Laravel, pero la curva de aprendizaje es bastante más severa. Esto ha posibilitado que desarrolladores que ya venían utilizando alternativas más sencillas para el desarrollo de aplicaciones y deseaban pasar a otros frameworks más completos, han podido acercarse con éxito a Laravel, sin demasiadas frustraciones por tener que aprender cosas nuevas.

Artisan: Laravel dispone de un potente sistema de comandos de consola para resumir muchas tareas tediosas y repetitivas. Artisan, la herramienta de línea de comandos de Laravel ha conseguido que la experiencia de desarrollo con Laravel sea muy atractiva para las personas que se han decidido por este framework.

Documentación y tutoriales: La documentación de Laravel siempre ha sido muy completa, además tiene un enfoque bastante didáctico, por lo que resulta muy sencillo de usar como base para el aprendizaje y no solamente la referencia. Además, Laravel ha venido con una serie de proyectos paralelos de tutoriales adicionales a la documentación que han ayudado mucho a su divulgación y la creación de una activa comunidad.

Solidez: Desde las versiones iniciales ha demostrado una gran solidez, lo que ha permitido que su crecimiento no rompiese de manera radical los proyectos que necesitaban ser actualizados. Esto ayuda bastante a que sus seguidores sigan confiando y apoyando las herramientas. A la vez, Laravel ha conseguido adaptarse desde el inicio con toda una serie de herramientas del mundo del desarrollo actual, como los proveedores de cloud para almacenamiento de archivos, proveedores de pagos electrónicos, sistemas de mensajería y email, etc.

Apoyarse en otros productos sin inventar la rueda: Laravel no ha necesitado reinventar el mundo del desarrollo. En cambio, ha sabido aprovechar mucha de la funcionalidad de productos ya consolidados. El ejemplo más importante es el framework Symfony, del que se ha extraído

cantidad de funcionalidad que ya estaba muy madura, adoptándola para hacer posible un crecimiento rápido, estable y seguro.

Lo cierto es que cualquier producto que viene desde atrás en el tiempo tiene muchas más posibilidades de llegar a una fórmula ganadora. Laravel ha conseguido tomar ventaja de esta situación y llegar a escalar lo suficiente para convertirse en el framework PHP más popular.

### 1.2.2. Ecosistema de Laravel

Una de las claves del éxito de Laravel es que se ha convertido en una industria de generación de servicios relacionados con el framework. Esto ha permitido que las personas cercanas a su núcleo, Taylor Otwell a la cabeza, tengan una base sólida para financiar el desarrollo de Laravel, agregar funcionalidad y seguir innovando con la creación de nuevas herramientas que hagan la vida más sencilla a los desarrolladores (Denzel Javier Ovando Ortega, 2019).

Algunos de los servicios y herramientas para el framework Laravel más populares y útiles son los siguientes:

- Forge: es un sistema de administración de servidores, como una especie de panel con el que puedes crear dominios, instalar Laravel, desplegar aplicaciones, etc. Pone a los desarrolladores toda una serie de herramientas para que instalar aplicaciones Laravel en producción sea muy sencillo.
- Vapor: permite desplegar aplicaciones Laravel sin necesidad de tener un servidor. Es decir, es un backend "serverless". Al no tener servidor se ahorra la necesidad de administrar el propio sistema, lo que teóricamente quita trabajo y aumenta la seguridad.
- Nova: es un sistema de gestión de contenido que permite crear los típicos paneles de administración de una manera muy cómoda. Es muy versátil y ahorra tiempo de programación que muchas veces es tedioso y repetitivo.
- Envoyer: Permite integración continua de las aplicaciones Laravel, simplemente haciendo un push al repositorio la aplicación se despliega en su nueva versión, sin interrumpir el servicio.
- Spark: Permite crear aplicaciones con una base de código encima del propio Laravel, evitando muchas partes que se pueden repetir en muchos de los proyectos.

Los anteriores son proyectos que, de alguna u otra manera, tienen cierta parte de pago y que sirven para financiar la maquinaria en torno de Laravel. Pero además hay muchos otros servicios que forman parte del ecosistema de Laravel que son gratuitos o están incluidos entre las funcionalidades del framework:

- Homestead: Permite máquinas virtuales para facilitar el desarrollo de Laravel en las condiciones más profesionales.
- Valet: Permite, en Mac, crear entornos de ejecución de Laravel para desarrollo muy ligeros y fáciles de configurar mediante comandos de consola.
- Cashier: Para pagos electrónicos y suscripciones.
- Scout: un sistema de búsqueda por texto.
- Passport: Un sistema de autenticación OAuth2.
- Mix: una envoltura de Webpack para compilar assets y realizar tareas del front, sin las complejidades de administrar un entorno directamente con Webpack.
- Echo: que permite trabajar con datos en tiempo real.
- Horizon: para la monitorización de colas de procesos.

Desarrollar aplicaciones usando Laravel es muy sencillo, fundamentalmente debido a su sintaxis expresiva, sus generadores de código, y su ORM incluido de paquete llamado Eloquent ORM. Laravel, propone una forma de desarrollar aplicaciones web de un modo mucho más ágil. Por ejemplo, en Laravel opcionalmente se puede usar el patrón de diseño MVC (Modelo-Vista-Controlador) tradicional, donde al igual que otros framework PHP, el controlador es programado como una clase (Laravel, 2015). Por lo tanto, un Controlador es una clase PHP que dispone de métodos públicos que son el punto de entrada final de una petición HTTP (Request PHP) a nuestra aplicación. Laravel da la opción de seguir usando la metodología tradicional MVC. Sin embargo, el framework propone una vía más rápida en PHP, la cual consiste en programar la interacción HTTP directamente como una función anónima asociada a una Ruta. Esto tiene la ventaja de reducir la cantidad de código, especialmente cuando sólo se necesita incluir una funcionalidad. Así, donde antes se requería programar una clase, ahora en Laravel sólo se necesita escribir una función en PHP.

API REST en Laravel es la implementación de una interfaz desarrollada en uno de los Frameworks PHP más utilizados para el desarrollo de aplicaciones. El framework también tiene como objetivo evolucionar con la web y ya ha incorporado varias características e ideas nuevas en el mundo del desarrollo web, como colas de trabajos, autenticación de API lista para usar, comunicación en tiempo real y mucho más.

El mundo del desarrollo web ha cambiado mucho en la última década y, por supuesto, la experiencia de uso de las aplicaciones, que hoy son mucho más dinámicas, rápidas e interactivas. Para conseguirlo, muchas de las funcionalidades que antes se desarrollaban con código del lado del servidor, lo que se conoce como «backend», se han llevado a código del lado del cliente con Javascript, llamado «frontend».

Por supuesto, en la medida que más y más código Javascript ha de ejecutarse en el lado del cliente, ha aumentado considerablemente la complejidad de las aplicaciones del lado del cliente. Es por ello que los desarrolladores han comenzado a apoyarse en lo que se conoce como los frameworks frontend. Entre los más populares actualmente tenemos a Angular, React y VueJS.

### *1.3. Framework Vue.js*

Vue.js se anuncia como un «framework progresivo». Esta calificación indica que en realidad con Vue se puede crear todo tipo de desarrollos. Podrían ser componentes sencillos, que implementan una parte determinada de una aplicación web, pero también aplicaciones frontend completas, con su sistema de routing y cantidad de lógica de negocio.

Con VueJS se implementa lo que se conoce como arquitectura de componentes. Este permite dividir las aplicaciones en bloques con funcionalidades independientes, llamados componentes. Esos bloques podrían ser una cabecera, un menú, un listado, una ficha de producto, etc. En realidad, cualquier cosa que se pueda necesitar puede ser un componente. Además, unos componentes se pueden apoyar en otros, de modo que en un listado de productos se puedan tener fichas de productos, que a su vez pueden estar compuestas por datos, botones, desplegables con información, etc. (Nelson, 2018).

Gracias a la arquitectura de componentes se pueden construir aplicaciones a base de piezas reutilizables, de modo que, al final, las aplicaciones no son más que árboles de componentes que trabajan entre sí para implementar funcionalidades tan complejas como sea necesario.

#### 1.3.1. ¿Qué es Vue.js?

Vue.js es un framework progresivo para construir interfaces de usuario. Es una alternativa a frameworks como Angular o React, y está siendo extremadamente popular en China, país de nacimiento de su creador Evan You, que trabajó para Google y para Meteor.

Su paso por Google ha influenciado el desarrollo del framework y toma las mejores herramientas de cada framework. Por ejemplo, para construir este framework ha empleado la potencia de las directivas de Angular y ha implementado el DOM virtual de React. Pero, sobre todo, lo que toda la comunidad está de acuerdo del framework es que es una herramienta “Developer Friendly”, y esta es una de las claves de su popularidad (Www.w3schools.com, 2020) .

#### 1.3.2. ¿Por qué elegir Vue.js?

En lo que respecta a los frameworks frontend existe panorama dominado por grandes organizaciones. Angular está apoyado por Google, mientras que React es creación de Facebook y, sin embargo, VueJS ha conseguido hacerse un hueco entre la comunidad y una muy representativa tasa de uso.

- Framework MVVM. La ventaja de este tipo de frameworks es la facilidad para construir código bien estructurado, para poder construir aplicaciones complejas.
- *Solución Ligera*. Una de las grandes ventajas de Vue es el tamaño en Kbs del núcleo. Este tamaño puede ir aumentando, debido a la flexibilidad y facilidad para extender el framework con variedad de soluciones de terceros que son bien recibidas por la comunidad. Este tamaño compactado es una opción inteligente ya que reducirá los tiempos de carga y velocidad de las aplicaciones web.
- Templates declarativos. Los templates o plantillas en Vue se escriben en html fácilmente modificable por cualquier involucrado en el proyecto, sin tener que aprender nueva sintaxis como JSX.

- *DOM Virtual.* La implementación del DOM Virtual proporciona un alto desempeño que ciertos benchmarks ponen a Vue como líder de rendimiento de renderizado.
- *Two-way Data Binding.* Al igual que Angular emplea un data-binding bidireccional que sincroniza automáticamente el modelo con el DOM.
- *Vanilla Javascripts.* Vue.js usa vanilla Javascript (javascript plano) sin necesidad de utilizar ningún conjunto de bibliotecas javascript como Typescript, EcmaScript 6.
- *Curva de Aprendizaje Baja.* Comparado con Angular y React, Vue es una de las tecnologías javascript más sencillas para comenzar a desarrollar. Una de las mejores características es que se trata de un framework “friendly developer”. Además, posee una excelente documentación, con muchos ejemplos. VueJS no requiere grandes esfuerzos para aprender, al menos para poder obtener sus principales beneficios, lo que produce que los desarrolladores que se acercan a la tecnología puedan empezar a trabajar con soltura, sin tener que invertir demasiado tiempo.
- *Experiencia de desarrollo.* Los desarrolladores de VueJS valoran muy positivamente la experiencia de trabajo con el framework en su día a día. Las funcionalidades de Vue abarcan casi todas las expectativas de los profesionales y aplicaciones web. Además, ofrece cierta libertad a la hora de organizar el código de los proyectos y mantener las cosas simples.
- *Flexibilidad.* Gracias a VueJS tenemos la posibilidad de desarrollar muchos tipos de proyectos. No es necesario realizar aplicaciones frontend complejas, como single page applications (SPA) o Progressive Web App (PWA) para extraer ventajas de Vue.
- *Rendimiento.* VueJS nos asegura un buen rendimiento de las aplicaciones, inclusive mayor que otras alternativas de frameworks populares.

A todo esto se añade que Vue presenta una comunidad entusiasta de desarrolladores y una excelente documentación. Por ello se puede encontrar cantidad de información para aprender rápidamente a usar este framework Javascript, así como respuestas a la mayoría de los problemas que se pueden plantear en el día a día.

### 1.3.3. ¿Cuándo usar Vue.js?

Vue puede ser usado para construir diferentes tipos de aplicaciones. Es una herramienta muy útil debido a su compatibilidad con otras bibliotecas Javascript y la capacidad de añadir más lógica compleja a aplicaciones existentes. Principalmente, Vue es una herramienta para construir *Single Page Apps*, pero el ecosistema permite construir aplicaciones móviles nativas con soluciones basadas en Vue como Weex. Además, se han construido frameworks de alto nivel sobre Vue como Nuxt.js que facilita el Server Side Rendering y otro conjunto de bibliotecas que amplían la funcionalidad base del framework como Vuex o Vue-router.

Otra de las ventajas de VueJS es la posibilidad de creación de vistas reactivas. Esto quiere decir que Vue permite actualizar el HTML y CSS (la parte visual de la página) cuando cambian los datos de la aplicación, sin que el desarrollador tenga que invertir tiempo en propagar esos cambios de manera manual en cada lugar de la página donde se visualizan los datos que se alteraron.

### 1.4. ApexCharts

Cuando se trata del mundo de la investigación, es fundamental compartir sus hallazgos con su audiencia. Una de las formas más comunes de hacerlo es mediante gráficos. Entonces es necesario renderizar los gráficos en los sitios web que puede realizarse utilizando imágenes. A veces, las imágenes no se reproducen bien en pantallas de píxeles altos. Además, se vuelven borrosos cuando se amplían. Para mitigar esto, se tendría que aumentar la resolución del gráfico pero esto conducirá a otro problema: las imágenes más grandes ocupan más memoria y, por lo tanto, tardan más en cargarse. Para enfrentar este problema se puede utilizar *ApexCharts.js*. Esta es una biblioteca de código abierto fácil de usar que permite crear gráficos interactivos para los proyectos. Estas son algunas de sus características:

Capacidad de respuesta: los gráficos se escalarán según el dispositivo de visualización. Esto contribuye a una mejor interfaz de usuario para todos los dispositivos.

