Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Domingo Ottoniel Guarchaj Catinac
Asesorado por el Ing. Daniel Estuardo Barrientos López

Guatemala, febrero de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

DOMINGO OTTONIEL GUARCHAJ CATINAC
ASESORADO POR EL ING. DANIEL ESTUARDO BARRIENTOS LOPEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, FEBRERO DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Kevin Vladimir Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADORA	Inga. Floriza Felipa Ávila Pesquera de Medinilla
EXAMINADOR	Ing. Sergio Leonel Gómez Bravo
EXAMINADOR	Ing. Carlos Alfredo Azurdia Morales
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 10 de marzo de 2021.

Domingo Ottoniel Guarchaj Catinac

Guatemala, 22 de noviembre de 2021

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería en Ciencias y Sistemas
Universidad de San Carlos de Guatemala

Estimado Ingeniero Alonzo:

Respetuosamente me dirijo a usted deseándole éxito y bendiciones en sus actividades cotidianas.

El motivo de la presente es para informarle que el estudiante **DOMINGO OTTONIEL GUARCHAJ CATINAC** quien se identifica con carné **201020975** y CUI **2059 78339 0705**, ha finalizado de manera satisfactoria el informe final del proyecto de EPS:

"DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA"

Cualquier información adicional, estoy a las órdenes a través del correo: daniel.barrientoslopez@gmail.com. Sin otro particular me suscribo, atentamente.

,

Daniel Estuardo Barrientos Lopez
Ingeniero en Ciencias y Sistemas
Colegiado No. 17152

Ing. Daniel Estuardo Barrientos Lopez
Asesor de Escuela
Colegiado No. 17152

Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Unidad de EPS

Guatemala, 19 de noviembre de 2021.
REF.EPS.DOC.482.11.2021.

Ing. Oscar Argueta Hernández
Director Unidad de EPS
Facultad de Ingeniería
Presente

Estimado Ingeniero Argueta Hernández:

Por este medio atentamente le informo que como Supervisora de la Práctica del Ejercicio Profesional Supervisado, (E.P.S) del estudiante universitario de la Carrera de Ingeniería en Ciencias y Sistemas, **Domingo Ottoniel Guarchaj Catinac, Registro Académico 201020975 y CUI 2059 78339 0705** procedí a revisar el informe final, cuyo título es **DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA.**

En tal virtud, **LO DOY POR APROBADO**, solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,

"Id y Enseñad a Todos"

Universidad de San Carlos de Guatemala
COORDINADOR(A) EPS
ÁREA TECNOLOGÍA Y CIENCIAS
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

Inga. Floriza Felipa Ávila Pesquera de Medinilla
Supervisora de EPS
Área de Ingeniería en Ciencias y Sistemas

FFAPdM/RA

Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Unidad de EPS

Guatemala, 19 de noviembre de 2021.
REF.EPS.D.261.11.2021.

Ing. Carlos Gustavo Alonzo
Director Escuela de Ingeniería Ciencias y Sistemas
Facultad de Ingeniería
Presente

Estimado Ingeniero Alonzo:

Por este medio atentamente le envío el informe final correspondiente a la práctica del Ejercicio Profesional Supervisado, (E.P.S) titulado **DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** que fue desarrollado por el estudiante universitario **Domingo Ottoniel Guarchaj Catinac, Registro Académico 201020975 y CUI 2059 78339 0705** quien fue debidamente asesorado por el Ing. Daniel Estuardo Barrientos López y supervisado por la Inga. Floriza Felipa Ávila Pesquera de Medinilla.

Por lo que habiendo cumplido con los objetivos y requisitos de ley del referido trabajo y existiendo la aprobación del mismo por parte del Asesor y la Supervisora de EPS, en mi calidad de Director apruebo su contenido solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,
"Id y Enseñad a Todos"

Universidad de San Carlos de Guatemala
DIRECCIÓN
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

Ing. Oscar Argueta Hernández
Director Unidad de EPS

/ra

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 22 de noviembre de 2021

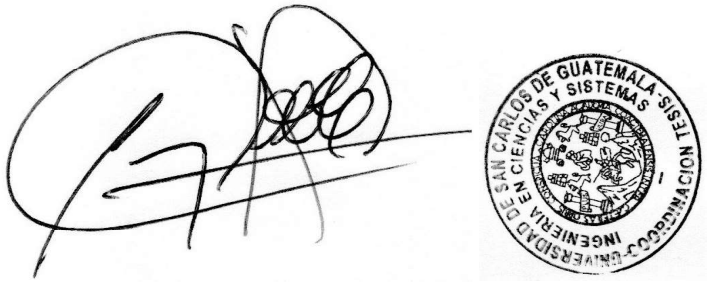
Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación-EPS del estudiante **DOMINGO OTTONIEL GUARCHAJ CATINAC** carné **201020975** y CUI **2059 78339 0705,** titulado: **"DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

LNG.DIRECTOR.024.EICCSS.2022

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**, presentado por: **Domingo Ottoniel Guarchaj Catinac**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

Ing. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, febrero de 2022

Facultad de Ingeniería

Decanato
24189101-
24189102
secretariadecanato@ingenieria.usac.edu.gt

LNG.DECANATO.OI.066.2022

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **DISEÑO DE ARQUITECTURA ORIENTADA A SERVICIOS Y CREACIÓN DE MÓDULO DE SEGUIMIENTO A CASOS ESTUDIANTILES PARA EL PORTAL DE POSTGRADO DE LA FACULTAD DE INGENIERÍA, DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** presentado por: **Domingo Ottoniel Guarchaj Catinac,** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Aurelia Anabela Cordova Estrada

Decana

Guatemala, febrero de 2022

AACE/gaoc

ACTO QUE DEDICO A:

Dios	Por todas las bendiciones en la vida.
Mis padres	Juan Rodrigo Guarchaj y Manuela Catinac Perechú, por su confianza, amor y sus inmensos esfuerzos por darme la oportunidad de cumplir esta meta en la vida.
Mis hermanos y hermanas	Por su paciencia, confianza y apoyo incondicional para lograr alcanzar este objetivo durante todo este tiempo. Especialmente a mi hermano German Guarchaj, por las buenas enseñanzas que me ha dejado en la vida.
Mi sobrino	Dominick Guarchaj, por llegar y volverse una motivación en mi vida.
Toda mi familia y amigos.	Por el apoyo, comprensión y motivación que siempre me brindaron, y por todas esas experiencias vividas que me han hecho crecer como persona.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por darme la oportunidad de estudiar una carrera en sus instalaciones. Y por ser mi segunda casa.
Facultad de Ingeniería	Por brindarme todos los conocimientos para poder desarrollarme como un profesional en la Ingeniería.
Mis amigos de la Facultad	Por su apoyo cuando los necesite, por su confianza en mí cuando fue necesario, por todos esos desvelos que pasamos, por su gran amistad y por todas esas experiencias vividas.
Mis asesores	Ing. Daniel Barrientos e Ing. Luis Cajas, por guiarme a través de todo el proceso de desarrollo de EPS y por toda esa disposición que me brindaron en todo momento.
Escuela de Estudios de Postgrado de la Facultad de Ingeniería.	Por darme la oportunidad de realizar mi Ejercicio Profesional Supervisado.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
API	*Application Programming Interfaces*
%	Porcentaje
Q	Quetzales
SSL	*Secure Sockets Layer*
TLS	*Transport Layer Security*

GLOSARIO

Access Token	En los sistemas informáticos, un *token* de acceso contiene las credenciales de seguridad para una sesión de inicio de sesión que identifica a un cliente.
API RESTful	Interfaz de programación de aplicación, es el mecanismo más útil para conectar dos *softwares* entre sí para el intercambio de mensajes o datos en formato estándar como XML o JSON.
Base de datos	Lugar donde se almacenan datos de manera estructurada sistemáticamente para su pronta recuperación.
Boot	Es un programa informático que efectúa automáticamente tareas reiterativas mediante internet a través de funciones autónomas.
Bootstrap	Conjunto de elementos que facilitan la creación de los componentes gráficos de una página web.
Caso de uso	Describe como un actor interactúa con un sistema, un actor puede ser una persona, proceso o sistema.

Composer	Es un sistema de gestión de paquetes para programar en PHP el cual provee los formatos estándar necesarios para manejar dependencias y librerías de PHP.
EEP	Siglas que significan: Escuela de Estudios de Postgrado de la Facultad de Ingeniería, Universidad de San Carlos de Guatemala.
Framework	Se refiere a un entorno de trabajo, que es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver problemas similares.
GitLab	Es un servicio web de control de versiones y desarrollo de *software* colaborativo. Además, ofrece también un sistema de seguimiento de errores.
Hardware	Conjunto de elementos físicos y tangibles de un sistema informático.
Issues	Es una de las grandes utilidades de GitLab, que representa la definición de cualquier problema que se detecta en el *software* y poder darle seguimiento, mantener discusiones sobre ellas y controlar los flujos de trabajo para su resolución.

Kernel	Es el núcleo o parte fundamental de un *software* y se ejecuta en modo privilegiado.
Laravel	Es un *framework* de código abierto para desarrollar aplicaciones y servicios web con PHP 5 y PHP 7.
Middleware	Es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware o sistemas operativos.
OAuth	*Open Authorization* es un estándar abierto que permite flujos de autorización para aplicaciones de escritorio, móviles y web.
PHP	Es un lenguaje de programación orientado a objetos para el desarrollo web del lado del servidor.
Refresh Token	Es un tipo de *token* que permite actualizar el tiempo de vida de un *access token*, esto con el fin de extender el tiempo de autorización para el acceso a determinado recurso.
Redis	Es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes pero que opcionalmente puede ser usada como una base de datos durable o persistente.

Singleton	En ingeniería de *software*, *singleton* o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.
Software	Programa de computadora o rutina que permite a un sistema informático realizar determinada tarea.
Token	Es una cadena de caracteres con una sintaxis definida que permite identificar de manera única un registro en una cola de registros.
Timeline	Una línea de tiempo es un elemento útil para establecer las etapas de los procesos históricos y que permite dar seguimiento en el tiempo para una auditoría.
TLS	Seguridad de la capa de transporte, es una versión mejorada de SSL, funcionando de un modo muy parecido, utilizando cifrado que protege la transferencia de datos e información.
Ventana Modal	Una ventana modal en el diseño de interfaces de usuario para aplicaciones informáticas, es un elemento de control grafico subordinado a la ventana principal de una aplicación.

RESUMEN

Este proyecto permitió el escalamiento del sistema actual de la institución, permitiendo así que los servicios actuales puedan ser consumidos tanto por el portal web de la institución, como para futuras aplicaciones móviles de la institución, con el fin de extender su alcance en proyectos futuros.

Se configuró una capa de seguridad para proteger los servicios de ataques mediante un servicio de autenticación a nivel de usuario y aplicación utilizando *tokens*, donde se protegerán los servicios para que sean consumidos únicamente por aplicaciones y usuarios que pertenezcan a la Institución, y los cuáles sean permitidos.

También, se trabajó en extender las funcionalidades del portal público y administrativo, adaptando un nuevo módulo para la atención y seguimiento de casos estudiantiles. Se tendrá la capacidad de atender estas nuevas funcionalidades a través del portal actual de la institución, así como también, soportará el consumo de estos mismos servicios a través de aplicaciones móviles en proyectos futuros.

OBJETIVOS

General

Desarrollar en la Escuela de Estudios de Postgrado un módulo que le permita a la institución centralizar y agilizar la atención de los casos estudiantiles, y al mismo tiempo crear un nuevo diseño de arquitectura de servicios para la institución.

Específicos

1. Centralizar mediante la creación de un nuevo módulo, la atención de casos estudiantiles y reducir en un 30 % el tiempo de respuesta y solución a los casos, del que se lleva actualmente.

2. Crear e integrar una arquitectura orientada a servicios a la actual infraestructura de proyectos de la institución, para poder habilitar la opción de poder migrar el resto de sus proyectos a uno con soporte a aplicaciones móviles de forma segura.

3. Definir los nuevos estándares derivados del nuevo diseño de la arquitectura de servicios y documentar las bases a seguir para adaptar nuevas funcionalidades en proyectos futuros enfocados en aplicaciones móviles y web.

INTRODUCCIÓN

Cuando se crea un sistema informático, este debe solucionar en su totalidad los problemas para los que fue creado, por lo tanto, con el transcurso del tiempo, el sistema debe tener mantenimiento para poder seguir cumpliendo con los objetivos por el cual fue creado. Para ello se debe llevar tareas de análisis del sistema; este proceso determina si un sistema informático sigue cumpliendo con los objetivos para los cuales fue creado. Los factores que pueden cambiar los objetivos son: nuevas reglas de negocio de la institución, modificaciones a las normativas existentes, entre otros.

La Escuela de Estudios de Postgrado se ha encontrado con la necesidad de ampliar las funcionalidades de su sistema actual:

- Crear un servicio de autorización, a través del uso del estándar OAuth2.0 mediante el uso de *tokens*, utilizando una nueva base de datos no relacional, con tal de reducir las peticiones o accesos a la base de datos relacional de PostgreSQL existente.

- Creación de una API RESTful capaz de atender peticiones a través de distintas aplicaciones de la institución, especialmente enfocados en atender tanto las aplicaciones web existentes, como para el soporte de aplicaciones móviles que la institución desea desarrollar próximamente.

- Creación de un nuevo módulo para la centralización del ingreso y atención de casos estudiantiles. La primera parte consiste en el ingreso de los casos estudiantiles por los estudiantes o aspirantes. La segunda parte se enfoca

en la atención y seguimiento de estos casos del lado del personal administrativos que involucra diferentes niveles de jerarquía.

Todos los requerimientos servirán para extender las funcionalidades de los servicios de la institución, así como también el permitir a la institución el escalamiento de sus aplicaciones y servicios.

1. FASE DE INVESTIGACIÓN

La Escuela de Estudios de Postgrado que pertenece a la Facultad de Ingeniería, la cual es una de las 10 facultades de la Universidad de San Carlos de Guatemala; atiende en promedio a una población de 1 700 estudiantes de postgrado.

Dicha institución cuenta con 7 programas de especialización, 15 programas de maestría y 2 de doctorados. Su sede está en la Ciudad Universitaria zona 12 de la ciudad de Guatemala, en el primer nivel del edificio S-11.

1.1. Antecedentes de la empresa

La Escuela de Estudios de Postgrado es la encargada de continuar los estudios de pregrado, para lo cual pone a disposición programas de postgrado, maestrías y doctorados. Además, cuenta con programas de pregrado, postgrado, este programa consiste en que el estudiante tiene derecho a cursar una maestría por un año mientras completa su graduación a nivel licenciatura.

1.1.1. Reseña histórica

La Escuela de Estudios de Postgrado fue fundada por la Coordinadora, General del Sistema de Estudios de Postgrado, según el punto cuarto de la nota EPFI-187-2013, acta 03-2003 y el punto quinto, inciso 5.9 del acta 16-2013 de la Junta Directiva de la Facultad de Ingeniería, sesión celebrada el 29 de mayo de 2003.

La Escuela de Estudios de Postgrado ofrece a profesionales con el grado de licenciatura egresados de la USAC, universidades privadas o extranjeras catalogadas de primera clase, la oportunidad de actualizar sus conocimientos y diversificar sus campos de actividad profesional para contribuir a la formación de docentes e investigadores de nivel superior.

1.1.2. Misión

"Formar profesionales de la Ingeniería y áreas afines a nivel de postgrado con alta capacidad técnica e investigativa para contribuir al desarrollo científico y tecnológico del país, de manera ética y coherente con la realidad nacional".[1]

1.1.3. Visión

"Ser innovadores de la formación profesional a nivel de postgrado relacionada con la práctica profesional de la ingeniería, gestionando la excelencia y pertinencia social de la investigación que contribuya a solucionar la problemática nacional".[2]

1.1.4. Servicios que realiza

Entre los servicios que brinda la Escuela de Estudios de Postgrado de la Facultad de Ingeniería está: impartir especializaciones, maestrías y doctorados.

A continuación, se presenta una lista de todas las especializaciones, maestrías y doctorados que están disponibles hasta la fecha del día 15 de noviembre de 2021.

[1] *Misión y visión, Escuela de Estudios de Postgrado FIUSAC.* https://postgrado.ingenieria.usac.edu.gt/nosotros. Consulta: 04 de mayo de 2021.
[2] ibid

- Especializaciones
  - Bioinformática y Biocomputación Molecular Biomédica
  - Gestión del Talento Humano
  - Sistemas de Gestión de la Calidad
  - Investigación Científica
  - Administración y Mantenimiento Hospitalario
  - Educación Virtual para el Nivel Superior
  - Seguros y Ciencias Actuariales

- Maestrías
  - Estructuras
  - Ingeniería Geotécnica
  - Geomática
  - Gestión de Recursos Hidrogeológicos
  - Gestión Industrial
  - Ingeniería de Mantenimiento
  - Ingeniería para el Desarrollo Municipal(B-*learning*)
  - Energía y Ambiente
  - Ingeniería Vial
  - Tecnologías de la Información y la Comunicación
  - Estadística Aplicada
  - Ciencia y Tecnología de los Alimentos
  - Gestión de Mercados Eléctricos Regulados
  - Gestión de la Planificación para el Desarrollo
  - Programas con el convenio del proyecto MEANING por la Unión Europea con el programa ERASMUS+

- Doctorados
  - Cambio Climático y Sostenibilidad

- Doctorado en Investigación con los énfasis:
  - Ingeniería y Ciencias Aplicadas
  - Tecnología
  - Ciencias Naturales
  - Desarrollo Social
  - Ciencias de la Salud
  - Ciencias Sociales y Humanísticas

1.2. Descripción de las necesidades

Las necesidades encontradas y mencionadas por parte del personal de la Escuela de Estudios de Postgrado que abarcan este proyecto se dividen en 3 partes.