Anotaciones: *Apex* también permite poner etiquetas en los gráficos, lo que ayuda a comprender los gráficos con facilidad.

Animaciones: al igual que las anotaciones, las animaciones brindan una mejor experiencia al usuario y permiten personalizar la apariencia de las animaciones.

### *1.5. Conclusiones parciales del capítulo.*

En el capítulo se hace un esbozo de las tecnologías, lenguajes y herramientas utilizadas para el desarrollo de APIs y aplicaciones web. Se decide utilizar Laravel como framework de desarrollo en el lenguaje de programación PHP para el desarrollo de la API REST y VueJS para la interfaz de usuario del sistema a desarrollar debido a las características positivas de estas tecnologías y por considerarse apropiadas para dar respuesta a las necesidades planteadas.



## CAPÍTULO 2: CONCEPCIÓN DEL SISTEMA PARA EL ANÁLISIS DE LA INFORMACIÓN GENERADA POR LA APLICACIÓN BIENESTAR RCC.

En este capítulo se describen la arquitectura del sistema propuesto, los principales conceptos que se gestionan por la aplicación Bienestar RCC, el diagrama de recursos de la API; se detallan las herramientas de software utilizadas en el desarrollo de la API y se exponen las clases y funciones importantes en su implementación.

### 2.1. Arquitectura del sistema

En este trabajo se desea lograr un alto grado de independencia entre los diferentes elementos de aplicación de usuario y los aspectos internos de manipulación y representación. Los niveles de transparencia asociados con capas se reflejan en la arquitectura que se muestra en la figura 1, las cuales permiten comprender las peculiaridades de los niveles básicos de esta arquitectura de referencia en tres capas. Por ello, desde el punto de vista metodológico se exponen los resultados siguiendo estos tres niveles de modelación de la arquitectura de referencia. Para lograr el objetivo general del presente trabajo, en el que software y aplicaciones en los diferentes lenguajes de programación puedan realizar la comunicación con el software Bienestar RCC, se propone el desarrollo de una API utilizando el protocolo REST por su visibilidad, fiabilidad y escalabilidad. La API REST se desarrolló utilizando el framework de PHP Laravel por las ventajas antes expuestas de este framework.



Figura 1: Arquitectura del sistema

El usuario envía una petición, esta se direcciona al WSO2 API Gateway quien se encarga de proveer la identidad del usuario autenticado, garantizando que se acceda de forma segura a los recursos de la API. La API interactúa con el modelo de datos del software Bienestar RCC, el modelo necesita pedir y retornar datos desde la base de datos Postgresql, enviando datos a la API e invocando el resultado a la vista; aquí entra el framework VueJS que mediante sus componentes JavaScript permite mostrar la interfaz del usuario adaptada automáticamente al tamaño de la computadora, tablet, celular o cualquier otro dispositivo.

## *2.2. Descripción de la fuente de datos*

El sistema automatizado Bienestar RCC registra información sobre los establecimientos que, en el mercado interno, realizan actividades de comercio mayorista, venta minorista de mercancías, gastronomía, servicios personales, técnicos comerciales, de uso doméstico, de alojamiento y de recreación, ya sea del sector estatal, cooperativo, mixto o privado (Yucabyte, 2021).

Una descripción de los datos que se pueden encontrar en su base de datos se muestra a continuación:

- Tipo de Establecimiento: Es la clasificación del establecimiento comercial atendiendo a la actividad comercial y los servicios que se realizan en el mismo. Ejemplos de estos son: Restaurante, Bar, Cafetería con Comida, Tienda Especializada, Hotel, etc.
- Instalación. Se entiende por Instalación toda entidad, centro Comercial, Educacional, Cultural, Recreativo, Deportivo, Industrial y otros lugares o sitios donde uno o varios establecimientos, Unidades o Puntos de Ventas realicen directamente transacciones comerciales.
- Establecimientos Comerciales: Son lugares delimitados por una construcción civil, en los cuales se realizan actividades comerciales, sean de bienes o servicios, o ambas. De estar ubicados independientes y fuera de una instalación, se constituyen de hecho en esta.
- Unidades Comerciales: Son lugares delimitados o no por una construcción civil, que realizan actividad comercial, y tributan al establecimiento al cual se subordina. De estar ubicados independientes y fuera de una instalación, se constituyen de hecho en esta.

- Grupos y subgrupo de actividades:
  - Actividades comerciales de comercio mayorista: Son las actividades de venta de mercancías al por mayor, de producción nacional o de importación con destino a entidades, comercializadores minoristas o mayoristas, consumidores industriales e institucionales, entre otros.
  - Actividades comerciales de comercio minorista: Son las actividades de venta de mercancías de producción nacional o de importación y la prestación de servicios gastronómicos y comerciales con destino al consumo personal o doméstico, al detalle o al por menor.
  - Venta de mercancías: Es la venta de productos alimenticios y no alimenticios listos para la venta, o sea, que no requieran de la espera del cliente por un proceso de elaboración.
  - Servicios Gastronómicos: Es la acción de servir y satisfacer a los clientes y comensales. Es el arte del buen comer y de servir alimentos y bebidas a los clientes y comensales, con las normas técnicas de un país o época. Se interactúa con las personas naturales.
  - Servicios Gastronómicos de Alimentación Pública: Son los servicios gastronómicos que se brindan en los establecimientos destinados a estos fines, tales como; restaurantes, cafeterías, bares, centros nocturnos y otros, en cualquiera de las modalidades y categorías de servicio que integran una red abierta a la que se puede acceder libremente.
  - Servicios Gastronómicos de Alimentación Pública Limitada: Son los servicios gastronómicos que se brindan en los establecimientos destinados a estos fines, tales como; restaurantes, cafeterías, bares, centros nocturnos y otros, en cualquiera de las modalidades y categorías de servicio que integran una red Cerrada o limitada a la que no se puede acceder libremente.
  - Servicios Gastronómicos de Alimentación Social: Son los servicios gastronómicos que se ofrecen a diferentes grupos sociales en comedores y merenderos obreros, escolares y de otras entidades e instituciones que conforman

una red cerrada a la cual solo pueden acceder sus miembros y está destinada a satisfacer las necesidades nutricionales y energéticas de estos grupos.

- **Servicios Comerciales:** Son los servicios técnicos y personales que se brindan en establecimientos comerciales creados para este fin.
- **Servicios Técnicos:** Son los servicios en los cuales se aplican conocimientos científico - técnicos para su prestación, que se realiza con bienes, tangibles o intangibles.
- **Servicios Personales:** Son los servicios mediante los cuales los individuos reciben una atención directa.
- **Punto de Venta Permanente que forma parte de la RED Comercial:** Reciben esta denominación los puntos de venta que comercializan productos de consumo ocasional, ubicados en contenedores u otra estructura delimitadas o no por una construcción civil, que no constituyen propiamente un establecimiento comercial minorista, de estar ubicados independientes y fuera de una instalación, se constituyen de hecho en esta.
- **Punto de Venta Permanente que no forma parte de la RED Comercial:** Reciben esta denominación los puntos de venta que comercializan productos de consumo ocasional, ubicados en sombrillas, tarimas, mesas u otro mobiliario, ya sea en áreas exteriores o en el interior de otras edificaciones, que no constituyen propiamente un establecimiento comercial minorista. Generalmente se encuentran ligados a la actividad comercial de la empresa que los tutela o a un establecimiento Comercial
- **Punto de Venta Temporal:** Reciben esta denominación los puntos de venta no permanentes que comercializan productos de consumo ocasional, ubicados en contenedores, sombrillas, tarimas, mesas u otro mobiliario. Las solicitudes de inscripción se realizan por carta.
- **Punto de Venta Móvil:** Reciben esta denominación los puntos de ventas móviles que comercializan productos de consumo ocasional, incluye las motos,

bicicletas, motos acuáticas, surfings, aviones, animales para montar, botes de remo y otros que se definan expresamente. Las solicitudes se incluyen en la planilla del establecimiento al que tributan.

### 2.3. Configuración del Ambiente de Desarrollo

Para el desarrollo de la API se han utilizado las siguientes herramientas:

- Lenguaje de programación PHP en su versión 7.0.0
- Framework Laravel
- Visual Studio Code como IDE de trabajo con las siguientes extensiones:
  - Laravel Docs, Laravel Tinker y Laravel Intellisense para la integración del marco de trabajo Laravel en el IDE
  - Composer 0.7.1 Para integrar el composer de php y gestionar paquetes.
  - Laravel Artisan para integrar los comandos del Laravel Artisan dentro del IDE
  - Git Extension Pack, para el control de versiones del proyecto.
  - Testing Api, para las pruebas rápidas a las rutas del API.
  - SQLTools. Para el trabajo con la BD y prueba de consultas SQL.
- Insomnia Core 2020.4.2, cliente HTTP para realizar solicitudes a las API RESTful, que incluye gestión de cookies, variables de entorno, generación de código y autenticación para Mac, Windows y Linux.

### 2.4. Diagrama de Clases UML

Para acceder a una API se necesita la definición de las rutas o paths que permiten acceder a las distintas operaciones de la API; dichas rutas se conforman utilizando *Recursos*. Los recursos son los conceptos que intervienen en una funcionalidad expuesta por una API, y como tal, no se describen como acciones sino como sujetos, es decir, que un recurso no puede ser “ObtenerProvinciasPais”, sino “Provincias”. Un Diagrama de Clases UML permite describir los recursos y las relaciones entre ellos de manera que, al combinarlas, se obtienen las distintas operaciones que expone la API. Los conceptos que corresponden a los recursos se describen con *clases* (*Clases UML*) y en plural, las asociaciones entre los recursos forman las rutas a las que luego se les asignan operaciones.

Las figuras muestran los distintos tipos de operaciones que contiene cada recurso (GET, PUT, PATCH, POST, DELETE), además, se añade una tabla con una descripción informativa sobre las operaciones que están implementadas por la API.

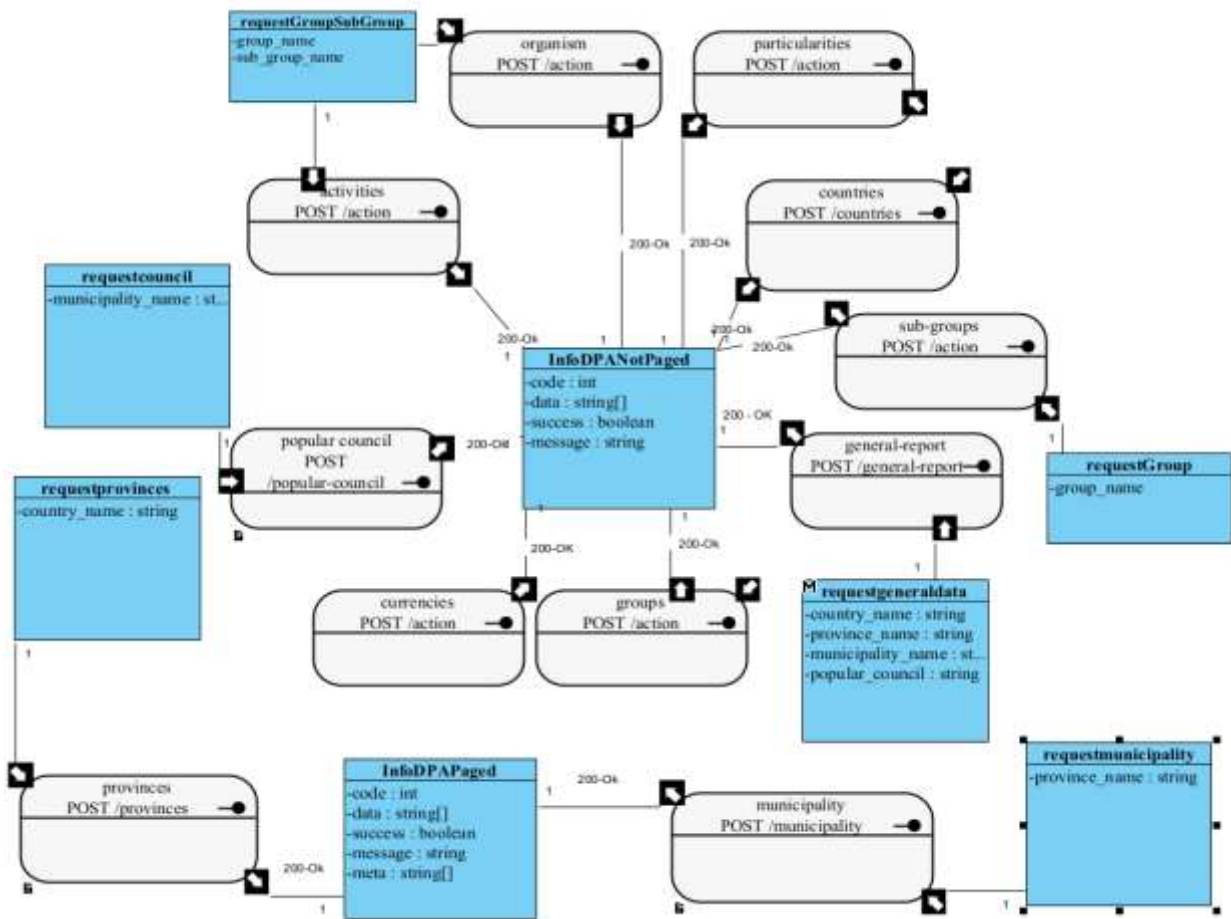


Figura 2: Diagrama de recursos UML para la sección de los nomencladores

La descripción de los recursos que se definen en el modelo pueden observarse en la tabla 1:

Tabla 1: Lista de Recursos de la sección de los nomencladores

Recurso	URI	Descripción
<i>countries</i>	<i>/countries</i>	Obtiene una lista de los países registrados.
<i>provinces</i>	<i>/provinces</i>	Obtiene una lista de las provincias que pertenecen a un país (parámetro <i>country_name</i> ).

<i>municipality</i>	<i>/municipality</i>	Obtiene una lista de todos los municipios de una provincia y país seleccionados (parámetros <i>province_name</i> , <i>country_name</i> ).
<i>popular-council</i>	<i>/popular-council</i>	Obtiene todos los consejos populares de un municipio (parámetro <i>municipality_name</i> ).
<i>general-report</i>	<i>/general-report</i>	Obtiene los datos generales de las instalaciones, filtrado por los parámetros <i>province_name</i> , <i>country_name</i> , <i>municipality_name</i> y <i>popular_council</i> .
<i>groups</i>	<i>/groups</i>	Muestra la lista de grupos por los que se asocian las actividades. Por ejemplo: Sistema de Alojamiento, Venta de Mercancía.
<i>Subgroups</i>	<i>/sub-groups</i>	Muestra la lista de los subgrupos pertenecientes a un grupo (parámetro <i>group_name</i> ). Por ejemplo: Subgrupo: venta minorista de productos alimenticios. Grupo: Venta de mercancías.
<i>Activities</i>	<i>/activities</i>	Muestra la lista de las actividades comerciales. Se pueden obtener las actividades específicas de un grupo o sub-grupo mediante el parámetro <i>group_name</i> o <i>subgroup_name</i> respectivamente.
<i>Organism</i>	<i>/organism</i>	Muestra la lista de organismos.
<i>particularities</i>	<i>/particularities</i>	Muestra la lista de las particularidades.
<i>currencies</i>	<i>/currencies</i>	Muestra la lista de monedas con las que opera la red comercial.

Las clases del diagrama representan la solicitud y la respuesta de cada recurso del REST API en formato JSON. Se puede describir de la siguiente manera (véase la tabla 2):

**Tabla 2: Respuestas de los recursos de la sección de los nomencladores**

Clase	Descripción	Tipo
<i>InfoDPANotPaged</i>	Contiene el atributo <i>code</i> que indica el estado del recurso si el <i>code</i> es 200 significa que se accedió correctamente	Respuesta

	al recurso, el atributo <i>data</i> que contiene los datos requeridos por el cliente, el atributo <i>success</i> que muestra si la operación se realizó o no correctamente, el atributo <i>message</i> muestra si la operación fue realizada correctamente o si ocurrió algún error durante el proceso.	
<i>InfoDPAPaged</i>	Contiene los mismos atributos de la clase <i>InfoDPANotPaged</i> y otro atributo <i>meta</i> que contiene los datos de la paginación del atributo <i>data</i> .	Respuesta
<i>requestprovinces</i>	Contiene el atributo <i>country_name</i> , filtra las provincias por ese valor.	Solicitud
<i>requestmunicipality</i>	Contiene el atributo <i>province_name</i> , filtra los municipios por ese valor.	Solicitud
<i>requestcouncil</i>	Contiene el atributo <i>municipality_name</i> , filtra los consejos populares por ese valor.	Solicitud
<i>requestgeneraldata</i>	Contiene los atributos <i>province_name</i> , <i>country_name</i> <i>municipality_name</i> , para filtrar los datos por el valor de dichos atributos.	Solicitud
<i>requestGroup</i>	Contiene el atributo <i>group_name</i> para filtrar los datos por el valor del atributo.	Solicitud
<i>requestGroupSubGroup</i>	Contiene los atributos <i>group_name</i> y <i>subgroup_name</i> para filtrar los datos por el valor del atributo.	Solicitud



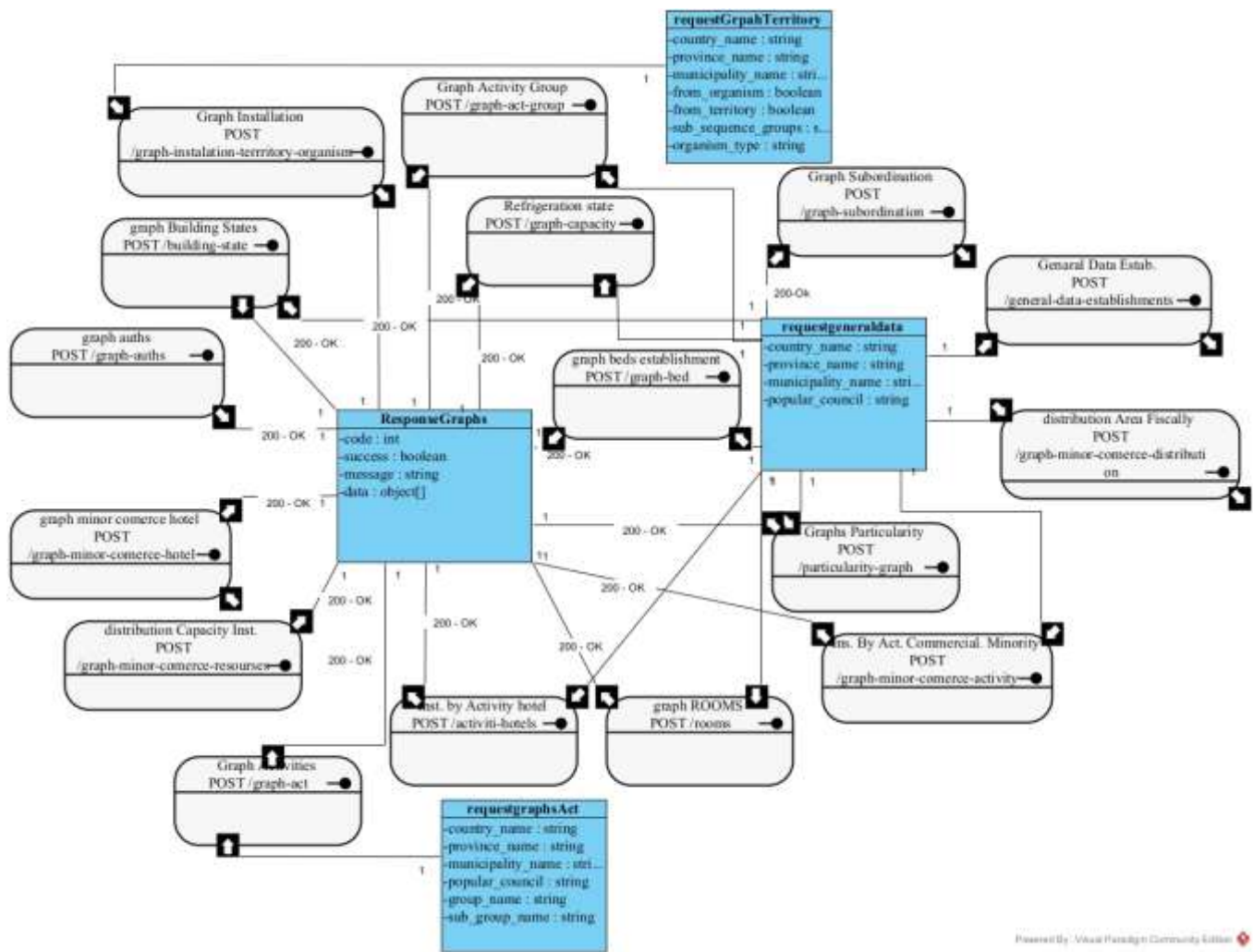


Figura 3: Diagrama de recursos UML para la sección de análisis de datos

En la tabla 3 se describen algunos de los recursos que se definen en el modelo:

Tabla 3: Lista de recursos para la sección de análisis de datos

Recurso	URI	Descripción
<i>Graph Activity Group</i>	/graph-act-group	Obtiene cantidad de instalaciones por grupos de actividades.
<i>Graph Activities</i>	/graph-act	Obtiene cantidad de instalaciones por actividad.

<i>Graph Installation</i>	<i>/graph-instalation-territory-organism</i>	Obtiene las instalaciones por organismos o territorios y de esas instalaciones cuántas pertenecen a uno o varios grupos.
<i>Graph Subordination</i>	<i>/graph-subordination</i>	Obtiene cantidad de instalaciones desglosadas por el nivel de subordinación (municipal, provincial, nacional)
<i>graph ROOMS</i>	<i>/rooms</i>	Obtiene cantidad de instalaciones agrupadas por el tipo de habitaciones que presentan sus establecimientos.
<i>Inst by Activity hotel</i>	<i>/activiti-hotels</i>	Obtiene cantidad de instalaciones desglosadas por las actividades de la red hotelera.
<i>graph Building States</i>	<i>/building-state</i>	Obtiene cantidad de instalaciones desglosadas por el estado constructivo de sus establecimientos.
<i>Graph Particularity</i>	<i>/particularity-graph</i>	Obtiene cantidad de instalaciones agrupadas por las particularidades según el tipo de clasificación; Programas nacionales, Priorizados por el MINCIN y Otros.
<i>General Data Estab.</i>	<i>/general-data-establishments</i>	Obtiene las estadísticas generales de los establecimientos atendiendo a su área constructiva y capacidad de almacenaje.
<i>Refrigeration state</i>	<i>/graph-capacity</i>	Obtiene la capacidad de almacenaje y la diferencia entre esta y el área de servicio desglosado por las actividades de productos alimenticios, No alimenticios y mixtos.
<i>graph beds establishment</i>	<i>/graph-bed</i>	Obtiene cantidad de instalaciones agrupadas por el tipo de cama que presentan los establecimientos.
<i>Ins. By Act. Commercial Minority</i>	<i>/graph-minor-commerce-activity</i>	Obtiene la cantidad de instalaciones que desarrollan actividades de comercio minorista.

<i>distribution Area Fiscally</i>	<i>/graph-minor-commerce-distribution</i>	Obtiene cantidad de instalaciones agrupadas por la distribución del área (Servicios o ventas, Almacén).
<i>Graph minor commerce Resources</i>	<i>/graph-minor-commerce-resources</i>	Obtiene cantidad de instalaciones agrupada por los recursos del establecimiento.
<i>graph minor commerce hotel</i>	<i>/graph-minor-commerce-hotel</i>	Obtiene cantidad de instalaciones que pertenecen a la red hotelera, agrupadas por las actividades de la red hotelera y clasificadas por tipo extrahotelera o intrahotelera
<i>graph auths</i>	<i>/graph-auths</i>	Obtiene cantidad de autorizos por instalaciones y el nivel de validez que presentan.

Las clases del diagrama representan la solicitud y la respuesta de cada recurso del REST API en formato JSON; estas se describen en la tabla 4 a continuación:

**Tabla 4: Respuestas de los recursos para la sección de análisis de datos**

Clase	Descripción	Tipo
<i>requestgeneraldata</i>	Contiene los atributos <i>province_name</i> , <i>country_name</i> <i>municipality_name</i> , para filtrar los datos por el valor de dichos atributos.	Solicitud
<i>requestgraphsAct</i>	Contiene los atributos <i>province_name</i> , <i>country_name</i> <i>municipality_name</i> , <i>group_name</i> , <i>subgroup_name</i> para filtrar los datos por el valor de dichos atributos.	Solicitud
<i>requestGrpahTerritory</i>	Contiene los atributos <i>province_name</i> , <i>country_name</i> <i>municipality_name</i> , para filtrar los datos por el valor de dichos atributos, el campo <i>from_organism</i> que indica si el dataset va dirigido a los organismos o <i>from_territory</i> , si va distribuido por territorios; contiene el dataset <i>sub_sequence_groups</i> que se utiliza para determinar la cantidad de instalaciones por cada grupo.	Solicitud

<i>ResponseGraphs</i>	<p>Contiene el atributo <i>code</i> que indica el estado del recurso, si el <i>code</i> es 200 significa que se accedió correctamente al recurso; el atributo <i>data</i> que contiene los dataset solicitados por el cliente; el atributo <i>success</i> que muestra si la operación se realizó o no correctamente; el atributo <i>message</i> muestra si la operación fue realizada correctamente o si ocurrió algún error durante el proceso</p>	Respuesta
-----------------------	---	-----------