- Se desea la creación de un nuevo servicio de autorización mediante el uso de *access tokens*, de tal manera que sea utilizada como una nueva capa de seguridad para todas las aplicaciones pertenecientes y autorizadas por la institución. Este servicio debe tener la capacidad de seguir creciendo e ir adaptándose a nuevas aplicaciones.

- Se desea diseñar una nueva arquitectura orientada a servicios que pueda soportar el consumo de los recursos de la institución a través de diferentes tipos de aplicaciones, incluyendo los sitios web actualmente existentes, hasta aplicaciones móviles las cuales desea desarrollar la institución en futuros proyectos. De tal manera que se pueda centralizar el resto de todos los recursos de la institución a un proyecto existente que la institución tiene creada con el nombre de *CORE*, el cual por ahora no cuenta con todas las nuevas características.

- Se desea crear un nuevo módulo de *software* que permita la centralización de la atención de casos estudiantiles, con el fin de reducir el tiempo de atención de estos. Dicho modulo tendrá la capacidad de darle seguimiento a estos casos mediante el desarrollo de un *timeline* a través de la creación de *tokens* que permitan identificar de manera única un caso. El módulo debe involucrar vistas del lado del estudiante y del lado del personal administrativo.

1.3. Priorización de las necesidades

Con el objetivo de llevar a cabo la centralización de los servicios de la institución, incluyendo los recursos nuevos del nuevo módulo de atención de casos estudiantiles. Se debe primero enfocar en la creación y solución de la nueva capa de seguridad a través de un nuevo servicio de autorización, ya que, al llegar al desarrollo del nuevo módulo de atención de casos estudiantiles, estos deben estar protegidos con la nueva capa de seguridad, así también se debería ya tener el nuevo diseño de arquitectura orientada a servicios, debido a que el nuevo módulo debe ser adaptado a este nuevo diseño de arquitectura. Por lo tanto, después de la creación del nuevo servicio de autorización, debe ser diseñado la nueva arquitectura de servicios y por último el desarrollo del nuevo módulo de atención de casos estudiantiles.

2. FASE TECNICO PROFESIONAL

Actualmente, se vive en un mundo donde todo está conectado, desde compartir música en Spotify con amigos de Facebook, hasta compartir registros de salud con el médico personal, desde hacer pagos en línea, hasta twittear sobre compras en Amazon. Todo esto es posible gracias a la creación de API's. Las API's crean nuevas oportunidades de negocios, aumentando el tamaño del mercado y generando más ingresos, en la actualidad casi todos los segmentos de servicios en línea tienen publicado API's. Sin embargo, las violaciones de seguridad se anuncian cada semana y uno de los desafíos que los desarrolladores quieren ver resuelto es la seguridad.

2.1. Descripción del proyecto

La primera parte del proyecto consiste en la creación de una capa de seguridad para proteger los recursos de la institución, mediante la creación de un nuevo proyecto destinado a ser un servidor de autorización a nivel de aplicaciones, esto a través del uso de *tokens* de acceso, donde se protegerán los recursos para que sean consumidos únicamente por aplicaciones clientes que tengan un tipo de relación con la Institución, y los cuáles sean permitidos y autorizados.

Seguidamente, se harán todas las configuraciones necesarias para la creación de una API RESTful, que permitirá el escalamiento del sistema actual de la institución hacia el soporte del consumo de sus recursos a través de aplicaciones móviles, sin dejar a un lado el consumo a través de los sitios web existentes en funcionamiento.

Por último, se trabajará en extender las funcionalidades del portal público y administrativo de la institución, adaptando un nuevo módulo a construir que será dedicado para la atención y seguimiento de casos estudiantiles. Se tendrá la capacidad de atender estas nuevas funcionalidades a través del portal actual de la institución, así como también, soportará el consumo de estos mismos recursos a través de aplicaciones móviles en proyectos futuros.

2.2. Investigación preliminar para la solución del proyecto

A continuación, se menciona una lista de tecnologías utilizadas para dar solución a las necesidades de la Institución, las cuales fueron investigadas de manera teórica y posteriormente llevadas a la práctica.

2.2.1. OAuth2.0

Con frecuencia, cuando se habla de seguridad se menciona dos palabras claves que son completamente diferentes en cuanto a los ámbitos a los cuales se refiere, la autenticación y la autorización.

La autenticación se refiere al proceso mediante el cual se verifica quién eres, o en otras palabras es un proceso que se refiere a la identificación. La autorización en cambio es el proceso mediante el cual se verifica a que se tiene acceso, o en otras palabras es un proceso que se enfoca al control de acceso.

Atendiendo a estos conceptos, OAuth es un *framework* estandarizado que permite delegar la autorización de acceso a las API’s y no es un protocolo de autenticación. Esta afirmación incluso lo informan en su documentación oficial.

Por tanto, el uso de este estándar ayuda a poder conseguir un equilibrio muy importante entre la seguridad y la usabilidad en cuanto a las estrategias de protección de API's, ya que una estrategia demasiado compleja disuade a usuarios malintencionados, pero también obstaculiza su uso por parte de usuarios legítimos.