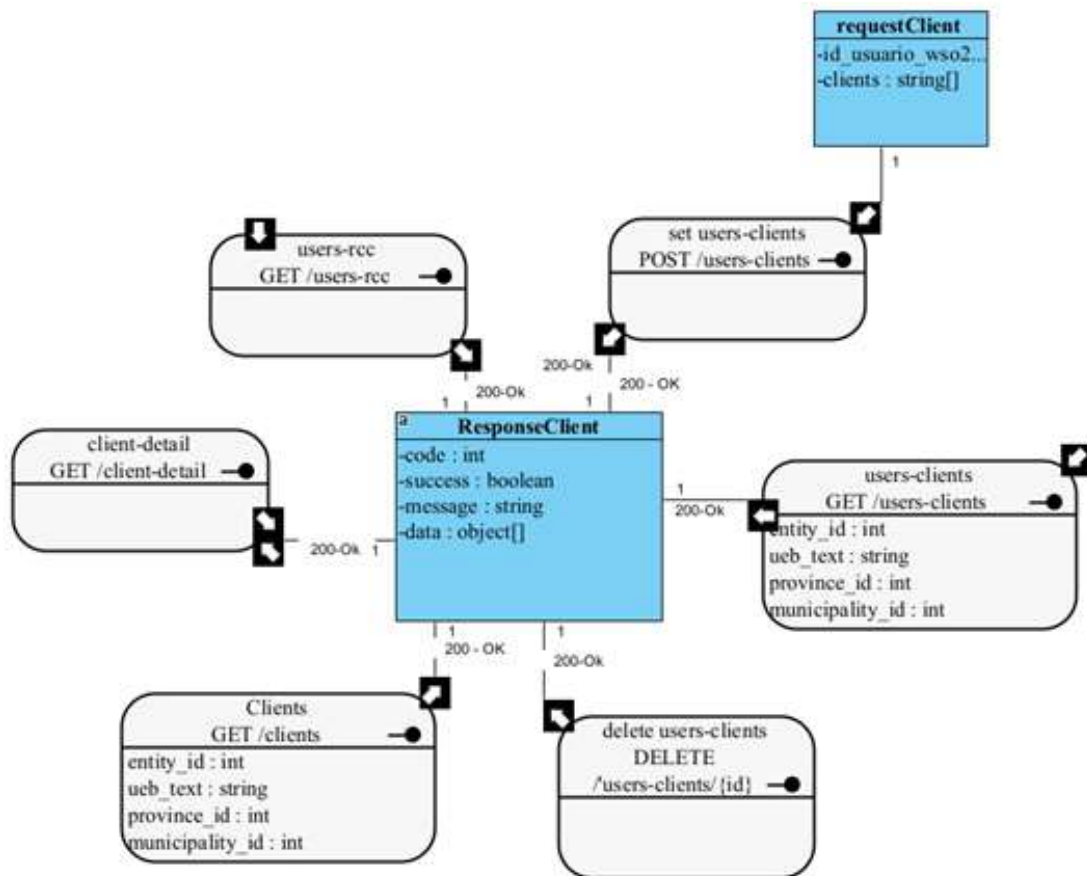


Figura 4: Diagrama recursos UML sección clientes

En la tabla 5 se muestra una descripción de algunos de los recursos que aparecen en el diagrama:

Tabla 5: Lista de recursos para la sección clientes

Recurso	URI	Descripción
<i>users-rc</i>	<i>/users-rc</i>	Muestra los usuarios con el rol RCC.

<i>Clients</i>	<i>/clients</i>	Muestra los clientes por entidades de la red comercial. Acepta los parámetros <i>entity_id</i> , <i>ueb_text</i> , <i>province_id</i> , <i>municipality_id</i> , <i>subordination</i> para realizar el filtrado de los datos.
<i>users-clients</i>	<i>GET /users-clients</i>	Muestra los usuarios con el rol RCC asociados a los clientes de la red comercial. Acepta los parámetros <i>entity_id</i> , <i>ueb_text</i> , <i>province_id</i> , <i>municipality_id</i> , <i>subordination</i> para realizar el filtrado de los datos.
<i>set users-clients</i>	<i>POST /users-clients</i>	Asocia un usuario con el rol de RCC a clientes de la red comercial.
	<i>DELETE /users-clients/{id}</i>	Elimina la asociación de un cliente a un usuario RCC utilizando el id del usuario.
<i>client-detail</i>	<i>/client-detail</i>	Obtiene los detalles de los clientes asociados al usuario de RCC autenticado.

Las clases del diagrama representan la solicitud y la respuesta de cada recurso del REST API en formato JSON. Se puede describir de la siguiente manera (véase la tabla 6):

**Tabla 6: Respuestas de los recursos de la sección clientes**

Clase	Descripción	Tipo
<i>ResponseClient</i>	Contiene el atributo <i>code</i> que indica el estado del recurso, si el <i>code</i> es 200 significa que se accedió correctamente al recurso; el atributo <i>data</i> con los datos solicitados; el atributo <i>success</i> que muestra si la operación se realizó o no correctamente; el atributo <i>message</i> muestra si la operación fue realizada correctamente o si ocurrió algún error durante el proceso.	Respuesta
<i>requestClient</i>	Contiene los datos requeridos para asociar un usuario a un cliente en rcc el id del cliente y el correo electrónico del usuario.	Solicitud

Los posibles valores del campo *code* en las clases *ResponseClient*, *ResponseGraphs*, *InfoDPAPaged*, *InfoDPANoTPaged* pueden ser:

- 200 OK - Respuesta a un GET, PUT, PATCH o DELETE exitoso. Puede ser usado también para un POST que no resulta en una creación.
- 400 Bad Request – [Petición Errónea] La petición está malformada, como, por ejemplo, si el contenido no fue bien reconocido. El error se debe mostrar también en el JSON de respuesta.
- 401 Unauthorized – [Desautorizada] Los detalles de autenticación son inválidos o no son otorgados. También útil para disparar un mensaje emergente (popup) de autorización si la API es usada desde un navegador.
- 403 Forbidden – [Prohibida] La autenticación es exitosa pero el usuario no tiene permiso al recurso en cuestión.
- 404 Not Found – [No encontrada] Se solicita un recurso no existente.
- 405 Method Not Allowed – [Método no permitido] Un método HTTP que está siendo pedido no está permitido para el usuario autenticado.
- 429 Too Many Requests – [Demasiadas peticiones] Una petición es rechazada debido a la tasa límite.
- 500 – Internal Server Error – [Error Interno del servidor] Los desarrolladores de API no deberían usar este código. En su lugar se debería registrar (log) el fallo y no devolver respuesta.

## 2.5. Implementación del API REST

A continuación se describe la implementación de la API REST, como parte de la arquitectura presentada en la sección 2.1.

### 2.5.1 Creación de las Rutas de la API RESTful

Para definir una API en Laravel, ya sea mediante controladores tipo RESTful o controladores normales, se utiliza el fichero de rutas routes/api.php o de routes/web.php. Es recomendable utilizar el routes/api.php. En este fichero, las rutas se definen usando la clase Route, los métodos get, post, put y delete, los controladores de recursos, los grupos de rutas, middleware, parámetros, etc. Además, al ejecutar el comando php artisan route:list estas rutas aparecerán en

el mismo listado. Las diferencias entre estos ficheros de rutas son dos («Routing - Documentation Laravel PHP Framework», 2021):

- Los filtros o middleware que se utilizan: Mientras que en routes/web.php se aplican filtros para inicializar la sesión, las cookies, los bindings y la protección CSRF; en las rutas que se añadan a routes/api.php únicamente se aplican filtros para limitar el número de peticiones y para cargar los bindings (la inyección de dependencias). En el fichero app/Http/Kernel.php, dentro de su array \$middlewareGroups vemos dos grupos: web y api, con los filtros que se aplican en cada caso (véase la figura 5).

```
/**
 * The priority-sorted list of middleware.
 *
 * This forces non-global middleware to always be in the given order.
 *
 * @var array
 */
protected $middlewarePriority = [
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    \App\Http\Middleware\Authenticate::class,
    \Illuminate\Routing\Middleware\ThrottleRequests::class,
    \App\Http\Controllers\NoThrottle::class,
    \Illuminate\Session\Middleware\AuthenticateSession::class,
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
    \Illuminate\Auth\Middleware\Authorize::class,
];
```

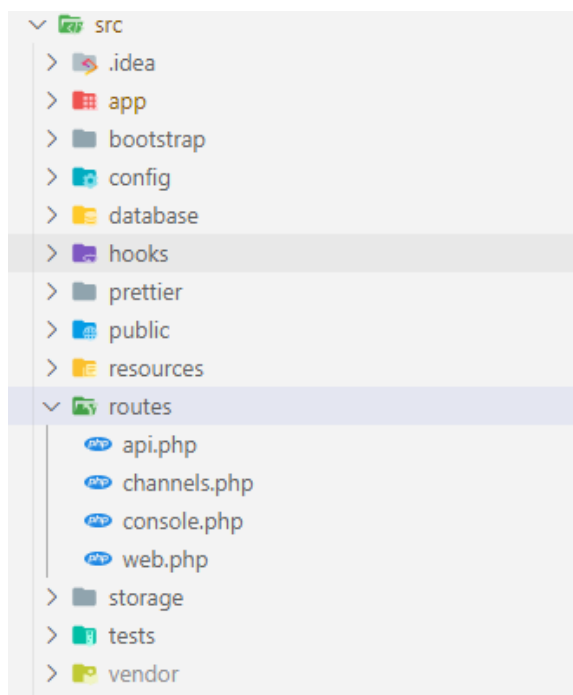
**Figura 5: Archivo Kernel.php**

- El prefijo de las rutas: A todas las rutas que se especifican en routes/api.php se les añadirá el prefijo api. Por lo que, si se añade la ruta *countries* para devolver el listado de países, para realizar la consulta se tendría que acceder a *api/countries*.

Estas diferencias aportan varias ventajas; por un lado, se tienen separadas todas las rutas de la API tanto en un fichero distinto como por un prefijo, por lo que no se mezclarán los distintos tipos de rutas del sitio web. Además, las peticiones a las rutas de la API no tienen que cargar middleware innecesario, permitiendo también definir otros filtros propios de una API, como el de rate limit. Las peticiones a estas rutas no mantendrán sesión, por lo tanto, se tendrán que

autenticar bien mediante tokens o mediante seguridad HTTP básica (usando las cabeceras de la petición). Además, en las respuestas no se tiene que devolver una vista (con HTML), sino directamente el dato o la lista de datos solicitados en formato plano, en XML o en JSON.

El fichero de rutas de este proyecto se encuentra ubicado en /routes/api.php (véase la figura 6)



**Figura 6: Estructura del proyecto para desarrollo de la API, definición de rutas.**

La forma más básica de rutas admite una URI y un *closure()* (función anónima). Por ejemplo:

```
Route::get('/', function()
{
    return 'Hola Mundo.';
});
```

En lugar de definir toda la lógica para la gestión de una petición dentro de Closures o funciones anónimas en los archivos de rutas, se puede organizar este comportamiento en unas clases llamadas Controladores (controllers) lo que se denomina una *ruta Resource* (Laravel, 2021).

Por ejemplo:

```
Route::post('provinces','Business\\NegocioController@getProvinces');
```



El fichero de rutas de este proyecto routes/api.php se realizó utilizando la última forma descrita, o sea, utilizando recursos de controladores RESTful (véase la figura 7):

```
<?php

use Illuminate\Support\Facades\Route;

Route::post('general-report', 'Business\\NegocioController@getGeneralDataInstalation');
Route::post('countries', 'Business\\NegocioController@getCountries');
Route::post('provinces', 'Business\\NegocioController@getProvinces');
Route::post('municipality', 'Business\\NegocioController@getMunicip');
Route::post('populate-consul', 'Business\\NegocioController@getConsul');
Route::post('groups', 'Business\\NegocioController@getGroups');
Route::post('sub-groups', 'Business\\NegocioController@getSubGroups');
Route::post('activities', 'Business\\NegocioController@getActividades');
Route::post('organisms', 'Business\\NegocioController@getOrganisms');
Route::post('particularities', 'Business\\NegocioController@getParticularidades');
Route::post('currencies', 'Business\\NegocioController@getTipoMoneda');
Route::post('graph-act-group', 'Business\\NegocioController@getGraphActividadesGrupos');
Route::post('graph-act', 'Business\\NegocioController@getGraphActividades');
Route::post('graph-installation-territory-organism', 'Business\\NegocioController@getGraaphInstalation');
Route::post('graph-subordination', 'Business\\NegocioController@getGraphSubordinacion');
Route::post('building-state', 'Business\\NegocioController@getBuildingStates');
```

**Figura 7: Fichero de rutas api.php**

Las rutas disponibles en el Proyecto se pueden listar a través del comando:

```
php artisan route:list --path="api/"
```

Domain	Method	URI	Name	Action	Middleware
	POST	api/activities-notals		App\Http\Controllers\Business\NegocioController@getInstalacionesNotals	api
	GET HEAD	api/activities		App\Http\Controllers\Business\NegocioController@getActivities	api
	POST	api/activities		App\Http\Controllers\Business\NegocioController@getActivities	api
	POST	api/auth-commercial		App\Http\Controllers\Business\NegocioController@getAutorizacionComerciales	api
	POST	api/building-state		App\Http\Controllers\Business\NegocioController@getBuildingStates	api
	GET HEAD	api/client-detail		App\Http\Controllers\Business\NegocioController@getClientDetail	api
	GET HEAD	api/clients		App\Http\Controllers\Business\NegocioController@getClients	api
	GET HEAD	api/construction_state		App\Http\Controllers\Business\NegocioController@getEstadoConstruction	api
	POST	api/countries		App\Http\Controllers\Business\NegocioController@getCountries	api
	POST	api/currencies		App\Http\Controllers\Business\NegocioController@getTiposMoneda	api
	POST	api/data-catalog		App\Http\Controllers\Business\NegocioController@getCatalogo	api
	GET HEAD	api/documentation	IS-swagger-api	ISwagger\Http\Controllers\SwaggerController@api	
	GET HEAD	api/entities		App\Http\Controllers\Business\NegocioController@getEntities	api
	POST	api/establishment		App\Http\Controllers\Business\NegocioController@getEstablecimientos	api
	POST	api/general-data-establishments		App\Http\Controllers\Business\NegocioController@getGeneralDataEstab	api
	POST	api/general-report		App\Http\Controllers\Business\NegocioController@getGeneralDataInstallation	api
	POST	api/graph-act		App\Http\Controllers\Business\NegocioController@getGraphActivities	api
	POST	api/graph-act-group		App\Http\Controllers\Business\NegocioController@getGraphActivitiesGroups	api
	POST	api/graph-auth		App\Http\Controllers\Business\NegocioController@getGraphAuthCommercial	api
	POST	api/graph-auth-parade		App\Http\Controllers\Business\NegocioController@getGraphAuthParadeCommercial	api
	POST	api/graph-auths		App\Http\Controllers\Business\NegocioController@getAuths	api
	POST	api/graph-bed		App\Http\Controllers\Business\NegocioController@getBED	api
	POST	api/graph-capacity		App\Http\Controllers\Business\NegocioController@getRefrigerationState	api
	POST	api/graph-installation-territory-organism		App\Http\Controllers\Business\NegocioController@getGraphInstallation	api
	POST	api/graph-minor-commerce-activity		App\Http\Controllers\Business\NegocioController@getInstByActCommercialHistoria	api
	POST	api/graph-minor-commerce-distribution		App\Http\Controllers\Business\NegocioController@getDistributionAreaFiscal	api
	POST	api/graph-minor-commerce-hotel		App\Http\Controllers\Business\NegocioController@getHotelDataNetwork	api
	POST	api/graph-minor-commerce-resources		App\Http\Controllers\Business\NegocioController@getDistributionCapacityFirst	api
	POST	api/graph-subordination		App\Http\Controllers\Business\NegocioController@getGraphSubordination	api
	POST	api/groups		App\Http\Controllers\Business\NegocioController@getGroups	api
	GET HEAD	api/groups		App\Http\Controllers\Business\NegocioController@getGroups	api
	POST	api/installation-establishment		App\Http\Controllers\Business\NegocioController@getInstalacionesComercialesEstablement	api
	POST	api/installations		App\Http\Controllers\Business\NegocioController@getInstalaciones	api
	POST	api/list-expired		App\Http\Controllers\Business\NegocioController@getVigenciaAuthExpired	api
	POST	api/municipality		App\Http\Controllers\Business\NegocioController@getMunicip	api
	GET HEAD	api/oauth2-callback	IS-swagger.oauth2_callback	ISwagger\Http\Controllers\SwaggerController@oauth2Callback	
	POST	api/organisms		App\Http\Controllers\Business\NegocioController@getOrganisms	api
	POST	api/particularities		App\Http\Controllers\Business\NegocioController@getParticularidades	api
	POST	api/particularity-graph		App\Http\Controllers\Business\NegocioController@getGraphParticularity	api
	POST	api/populate-console		App\Http\Controllers\Business\NegocioController@getConsole	api
	POST	api/provinces		App\Http\Controllers\Business\NegocioController@getProvinces	api
	POST	api/rooms		App\Http\Controllers\Business\NegocioController@getRooms	api
	GET HEAD	api/routes		closure	api
	GET HEAD	api/routes		App\Http\Controllers\Business\NegocioController@getEstado	api
	POST	api/sub-groups		App\Http\Controllers\Business\NegocioController@getSubgroups	api
	GET HEAD	api/sub-groups		App\Http\Controllers\Business\NegocioController@getSubgroups	api
	GET HEAD	api/subordination		App\Http\Controllers\Business\NegocioController@getSubordination	api
	POST	api/user-clients		App\Http\Controllers\Business\NegocioController@getAssociatedClientUser	api
	POST	api/user-rol		App\Http\Controllers\Business\NegocioController@getUserRoles	api
	POST	api/users-clients		App\Http\Controllers\Business\NegocioController@getAssociatedClients	api
	GET HEAD	api/users-clients		App\Http\Controllers\Business\NegocioController@getAssociatedClients	api
	DELETE	api/users-clients/{id}		App\Http\Controllers\Business\NegocioController@deleteAssociatedClients	api
	GET HEAD	api/users-rcc		App\Http\Controllers\Business\NegocioController@getUsers	api

Figura 8: Listado de rutas de la API.

### 2.5.2 Diseño e implementación de la clase controladora

Laravel brinda una capa de controladores que permite organizar la lógica de las rutas dentro de métodos llamados «acciones» y de clases conocidas como «controladores»

Los Controladores permiten hacer las siguientes tareas:

- En lugar de definir toda la lógica de las rutas en el fichero *routes/api.php*, se puede organizar toda esa lógica a través de un *Controller*.
- En los *Controllers* se puede agrupar toda la lógica de las peticiones HTTP dentro de una misma clase.

En el Controlador se programan las tareas a realizar para cada una de las peticiones de la API RESTful. La clase *NegocioController* es la clase controladora en este proyecto que se encarga de gestionar las peticiones del archivo de rutas (véase la figura 9).

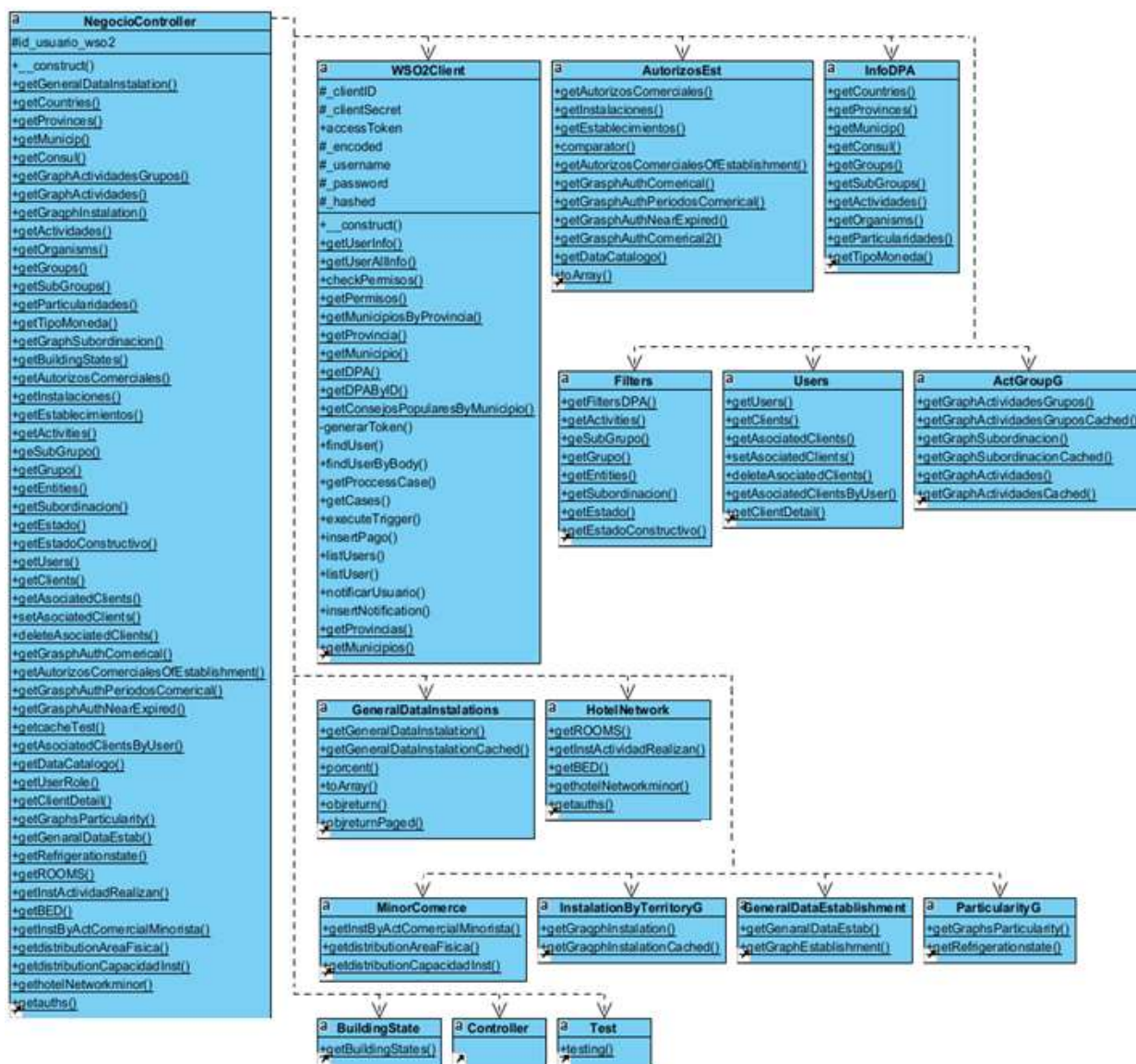
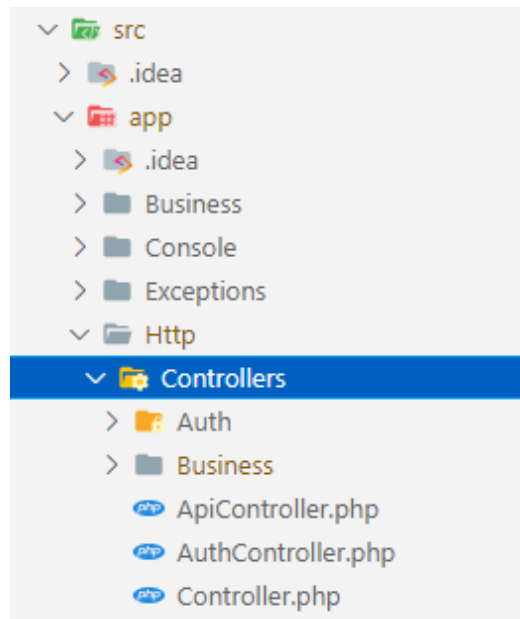


Figura 9: Diagrama de clases: clase controladora

- La clase GeneralDataInstalations contiene las funciones para acceder a la información relacionada con las instalaciones.
- La clase HotelNetwork contiene las funciones relacionada con la Red hotelera.
- La clase MinorCommerce contiene las funciones relacionadas con las actividades del comercio Minorista.
- La clase InstalationByterritoryG contiene funciones que permiten obtener información sobre las instalaciones por territorios u organismos.

- La clase `GeneralDataEstablishment` contiene las funciones necesarias para listar los datos generales de los establecimientos.
- La clase `ParticularityG` contiene las funciones necesarias para obtener los datos de las instalaciones por particularidades, así como por el área para almacenamiento.
- La clase `BuildingState` contiene funciones necesarias para obtener las instalaciones según el estado constructivo de sus establecimientos.
- La clase `WSO2Client` contiene todo lo relacionado con la autenticación, el *token* y los datos del usuario, entre las que se incluyen las funciones para generar el *token*, obtener los datos del usuario etc.
- La clase `AutorizosEstablecimientos` Contiene todas las funciones para obtener los autorizos por cada instalación y establecimiento, así como su nivel de validez, agrupados por años y meses.
- La clase `InfoDpa` contiene las funciones relacionadas con los datos de los países, las provincias, municipios, consejos populares, los grupos, las actividades, las particularidades, los organismos, y los tipos de monedas.
- La clase `Users` contiene las funciones relacionadas con la gestión de los datos de los usuarios de RCC, asociación de usuarios en el sistema y la compartimentación de la información por usuarios y los clientes.
- La clase `ActGroupG` permite obtener cantidad de instalaciones por actividades, particularidades y por el nivel de subordinación.
- La clase `NegocioController` hereda de la clase *Controller* de Laravel, la cual contiene las principales funciones de los controladores, no es obligatorio que todos los controladores extiendan de ella, pero si es recomendable pues contiene una serie de atributos y métodos que pueden facilitar el trabajo con el uso de estos.

Estos controladores se encuentran normalmente en el directorio `app/Http/Controllers` (véase la figura 10).



**Figura 10: Estructura del proyecto, Carpeta `app/Http/Controllers`**

Un controlador no es más que un archivo `.php` con una clase que extiende de la clase `App\Http\Controllers\Controller`.