Las ventajas de uso de este *framework* pueden ser:

- El acceso es limitado, tanto en tiempo porque los *tokens* caducan, como en funcionalidad gracias a los *scopes*.

- Permite compartir datos sin tener que revelar información personal.

- Es un protocolo muy flexible que se basa en SSL (*Secure Sockets Layer*) que cifra las conexiones y permite manejar el *access token* de forma segura.

2.2.2. ¿Qué no utilizar para proteger nuestras API's?

- OAuth 1.0: La versión 2.0 introdujo como mejoras la inclusión de múltiples dispositivos (browsers, nativos, servidores, entre otros) y sobre todo se sustituyó la firma de las peticiones por TLS.

- Autenticación básica: Solo autentica, no gestiona la autorización y las credenciales incluida la contraseña son fácilmente obtenibles.

- Métodos customizados: No es necesario reinventar el agua azucarada, OAuth es un estándar suficientemente testeado y confiable como para usarse, además los desarrolladores ya están habituados a su uso.

2.2.3. ¿Cuáles son los flujos de OAuth y cuándo los uso?

Como primer paso antes de poder entender los flujos de OAuth y los escenarios posibles de aplicación es comprender algunos conceptos como son: los actores que intervienen en los flujos, los *tokens* y los *scopes* de acceso.

2.2.3.1. Concepto: *scopes*, *tokens* y actores

Los *scopes* son identificadores que permiten determinar a qué se solicita acceso por parte de las aplicaciones o a que se autoriza acceder por parte del usuario.

Los *tokens* e identificadores que se manejan en OAuth son:

- El *client id* es un identificador único que representa la aplicación en el servidor de autorización.

- El *client secret* es una clave secreta que pertenece a la aplicación que es utilizada por el servidor de autorización para comprobar si dicha aplicación está autorizada.

- Los *access token* son claves proporcionados por el servidor de autorización a la aplicación y que le permiten el acceso a la API durante un tiempo limitado de validez.

- Los *refresh token* son claves que también son proporcionados por el servidor de autorización a la aplicación y que le permite solicitar nuevos *access token* después de que estos hayan caducado.

Los *actores* que intervienen en todos los flujos OAuth son:

- El propietario del recurso o el usuario es la entidad que autoriza el acceso a los recursos y limita el alcance de dicha autorización.

- La aplicación que quiere acceder al recurso. Esta aplicación necesita la autorización del propietario.

- El servidor de autorización es el encargado de garantizar el acceso a los datos por parte de la aplicación, después de que el cliente haya dado su consentimiento.

- El servidor de recursos es la API o servicio que expone la información o datos consumibles a los cuales se quiere acceder. En algunos casos se trabaja el servidor de autorización con el servidor de recursos como un solo servidor.

2.2.3.2. Flujos o *grant types*

Los flujos de OAuth también denominados *grant types* se refieren al modo en que las aplicaciones obtienen un *access token* del servidor de autorización.

La existencia de estos flujos en el estándar surge para dar una solución a todos los escenarios posibles de negocio que pueden existir que se refieran al consumo de las API's, estos son totalmente dependientes a tres variables:

- El tipo de aplicación consumidora.

- Su grado de confianza.

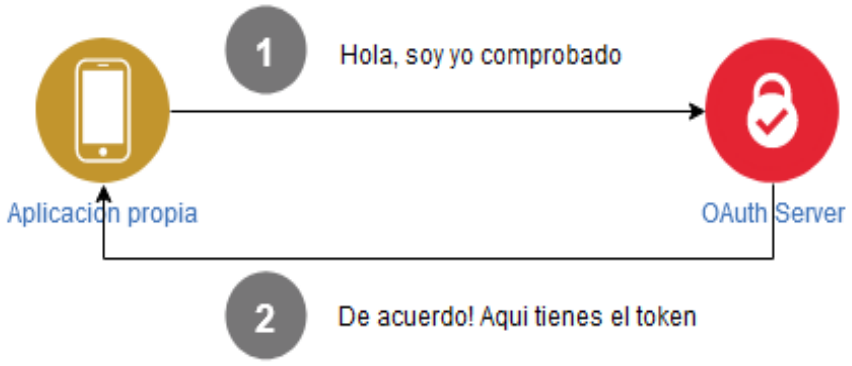
- Como es la interacción por parte del usuario en el proceso.

2.2.3.2.1. *Client credentials*

Este flujo representa el caso en el que la aplicación pertenece al mismo usuario o propietario de los recursos y no hay necesidad de que este se autentique, es decir, el *access token* que envía el servidor de autorización se obtiene autenticando solo a la aplicación, no al usuario o propietario. Se utiliza habitualmente en comunicaciones servidor-servidor.

El siguiente diagrama representa de manera simplificada el flujo de obtención de *token* con este tipo de *grant*.

Figura 1. ***Client credentials grant type***



Fuente: elaboración propia, empleando online.visual-paradigm.

En cuanto a lo que se refiere a este proyecto, este tipo de *grant* es posible utilizarlo para las aplicaciones existentes de la institución, sin embargo, este *grant* no proporciona un *refresh token*, únicamente un *access token*. Por lo tanto, queda

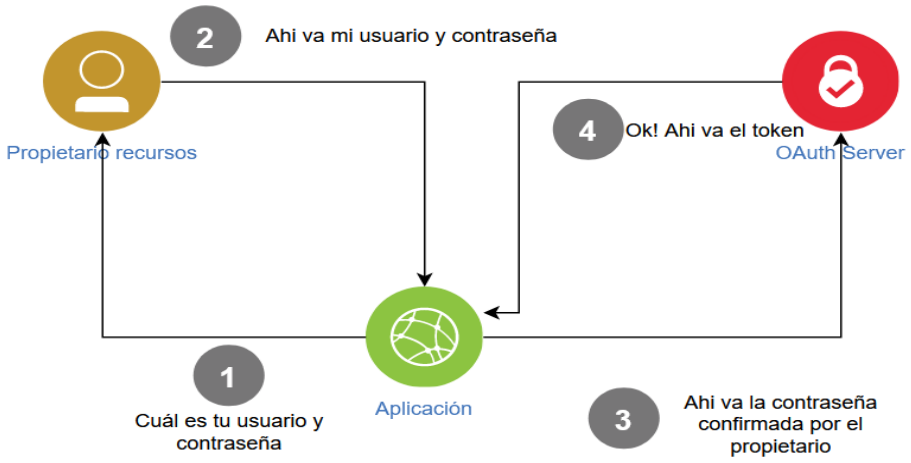
descartado ya que para efectos del proyecto necesitaremos el uso de los *refresh tokens*.