La clase *NegocioController* contiene todas las funciones. A continuación se muestra la implementación de dicha clase, ubicada en el fichero `app/Http/Controllers/ Business/ NegocioController.php`:

```

class NegocioController extends Controller
{
    protected $id_usuario_wso2;

    public function __construct(Request $request)
    {
        $this->id_usuario_wso2 = WSO2Client::getUserInfo($request->header('hashed'),$request->email);
    }

    public static function getGeneralDataInstalation(Request $request){
        return response()->json(GeneralDataInstalations::getGeneralDataInstalationCached($request));
    }

    public static function getCountries(Request $request){
        return response()->json(InfoDPA::getCountries($request));
    }

    public static function getProvinces(Request $request){
        return response()->json(InfoDPA::getProvinces($request));
    }

    public static function getMunicip(Request $request){
        return response()->json(InfoDPA::getMunicip($request));
    }

    public static function getConsul(Request $request){
        return response()->json(InfoDPA::getConsul($request));
    }

    public static function getGraphActividadesGrupos(Request $request){
        return response()->json(ActGroupG::getGraphActividadesGruposCached($request));
    }

    public static function getGraphActividades(Request $request){
        return response()->json(ActGroupG::getGraphActividadesCached($request));
    }

    public static function getGraqphInstalation(Request $request){
        return response()->json(InstalationByTerritoryG::getGraqphInstalationCached($request));
    }

    public static function getActividades(Request $request){
        return response()->json(InfoDPA::getActividades($request));
    }

    public static function getOrganisms(Request $request){
        return response()->json(InfoDPA::getOrganisms($request));
    }

    public static function getGroups(Request $request){
        return response()->json(InfoDPA::getGroups($request));
    }

    public static function getSubGroups(Request $request){
        return response()->json(InfoDPA::getSubGroups($request));
    }

    public static function getParticularidades(Request $request){
        return response()->json(InfoDPA::getParticularidades($request));
    }

    public static function getTipoMoneda(Request $request){
        return response()->json(InfoDPA::getTipoMoneda($request));
    }

    public static function getGraphSubordinacion(Request $request){
        return response()->json(ActGroupG::getGraphSubordinacionCached($request));
    }

    public static function getBuildingStates(Request $request){
        return response()->json(BuildingState::getBuildingStates($request));
    }
}

```

**Figura 11: Implementación de la clase NegocioController.php**

Seguidamente se muestra la implementación de una de las funciones de la Clase Controladora *NegocioController.php*:



```

public static function getGraphActividades(Request $request){
    return response()->json(ActGroupG::getGraphActividadesCached($request));
}

```

**Figura 12: Implementación de la clase getGraphActividades**

Se ejecuta el método *getGraphActividadesCached(\$request)* de la clase *ActGroupG* al consumir el recurso *graph-act* definido en el fichero de rutas; este método recibe un objeto *Request* como parámetro que contiene un JSON con los valores del país, la provincia, el municipio, el grupo y el subgrupo de la actividad. La función busca en la base de datos de caché para revisar si ya se ha hecho una petición anterior a ese recurso, en caso afirmativo se devuelven los datos en caché, si no, se conecta a la base de datos original; en ambos casos devuelve en formato JSON un dataset con la cantidad de instalaciones por cada actividad atendiendo a los parámetros especificados en el objeto *request*. Un ejemplo de uso de esta función puede ser.

```

method: 'POST',
url: 'http://127.0.0.1:8000/api/graph-act',
headers: {'Content-Type': 'application/json'},
data: {group_name: 'SISTEMA DE ALOJAMIENTO', province_name: 'Artemisa'}

```

**Figura 13: Ejemplo de petición al recurso *graph-act***

De este modo, se obtiene como respuesta lo que se puede observar en la figura 14.

```
{
  "info": {
    "code": 2000,
    "success": true,
    "message": "Success Operation"
  },
  "data": [
    {
      "activity": "No especificado",
      "total": 0
    },
    {
      "activity": "Hotel",
      "total": 311
    },
    {
      "activity": "Aparthotel",
      "total": 34
    },
    {
      "activity": "Villa",
      "total": 204
    },
    {
      "activity": "Motel",
      "total": 36
    },
    {
      "activity": "Casa de Alojamiento",
      "total": 327
    },
    {
      "activity": "Casa de Tránsito",
      "total": 18
    },
    {
      "activity": "Albergue",
      "total": 29
    },
    {
      "activity": "Supermercado",
      "total": 245
    },
    {
      "activity": "Minimercado",
      "total": 2281
    },
    {
      "activity": "Bodega",
      "total": 984
    },
    {
      "activity": "Bodega Mixta",
      "total": 10104
    },
    {
      "activity": "Combinado Alimentario Industrial",
      "total": 2
    }
  ]
}
```

Figura 14: Ejemplo de respuesta del recurso graph-act



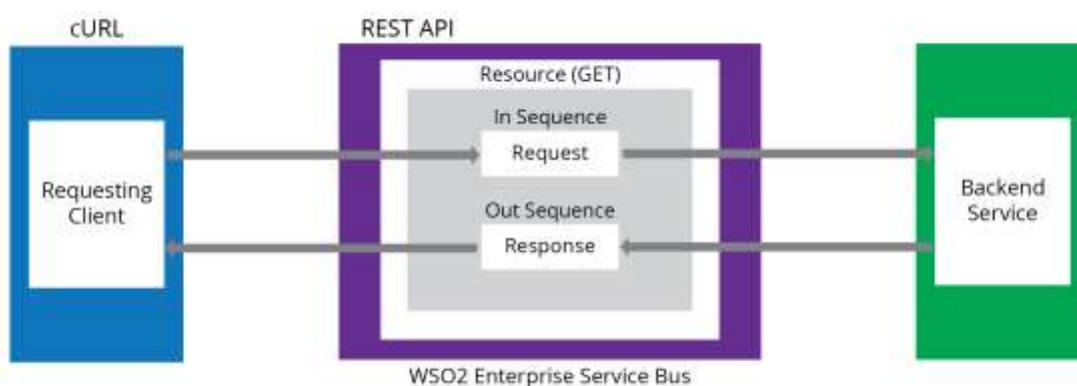
### 2.5.3 WSO2 y seguridad de la API

WSO2 es una plataforma empresarial que ofrece diferentes soluciones para crear, integrar, administrar, proteger y analizar APIs y servicios Web.

Como se mencionó anteriormente, las peticiones a las rutas del API no mantendrán sesión, por lo tanto, se tendrán que autenticar, mediante tokens o mediante cabeceras HTTP. El WSO2 API Gateway se encarga de generar tokens válidos para los usuarios autenticados, así como proveer la identidad del usuario autenticado, garantizando que se acceda de forma segura a los recursos de la API (*Secure APIs using OAuth2 Access Tokens - WSO2 API Manager Documentation 3.0.0*, 2020).

Con el WSO2 API Manager se pueden aplicar políticas de seguridad a las API (autenticación, autorización), trabajar con el estándar OAuth2 para el acceso a la API (implícito, código de autorización, cliente, SAML, tipo de concesión IWA)

Por otra parte, se pueden restringir tokens de acceso API a dominios / IPs, soportar las aplicaciones de servidores clave de terceros partidos para el registro de aplicación y la generación y validación de tokens, además, se puede configurar Single *Sign-On* (SSO) con SAML 2.0 para una fácil integración con las aplicaciones web existentes.



**Figura 15: WSO2 - diagrama del API gateway**

WSO2 recibirá la solicitud del cliente, en lugar de que el cliente envíe mensajes directamente al servicio backend, desacoplando así al cliente y el servicio backend.

Para generar dicho token, WSO2 provee varias URIs que pueden ser utilizadas tanto por el cliente que se conecta al API Gateway como para la propia API. En este proyecto las llamadas a esos recursos se encuentran en la carpeta modules/WSO2Client.php de este proyecto.

Implementación de la clase WSO2Client:

Esta clase contiene llamadas a los recursos de WSO2 necesarios para generar el token, obtener datos de usuarios, refrescar el token, obtener todos los usuarios etc.

- Función generarToken

```
private function generarToken()
{
    $client = new Client(['verify' => false]);

    $response = $client->post(env('WSO2_URL') . Constants::WSO2_TOKEN, [
        'headers' => [
            'Content-Type' => 'application/x-www-form-urlencoded',
            'Authorization' => 'Basic ' . base64_encode($this->_clientId . ':' . $this->_clientSecret)
        ],
        'form_params' => [
            'grant_type' => 'password',
            'username' => $this->_username,
            'password' => $this->_password,
            'consumerSecret' => $this->_clientSecret,
            'scope' => 'openid',
        ],
        'proxy' => ''
    ]);
    return json_decode($response->getBody()->getContents());
}
```

**Figura 16: Función generar token**

La función *generarToken* consume el recurso token de la API de WSO2 con URL: <http://localhost:8000/oauth2/token>, recibe como parámetro los datos del usuario que está intentando acceder al api, si el usuario existe y es correcto entonces devuelve en formato JSON los datos del token generado, incluyendo el tiempo de vida y el tipo de token, en este caso Bearer (véase la figura 17).

scope: openid

- Funcion *getUserInfo*

Esta función consume el recurso *userinfo* de la API de WSO2 en la url <http://localhost:8000/oauth2/token>, recibe como parámetro en el header el campo Authorization que contiene la información decodificada del token Bearer, obteniendo todo lo relacionado con los datos del usuario.

- Esta función consume el recurso *userinfo* de la API de WSO2 en la url <http://localhost:8000/oauth2/token>, recibe como parámetro en el header el campo Authorization que contiene la información decodificada del token Bearer, obteniendo todo lo relacionado con los datos del usuario.

```

public static function getUserInfo($token,$request=null)
{
    $url = env('WSO2_URL');
    $client = new Client(['verify' => false]);
    try {
        $response = $client->get($url . '/oauth2/userinfo', [
            'headers' => [
                'Content-Type' => 'application/json',
                'Authorization' => base64_decode($token) . '//token',
                'Accept' => 'application/json'
            ],
            // Trabajo Local con proxy
            // 'proxy' => env('PROXY')
            // Producción sin proxy
            'proxy' => ''
        ]);
        $data = json_decode($response->getBody()->getContents());
        $data->email = $data->sub;

    } catch (\Exception $e) {
        abort($e->getCode(), json_encode($e->getMessage()));
        //dd([$e,$e->getMessage(),$e->getLine()]);
    }
    return $data;
}

```

Figura 18: Función para obtener los datos del usuario autenticado.

- Función *listUsers*

Esta función consume el recurso users de una API externa llamada peril mediante la URL <http://localhost:8001/api/users?application=rcc>, recibe como parámetro en el header el campo Authorization con la información del token y el hashed (token codificado a base64), con esta información y el parámetro application se obtienen todos los usuarios registrados en la aplicación. Solo devuelve la lista de usuarios en el caso que el usuario sea válido y tenga permiso, en caso contrario obtiene código de error 403 Forbidden.

```

function listUsers($app = null)
{
    $client = new Client(['verify' => false]);
    $response = $client->get(env('WSO2_URL') . Constants::WSO2_API . Constants::WSO2_API_PERFIL_USER_INFO . "?application=" . $app, [
        'headers' => [
            'Content-Type' => 'application/x-www-form-urlencoded',
            'Authorization' => 'Bearer ' . $this->accessToken,
            'hashed' => $this->hashed
        ],
        'proxy' => ''
    ]);
    return json_decode($response->getBody()->getContents());
}

```

Figura 19: Función para listar los usuarios del sistema

### *2.6. Conclusiones parciales del capítulo.*

En el capítulo se detallaron las herramientas utilizadas para el desarrollo de API situada en la capa intermedia de la arquitectura de referencia presentada. Se expusieron, a través del diagrama de recursos, todas las rutas definidas en la API que permiten acceder a la información almacenada por la aplicación Bienestar RCC. Se decide la utilización de la plataforma empresarial WSO2 por las soluciones que ofrece para administrar y proteger el acceso a las APIs.

## **CAPÍTULO 3 IMPLEMENTACIÓN DEL SISTEMA PARA EL ANÁLISIS DE LA INFORMACIÓN GENERADA POR LA APLICACIÓN BIENESTAR RCC**

En este capítulo se detallan las herramientas usadas y las interioridades de la programación para la implementación del sistema informático, se describe la estructura del proyecto, así como la estructura interna de los archivos según el framework VueJS, además se definen los plugins utilizados y la interfaz de usuario. Los resultados del análisis de los datos se muestran a través de gráficos o tableros de control ya que estos constituyen herramientas para presentar el estado de estos elementos de la medición corporativa que permiten la comunicación de las estrategias claves, los indicadores e incluso los proyectos de las organizaciones.

El tablero de control es una herramienta gerencial que tiene por objetivo principal presentar el estado actual de uno o varios elementos de la medición (indicadores, planes, estrategias, iniciativas) de la gestión de una compañía, bien sea a nivel global o por cada una de sus áreas o procesos (Fleitman, 2006).

En la aplicación, toda la información que se muestra está soportada en gráficas, semáforos de cumplimiento e íconos que son entendibles para cualquier equipo de trabajo. Se tiene acceso a los indicadores más importantes en una sola pantalla, en un solo sistema, con gráficas interactivas, lo que posibilita la toma de decisiones.

### *3.1. Herramientas utilizadas en el ambiente de desarrollo*

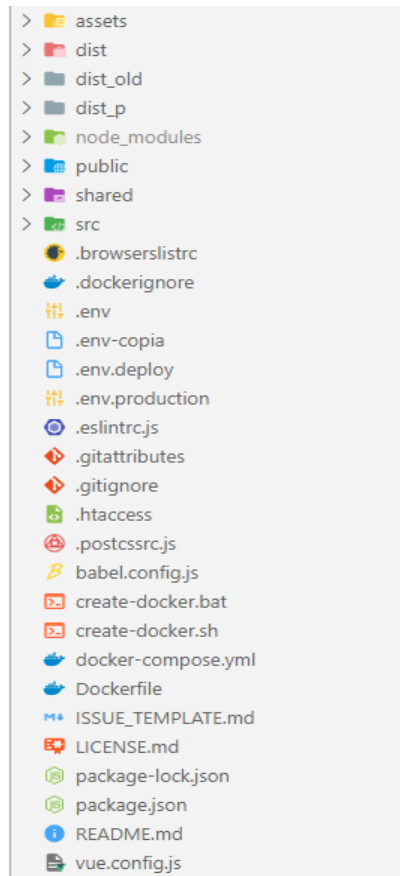
Para la elaboración de la aplicación web se han utilizado las siguientes herramientas:

- Node.js versión 14.0
- Vue.JS con los siguientes plugins:
  - Vuetify.js
  - Vue router
  - ApexChart
  - Axios
  - Vuex
- Visual Studio Code con las siguientes extensiones:
  - ESLint 2.1 Para el correcto formato y estructura del código

- Vue Preview 1.73
- Vetur paquete de extensiones para el trabajo con vue.js

### 3.2. Estructura del proyecto

La estructura de carpetas a nivel raíz del proyecto se muestran a continuación:

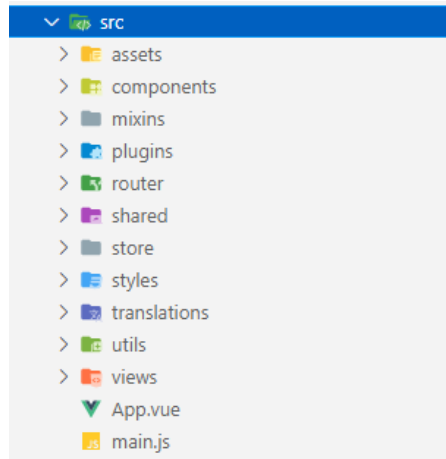


**Figura 20: Estructura del proyecto Vue.js**

En la carpeta *public* se encuentran los archivos públicos de la aplicación. Aquí también estará el punto de entrada el cual es el archivo `index.html`.