2.2.3.2.2. *Password*

Este flujo se caracteriza por la intervención de parte del usuario o propietario de los recursos, proporcionando su usuario y contraseña en la aplicación cliente, lo que implica que los dueños del recurso deben confiar plenamente en la aplicación para introducir sus credenciales. Se utiliza habitualmente en comunicaciones cliente-servidor.

El siguiente diagrama representa de manera simplificada el flujo de obtención de *token* con este tipo de *grant*.

Figura 2. ***Password grant type***



Fuente: elaboración propia, empleando online.visual-paradigm.

Para efectos del proyecto, este tipo de *grant* tiene todas las características que se necesitan para asegurar el consumo de los recursos de la institución,

pudiendo el propietario registrar todas las aplicaciones propias que desee de cualquier tipo, y también algo importante a mencionar es que este tipo de *grant* si proporciona *refresh tokens* que se utilizará para extender el tiempo de autorización a las aplicaciones.

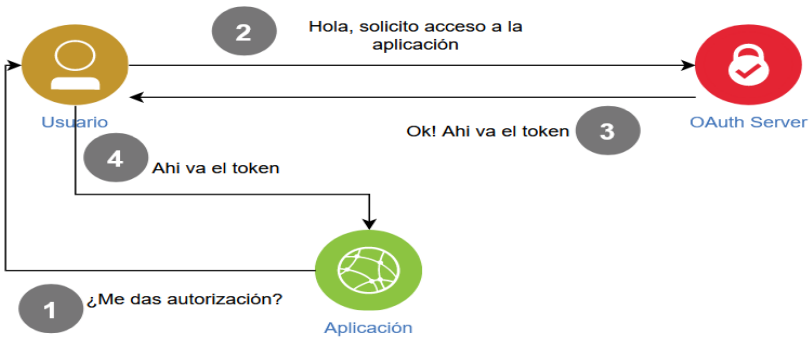
2.2.3.2.3. *Implicit*

En este flujo el usuario interviene directamente para dar su consentimiento explícito y posteriormente hace uso del *access token*. Se trata del flujo menos seguro y su utilización debe limitarse a casos muy concretos como para aplicaciones de una sola página ya que el *token* de acceso se devuelve en la URL directamente.

Otra desventaja es que no tiene un *refresh token*, ya que no existe un canal secundario, es necesario obtener un nuevo *access token*. Recibe el nombre de implícito porque toda la comunicación reside en el navegador.

El siguiente diagrama representa de manera simplificada el flujo.

Figura 3. ***Implicit grant type***



Fuente: elaboración propia, empleando online.visual-paradigm.

2.3. Presentación a la solución al proyecto

A continuación, se describen las soluciones propuestas a los problemas presentados por la Escuela de Estudios de Postgrado.

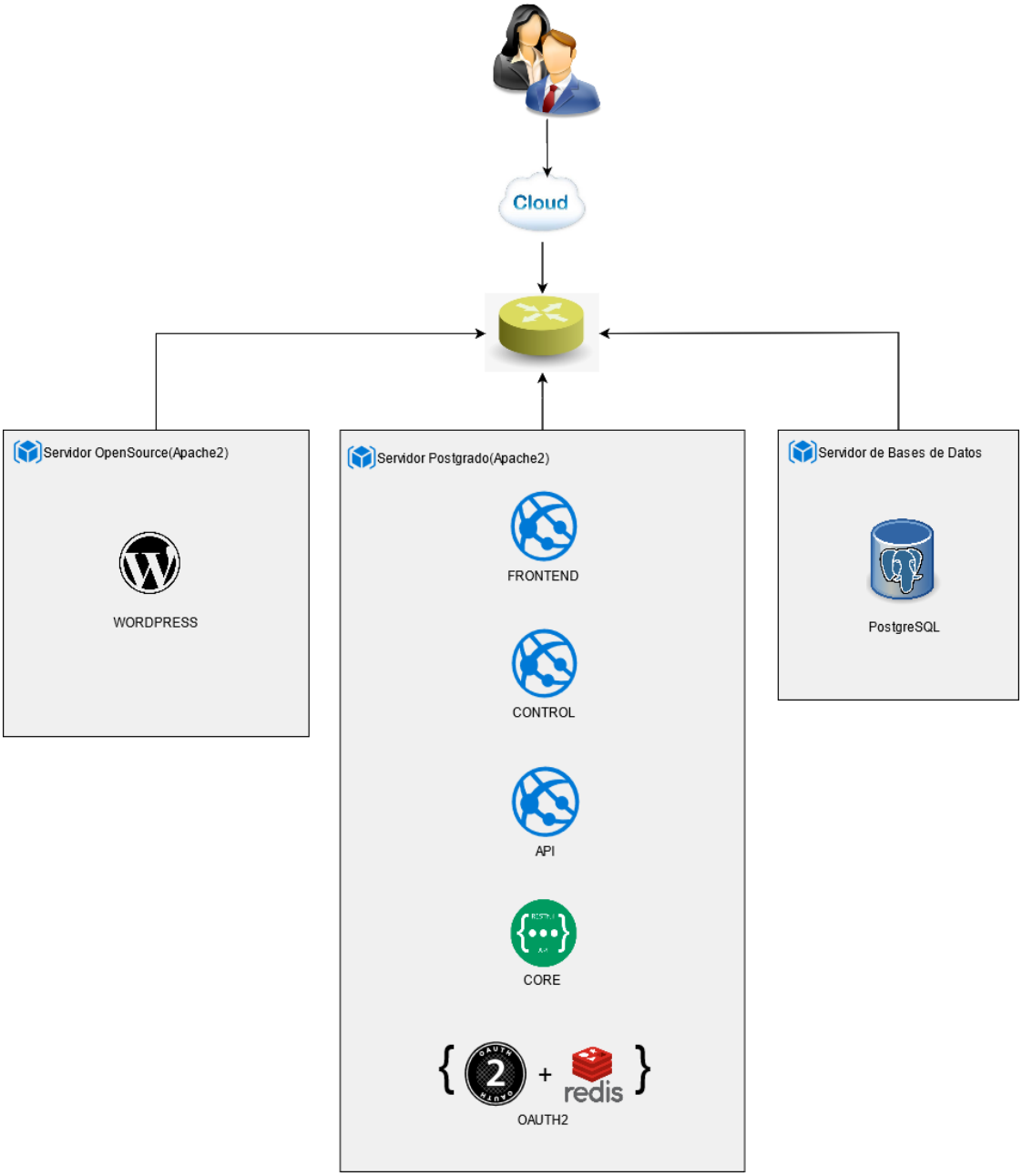
2.3.1. Arquitectura de aplicaciones

Uno de los alcances principales de este proyecto, es la propuesta nueva del diseño de arquitectura para las aplicaciones de la institución, tomando como base la infraestructura existente (Ver Anexo 1).

Para esto, se procedió a crear un nuevo proyecto llamado OAUTH2 para el servicio de autorización, en cuanto a la nueva capa de seguridad para la institución se refiere, y posteriormente se hicieron las configuraciones necesarias en el proyecto ya existente llamado *CORE*, para que este sea una aplicación de tipo API RESTful que pueda soportar el consumo de los recursos de la institución desde cualquier tipo de aplicación autorizada, desde las aplicaciones web existentes, hasta la nueva aplicación móvil que la institución tiene prevista desarrollar en próximos proyectos de EPS.

También se ha tenido que crear una instancia de base de datos no relacional en memoria dentro del servidor de las aplicaciones utilizando REDIS, esto para almacenar toda la información que el servicio de autorización necesita para funcionar, con el fin de separar las funcionalidades de este nuevo servicio de la base de datos relacional PostgreSQL y así evitar estresar el servidor de la base de datos relacional con todo lo que se refiere y necesita el nuevo servicio OAUTH2 para la autorización de aplicaciones clientes mediante los *tokens* de acceso.

Figura 4. **Arquitectura de aplicaciones propuesta**



Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.2. Servicio de autorización OAuth2

Este es un nuevo proyecto creado en el servidor de la institución llamado Postgrado (Apache2), la cual ha sido inicializada con el *Framework* de Laravel que utiliza el lenguaje de programación PHP y que contiene una implementación del estándar de código abierto *Open Authorization* la cual permite flujos de autorización seguros de modo estándar y simple, para cualquier tipo de aplicaciones informáticas, desde aplicaciones de escritorio, pasando por aplicaciones web, hasta aplicaciones móviles. Algo importante a mencionar, es que este servicio a pesar de ser un proyecto creado con Laravel no utiliza el patrón de arquitectura MVC ya que tendrá una funcionalidad de tipo API.

Dicho proyecto, ofrece los recursos necesarios para autorizar cualquier tipo de aplicaciones que la institución desee, proveyendo hacia estas aplicaciones *tokens* de acceso, para poder autorizarles el consumo de los recursos que la institución ofrece en el proyecto centralizado de tipo API RESTful llamado *CORE*.

Este nuevo servicio de autorización utiliza Redis como base de datos no relacional en memoria para almacenar toda la información necesaria para su funcionamiento, las cuales se refiere a la información las aplicaciones clientes que la institución desee registrar, así como también, toda esa información referente a los a*ccess tokens* y *refresh tokens*, de tal manera que la base de datos PostgreSQL utilizada por la institución como base de datos primaria, quede totalmente aislada de este nuevo servicio, permitiendo así evitar estresar la base de datos relacional primaria con todos los recursos y flujos de este nuevo servicio.

Entre los servicios que provee se encuentran:

- Obtención de *access token* por aplicaciones clientes consumidores de recursos.
- Validación de *access token* por aplicaciones proveedoras de recursos de la institución.
- Refrescamiento de *access token* por aplicaciones clientes consumidores de recursos.

2.3.2.1. Librerías y configuraciones iniciales

Después de inicializar un nuevo proyecto vacío con Laravel, se procede a instalar algunas librerías necesarias para llevar a cabo todas las funcionalidades requeridas, estas se pueden instalar fácilmente utilizando *composer* en la línea de comandos.

- bshaffer/oauth2-server-php "^1.10".
- predis/predis

La primera es la librería oficial para el uso del estándar OAuth2.0 con el lenguaje de programación PHP, la segunda es la librería que permite interactuar con la instancia de la base de datos en memoria Redis.

Seguidamente, se crean algunas instancias de objetos necesarias utilizando el patrón de diseño *Singleton*, las cuales nos servirán para el funcionamiento correcto del servicio de autorización, de tal manera que solo se reserve memoria una única vez para todas las instancias necesarias para todo el proyecto durante todo el tiempo de funcionamiento.

Para conseguir esto siguiendo la documentación oficial de Laravel, procedemos a registrar las instancias que necesitamos en el archivo de la ruta

app/Providers/AppServiceProviders.php en la función con el nombre *register (),* como a continuación se presenta. (Ver figura 5).