En otra carpeta denominada *dist/* se almacenan los archivos finales procesados, como `.js` y `.css`, por ejemplo, que son los archivos procesados finales que irán desplegados en el servidor o web en producción definitiva.

La carpeta *src* es probablemente una de las más importantes. En ella se almacena el código fuente (*source*) de este proyecto. Dentro de *src* se encuentran los archivos originales sin procesar.



**Figura 21: Estructura de la carpeta /src**

Los ficheros *.vue* son los denominados SFC de Vue (Single File Components); se trata de un archivo especial de Vue, muy similar a un *.html* que incluye tres etiquetas HTML especiales: `<template>`, `<script>` y `<style>`.

El fichero *App.vue* es un fichero SFC especial, el componente general e inicial de la aplicación, desde donde se irán cargando los demás componentes.

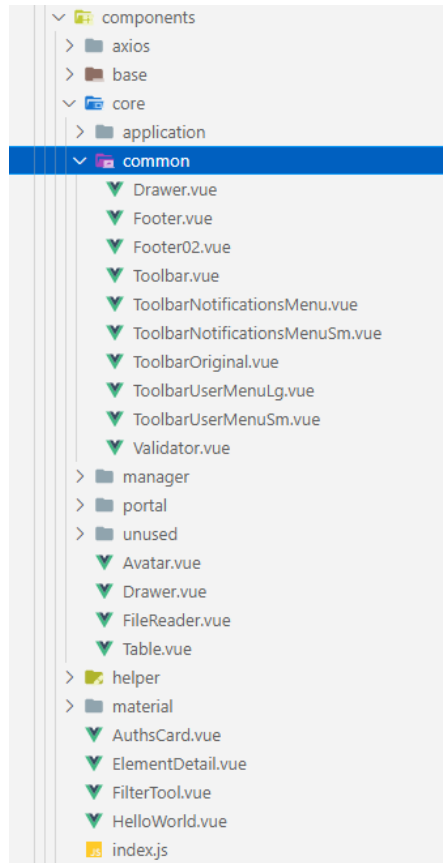
*Main.js* es el fichero principal que arranca el proyecto Vue y que se insertará en la plantilla *index.html* que se incluye en la carpeta *public/*. Este archivo se encargará de cargar *Vue* y todos sus *plugins* asociados. *Main.js* carga el framework y sus *plugins* y lee el fichero SFC *App.vue*, donde comienza a crearse la aplicación *Vue*.

Dentro de la carpeta *src/* se encuentra la carpeta *router*, debido a que se eligió utilizar *Vue Router* en la aplicación para crear rutas desde el frontend. En su interior aparece un archivo *index.js* donde se gestionan las rutas de la aplicación y los componentes que se cargarán.

Vue Router es la biblioteca de enrutamiento oficial del lado del cliente que proporciona las herramientas necesarias para asignar los componentes de una aplicación a diferentes rutas de URL del navegador.



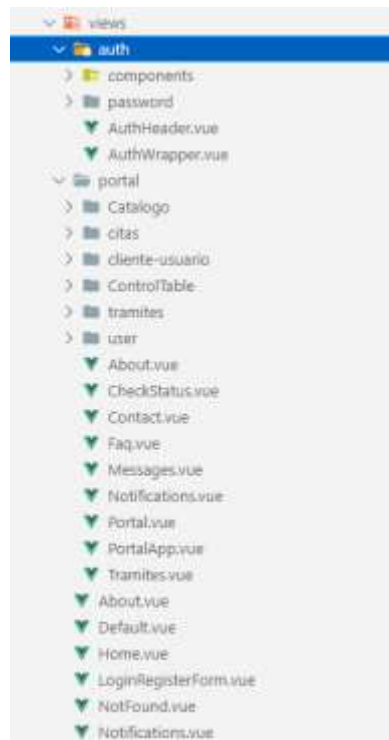
También en el interior de la carpeta *src* se encuentra components, probablemente una de las carpetas más importantes de este proyecto Vue. En ella aparecen los componentes .vue que se crearon durante este proyecto. Los componentes .vue son archivos que contienen el HTML, CSS y Javascript que está relacionado con una determinada parte de la página, como podría ser un botón, un panel desplegable o un comparador de imágenes.



**Figura 22: Contenido de la carpeta components**

Al tener activado Vue Router, aparece la carpeta views donde también se guardan componentes. La diferencia respecto a la carpeta components, es que en views se guardan componentes que definen la estructura general de una página, mientras que en components se guardan partes reutilizables que se utilizan en múltiples lugares de nuestra web.

La siguiente imagen muestra la estructura de la carpeta view del proyecto:



**Figura 23: Estructura de la carpeta views**

### 3.3. Implementación de los ficheros vue

Un archivo .vue se trata de una variación de un fichero .html especialmente diseñada para cubrir las demandas de los componentes de un framework de frontend.

En Vue, por defecto, se suele utilizar un sólo fichero .vue para los componentes, el cual incluye todas las tecnologías relacionadas en él. Es decir, en un sólo fichero se escriben las tres tecnologías principales del frontend: HTML, Javascript y CSS.

```
<template>
  <!-- Código HTML -->
</template>

<script>
  // Código Javascript
</script>

<style>
  /* Código CSS */
</style>
```

La etiqueta `<template>` representa la plantilla que contiene el contenido HTML del componente.

La etiqueta `<script>` contendrá todo el Javascript de Vue relacionado con el componente.

La etiqueta `<style>` donde se indican los estilos CSS implicados con el componente en cuestión.

A continuación se muestra un ejemplo de fichero `.vue` del proyecto, es el fichero “LoginForm.vue” utilizado para mostrar todo lo relacionado con el inicio y el registro del usuario:

```
<template>
  <v-form ...
</v-form>
</template>

<script>
import { isEmail } from '@utils/regex.js'
import {
  login
} from '@components/axios/auth/auth'

export default {
  data: () => ({
    valid: true,
    loading: false,
    passwordHidden: true,

    form: { ...
  },
  emailRules: { ...
  },
  passwordRules: {
    v => !!v || 'La contraseña es requerida'
  },
  remember: false
  }),
  created () {
    this.form.username = this.$route.query.username || null
  },
  methods: { ...
  }
}
</script>
```

Figura 24: Estructura de un archivo `.vue`

Dentro del script se puede encontrar todo lo relacionado con el código de implementación del archivo `.vue`, por ejemplo, sus propiedades y las funciones a ejecutar durante cada ciclo de vida de la página.

Los ciclos de vida más importantes para ayudar al control de la página son:

1. `beforeCreate`: se ejecuta justo después de la inicialización de la instancia.
2. `created`: se ejecuta cuando la instancia y los eventos, las computed properties, el `data` y los métodos están creados. Normalmente se utiliza para inicializar propiedades del objeto `data` con consultas HTTP Get.
3. `beforeMount`: se ejecuta justo antes de que se añada al DOM.
4. `mounted`: se ejecuta después de añadirlo al DOM. Se puede utilizar para inicializar bibliotecas que dependan del DOM.
5. `beforeUpdate`: se ejecuta cuando el `data` cambia, pero el DOM aún no ha plasmado los cambios.
6. `updated`: se ejecuta después de que el `data` cambie y el DOM muestre estos cambios.
7. `beforeDestroy`: se ejecuta justo antes de eliminar la instancia.
8. `destroyed`: se ejecuta cuando la instancia, los eventos, directivas e hijos del componente se han eliminado.

También se muestran otras propiedades importantes dentro de la sección de `<script>` como son:

*Export*: indica que se va a exportar por defecto un objeto que son las opciones del componente Vue (*Vue API Option*). Como mínimo, este objeto de opciones tendrá la propiedad `name` con el nombre que le hemos dado al componente.

*Import*: importar el componente utilizando `import` y haciendo referencia al archivo `.vue` del componente. También puede utilizarse para importar otros archivos externos js y utilizar sus funciones y atributos que se hayan exportado en el archivo js.

La propiedad `components` contendrá un objeto con todos los componentes que se han importado y se desean utilizar en el archivo `.vue`.

En este archivo .vue de ejemplo se importó la función login del archivo auth.js, cuya función es, mediante el plugin de axios, conectarse a nuestra api red-comercial e intentar la autenticación del usuario.

```

1  import axios from 'axios'
2  import { AGENT, SERVER_DIR, CLIENT_ID, CLIENT_SECRET } from '../../utils/constants'
3  const querystring = require('querystring')
4  // import store from '../../store'
5
6  > function auth () { ...
7
8  }
9
10 > function requestDataAuth () { ...
11
12 }
13
14 // Auth login
15 export async function login (data) {
16   const url = `${SERVER_DIR}/oauth2/token`
17
18   console.log('data', data)
19   const response = await axios.post(url,
20     data,
21     requestDataAuth())
22   return response
23 }
24
25 // Auth register
26 > export async function register (data) { ...
27
28 }
29
30 // Get Roles
31 > export async function getRoles () { ...
32
33 }
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

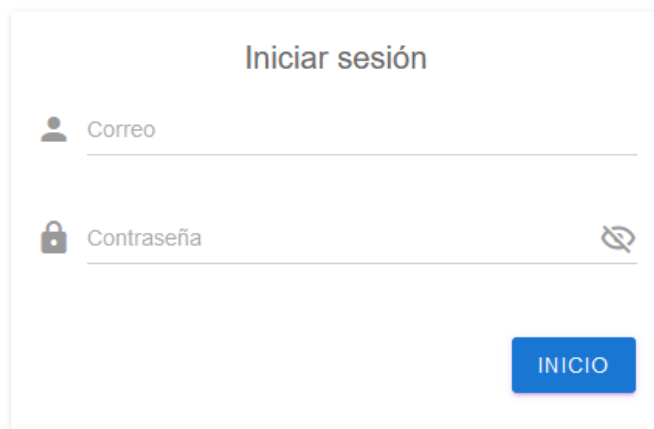
**Figura 25: Implementación del archivo auth.js**

Axios es una biblioteca JavaScript que puede ejecutarse en el navegador y que permite hacer sencillas las operaciones como cliente HTTP, por lo que se puede configurar y realizar solicitudes a un servidor y recibir respuestas fáciles de procesar (Axios, 2021).

### 3.4. Interfaz de usuario

#### 3.4.1. Autenticación

El sistema se presenta ante el usuario con una pantalla de autenticación a través de la cual el usuario podrá introducir su nombre de usuario, contraseña:



**Figura 26: Pantalla de Inicio de sesión**

### 3.4.2. Pantalla principal del sistema

La pantalla principal de la aplicación está compuesta de varias partes y muestra de una sola vez todo el entorno gráfico disponible de manera que el usuario solo se moverá dentro de la aplicación a partir de los elementos que visualiza en esta ventana principal.



**Figura 27: Vista principal de la sección del tablero de control**

A la izquierda de la pantalla aparece el panel de filtros, antes de poder visualizar un gráfico con un conjunto de datos, se le permitirá al usuario la entrada de ciertos parámetros que servirán de filtro para seleccionar los datos que realmente se desean. Luego de seleccionados, el usuario podrá visualizar el gráfico con el contenido de los datos a partir de los criterios seleccionados.

**Filtros**

Lista filtros tablero

País  
Cuba

Provincia  
La Habana

Municipios

Grupos  
Sistema de Alojamiento

Grupos Para Sub Secuencia  
Sistema de Alojamiento,  
Venta de Mercancías

Sub Grupos

Actividad Principal

Organismos

Particularidades

Tipo Moneda

Tipo Organismo  
Ministerio

**Figura 28: Panel de filtros**

En el Panel principal se muestran los gráficos agrupados por varios conceptos:



Figura 29: Sección del comercio minorista

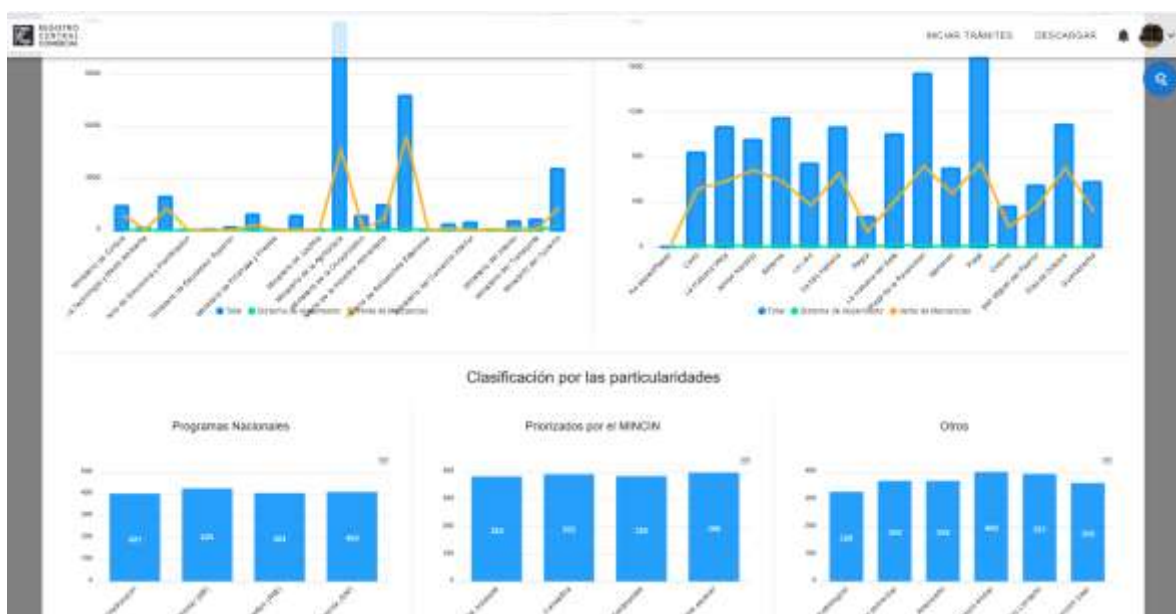
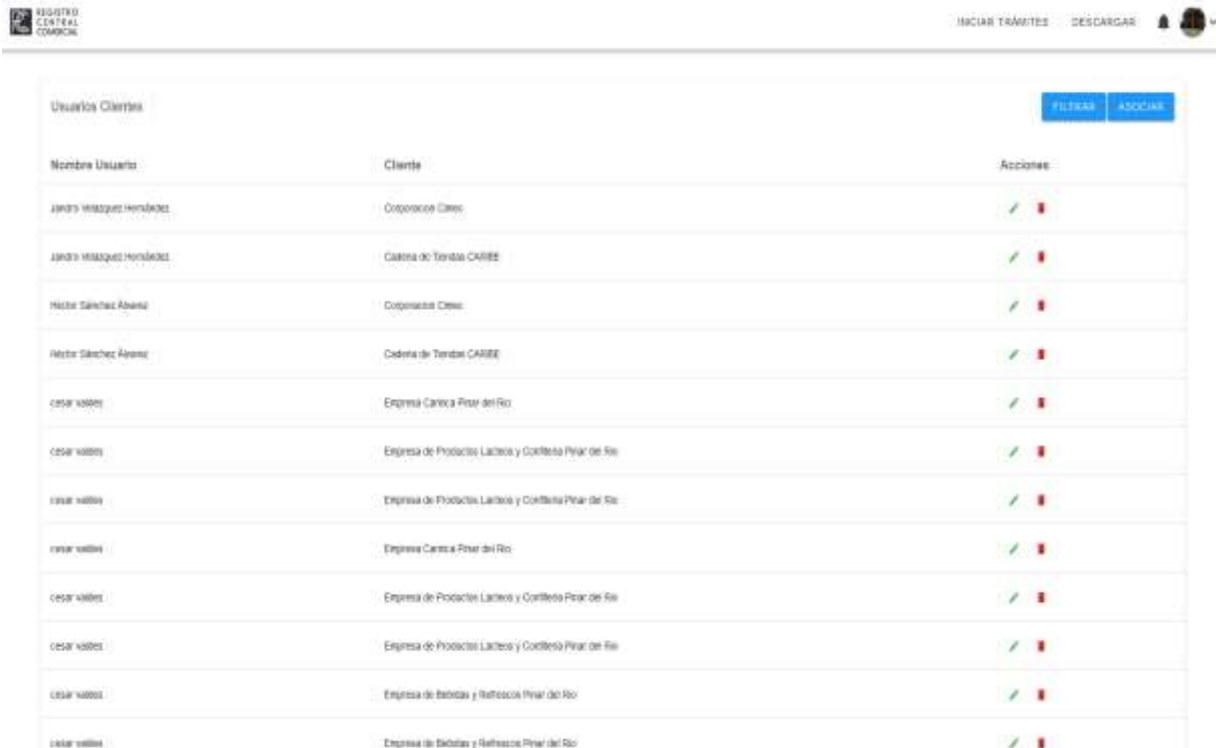


























Figura 30: Clasificación de las instalaciones por particularidades

El usuario puede asociar los usuarios de la aplicación con los clientes del sistema, como las entidades y UEB. Esto posibilitará que los representantes de las entidades una vez registrados en el sistema puedan ser autorizados por los especialistas de las filiales a visualizar los datos relacionados con los autorizos comerciales de su entidad.





Nombre Usuario	Cliente	Acciones
JAVIER VILLALBA HERNÁNDEZ	Corporación Ceres	 
JAVIER VILLALBA HERNÁNDEZ	Cadena de Tienda CARIBE	 
Néstor Sánchez Álvarez	Corporación Ceres	 
Néstor Sánchez Álvarez	Cadena de Tienda CARIBE	 
cesar valdez	Empresa Cereza Pinar del Río	 
cesar valdez	Empresa de Productos Lácteos y Condensados Pinar del Río	 
cesar valdez	Empresa de Productos Lácteos y Condensados Pinar del Río	 
cesar valdez	Empresa Cereza Pinar del Río	 
cesar valdez	Empresa de Productos Lácteos y Condensados Pinar del Río	 
cesar valdez	Empresa de Productos Lácteos y Condensados Pinar del Río	 
cesar valdez	Empresa de Bebidas y Refrescos Pinar del Río	 
cesar valdez	Empresa de Bebidas y Refrescos Pinar del Río	 

**Figura 31: Listado de los usuarios asociados a los clientes (entidades o UEB)**

Además el usuario puede visualizar estadísticas de los clientes que tiene asociado, puede realizar comparaciones entre autorizos pendientes a vencer, vencidos y renovados así como el precio total a pagar por renovar los autorizos en las diferentes monedas.



Figura 32: Tablero de control del proceso registral de la red comercial

### 3.5. Conclusiones parciales del capítulo.

En el capítulo se listan las tecnologías, lenguajes y herramientas utilizadas para el desarrollo de la aplicación web. Se describe la estructura de un proyecto desarrollado usando VueJS y se muestra la interfaz del usuario; se destaca la utilización de gráficos por su utilidad para la ayuda a la toma de decisiones.

## CONCLUSIONES

1. Laravel es el framework seleccionado para la creación del Api REST pues este framework PHP es apropiado para el desarrollo de softwares y aplicaciones desde cero, hace que la dinámica de trabajo sea más fluida, permite reducir plazos y partir de una base sólida y elegante.
2. La Api REST garantiza escalabilidad, flexibilidad y portabilidad. Debido a la separación entre el cliente y el servidor, este protocolo facilitó el desarrollo de las diferentes partes del proyecto de manera independiente. Además, se adaptó en todo momento al tipo de sintaxis o plataforma de trabajo. Esto brindó la oportunidad de probar varios entornos dentro del desarrollo.
3. WSO2 permitió construir una base digital para el desarrollo y gestión de APIs externas e internas, así como un aumento de la agilidad interna generando activos digitales. Se aprovechó la estructura digital anterior y la protección de las posibles amenazas externas.
4. ApexCharts es una de las bibliotecas de visualización de JavaScript que permitió presentar datos estadísticos en gráficos SVG sensibles, elegantes y precisos. Su combinación con Vue.js mejor las características del sitio web.
5. La implementación de este sistema ayudará a la toma de decisiones sobre todo lo relacionado con la red comercial, brindando la información de forma segura, confiable y oportuna.

## **RECOMENDACIONES**

1. El desarrollo de una aplicación para dispositivos móviles que haga uso de la API REST para la comunicación con el sistema Bienestar RCC permitirá el análisis de los datos en un ambiente versátil.
2. La continuación del desarrollo de la aplicación web con funcionalidades de análisis geoespacial de los datos puede ser interesante y promisorio.

## REFERENCIAS BIBLIOGRÁFICAS

- Alonso, S. (2015) «Development of a Restful Api». Disponible en: <http://www.theseus.fi/handle/10024/96804> (Accedido: 18 de noviembre de 2021).
- Axios (2021) *Getting Started | Axios Docs*. Disponible en: <https://axios-http.com/docs/intro> (Accedido: 18 de noviembre de 2021).
- Castillo, P. A. *et al.* (2011) «SOAP vs REST: Comparing a master-slave GA implementation». Disponible en: <https://arxiv.org/abs/1105.4978v1> (Accedido: 18 de noviembre de 2021).
- Chen, X. *et al.* (2017) «Restful API Architecture Based on Laravel Framework», *Journal of Physics: Conference Series*, 910(1). doi: 10.1088/1742-6596/910/1/012016.
- Clarke, S. (2004) «Measuring API Usability», *Dr. Dobbs's Journal*. Disponible en: <http://www.drdobbs.com/windows/measuring-api-usability/184405654> (Accedido: 14 de noviembre de 2021).
- Denzel Javier Ovando Ortega (2019) «Bootstrap y Laravel, herramientas para el desarrollo de aplicaciones web», *Αγαη*, 8(5), p. 55.
- Fleitman, J. (2006) «La importancia de los tableros de control», XXXIII(Cmi), pp. 1-4.
- Group, T. P. (2016) *History of PHP and Related Projects, Php Documentation*. The PHP Group. Disponible en: <http://www.php.net/history> (Accedido: 15 de noviembre de 2021).
- Jin, B., Sahni, S. y Shevat, A. (2018) *Designing Web APIs*. O'Reilly Media. Disponible en: [https://www.google.com/books/edition/Designing\\_Web\\_APIs/Dg1rDwAAQBAJ](https://www.google.com/books/edition/Designing_Web_APIs/Dg1rDwAAQBAJ) (Accedido: 14 de noviembre de 2021).
- Laravel (2015) «Laravel Documentation», 30/11/2015, p. 1. Disponible en: <http://web.archive.org/web/20140920185439/http://laravel.com/docs/introduction>.
- Laravel (2021) *Laravel Documentation - The PHP Framework For Web Artisans*. Disponible en: <https://laravel.com/docs/8.x/controllers> (Accedido: 18 de noviembre de 2021).
- Liew, Z. (2018) «Car Service APIs Are Everywhere, But What's In It For Partner Apps», *Smashing Magazine*. Disponible en: <https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/> (Accedido: 14 de

noviembre de 2021).

Luca (2019) «¿Qué es Python?», *LUCA*. Disponible en: <https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje> (Accedido: 15 de noviembre de 2021).

Miguel, L. y Hernández (2021) «Arquitectura REST para el desarrollo de aplicaciones web empresariales REST architecture for enterprise web application development», 8.

Musciano, C. y Kennedy, B. (2000) *HTML & XHTML : The Definitive Guide 4th edition*.

Nelson, B. (2018) *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*. Apress. Disponible en: <https://books.google.com/books?id=mTpsDwAAQBAJ> (Accedido: 14 de noviembre de 2021).

Palvia, S. C. J., Palvia, S. C. J. y Sharma, S. S. (2007) *E-Government and E-Governance: Definitions/Domain Framework and Status around the World*. ICEG. Disponible en: [http://www.iceg.net/2007/books/1/1\\_369.pdf](http://www.iceg.net/2007/books/1/1_369.pdf) (Accedido: 14 de noviembre de 2021).

Pantoja, L. y Pardo, C. (2016) «Evaluando la Facilidad de Aprendizaje de Frameworks mvc en el Desarrollo de Aplicaciones Web», *Publicaciones e Investigación*, 10, p. 129. doi: 10.22490/25394088.1592.

Reddy, M. (2011) *API design for c++*. 1.<sup>a</sup> ed, *API Design for C++*. 1.<sup>a</sup> ed. Elsevier Science. doi: 10.1016/C2010-0-65832-9.

«Routing - Documentation Laravel PHP Framework» (2021) <http://laravel.com>. Disponible en: <http://laravel.com/docs/routing> (Accedido: 14 de noviembre de 2021).

Schmelzer, J. (2019) «NET 2015 Overview», *Channel 9*, p. 0: 07: 32. Disponible en: <https://channel9.msdn.com/Events/Visual-Studio/Connect-event-2015/NET-2015-Overview> (Accedido: 15 de noviembre de 2021).

*Secure APIs using OAuth2 Access Tokens - WSO2 API Manager Documentation 3.0.0* (2020). Disponible en: <https://apim.docs.wso2.com/en/3.0.0/learn/api-security/api-authentication/secure-apis-using-oauth2-tokens/> (Accedido: 18 de noviembre de 2021).

So, P. (2018) «Vue.js», en *Decoupled Drupal in Practice*. Apress, Berkeley, CA, pp. 381-397. doi: 10.1007/978-1-4842-4072-4\_20.

De Souza, I. (2020) *API Rest: ¿qué es y cómo funciona ese recurso?* Disponible en: <https://rockcontent.com/es/blog/api-rest/> (Accedido: 14 de noviembre de 2021).

Toledo, R. M. (2016) «Ciberciudadanía y Gobierno Electrónico», pp. 1-52.

Www.w3schools.com (2020) *What is Vue.js*, *www.w3schools.com*. Disponible en: [https://www.w3schools.com/whatis/whatis\\_vue.asp](https://www.w3schools.com/whatis/whatis_vue.asp) (Accedido: 14 de noviembre de 2021).

Yucabyte (2021) *Plataforma «Bienestar» afianza informatización militar de la sociedad*, *Yucabyte*. Disponible en: <https://www.yucabyte.org/2021/05/19/plataforma-bienestar-militar/> (Accedido: 14 de noviembre de 2021).