Figura 5. **Librerías y configuración inicial**

```php
<?php
namespace App\Providers;
use Predis\Client;
use OAuth2\Storage\Redis;
use OAuth2\Server;
class AppServiceProvider extends ServiceProvider
{

    /**
     * Register any application services.
     * @return void
     */
    public function register()
    {
        $this->app->singleton('PredisClient', function ($app) {
            return new Client();
        });

        $this->app->singleton('PredisStorage', function ($app) {
            return new  Redis(app('PredisClient'));
        });

        $this->app->singleton('ServerRedis', function ($app) {

            return new Server(app('PredisStorage'),array(
                'id_lifetime'                => 1800,
                'access_lifetime'            => 1800,
                'always_issue_new_refresh_token' => true,
                'refresh_token_lifetime' => 2400,
            ));
        });
    }
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

El primer bloque de código crea una instancia única de un cliente de Redis.

El segundo bloque permite crear una instancia única de referencia al almacenamiento de Redis, recibe de parámetro en su constructor el cliente de Redis creado anteriormente.

Por último, el bloque que nos permite crear y configurar una instancia única del servidor de autorización. El parámetro *id_lifetime*, permite indicar el tiempo de vida en segundos que duran los *tokens* en memoria, tanto para *access_token* como para *refresh_token*, permitiendo así ir liberando de manera automatizada el espacio en memoria utilizado cuando los *tokens* hayan expirado. Los parámetros *access_lifetime* y *refresh_token_lifetime* permiten indicar el tiempo en segundos con los cuales los *tokens* serán válidos. En este caso 30 minutos.

2.3.2.2. Controlador de recursos de autorización

Para poder ofrecer las funcionalidades necesarias se creó un controlador que sea capaz de capturar las peticiones POST para obtener, refrescar y validar los *tokens* de acceso por parte de las aplicaciones clientes. Este controlador se nombra *OAuthController.php* ubicado en la ruta *App/Http/Controllers* del proyecto de Laravel.

2.3.2.2.1. Obtención de *access token*

La ruta definida para la obtención de un *access token* es */token*, la cual ejecuta la función *token(Request $request)* dentro del controlador *OAuthController.php* antes mencionado. La petición que recibe como parámetro debe ser de tipo POST y en el cuerpo de esta petición debe venir todas las credenciales posibles para poder verificar si la aplicación cliente está autorizada para obtener un *token*. A continuación, se puede ver la estructura del código fuente dentro de la función *token().* (Ver figura 6).

Figura 6. **Función para obtención de *access token***

```
public function token(Request $request)
{
    //Obtenemos todo el contenido del request
    $input = $request->all();
    //Reporte de errores
    ini_set('display_errors',1);error_reporting(E_ALL);
    //Llamamos al singleton o instancia única del Storage o almacenamiento de Redis
    $storage=app('PredisStorage');
    //Aqui agregamos los clientes/usuarios/scropes de la institucion
    $usuarios = OauthUser::all();
    foreach ($usuarios as $usuario) {
        $storage->setUser($username=$usuario->username, $password=$usuario->password, $first_name=$usuario->first_name,$
            last_name=$usuario->last_name);
    }

    $clientes = OauthClient::all();
    foreach ($clientes as $cliente) {
        $storage->setClientDetails($client_id=$cliente->client_id, $client_secret=$cliente->client_secret, $redirect_uri=$
            cliente->redirect_uri,$grant_type=null,$scope=$cliente->scope,$user_id=$cliente->usuario->username);

    }

    //Configuramos el servidor
    $server = app('ServerRedis');
    //Devolvemos el token solicitado
    $server->handleTokenRequest(OAuth2\Request::createFromGlobals())->send();
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

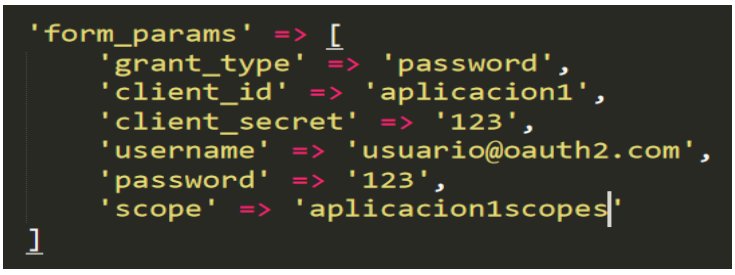
Esta es la función principal del servicio de autorización, en esta se registran los usuarios y aplicaciones clientes, las cuales son almacenadas en la base de datos no relacional Redis gracias a las funciones *setUser()* y *setClientCredentials()* de la instancia única *PredisStorage* creados con el patrón de diseño *singleton* con anterioridad, todo esto para que al momento de recibir una petición de alguna aplicación cliente, se pueda verificar mediante el método *handleTokenRequest()* del servidor si el cliente es autorizado o no. Debido a que los datos de los usuario y clientes son definidos de tal manera que no varían y se comportan de manera estática, las llaves con las que se guardan en la base de datos de Redis, siguiendo el algoritmo de almacenamiento, siempre resultara con la misma clave, por lo tanto, no importa las veces que se ejecute la función, estos se guardaran una única vez. También se pueden ir agregando todas las aplicaciones clientes que se vayan deseando conforme crezcan las aplicaciones de la institución. Para efectos de este proyecto de EPS, se definieron tres aplicaciones clientes para la institución: EEPFRONTEND, EEPCONTROL y

EEPMOVIL y un usuario administrador, los cuales se encuentran registrados dentro de la base de datos relacional PostgreSQL en las tablas oauth_*users* y oauth_*clients* (*Ver Apéndice 1*), con el único fin de que la institución pueda ir fácilmente a registrar más aplicaciones y usuarios si así lo llegara a necesitar.

Con esto se asegura que únicamente estas aplicaciones de la institución podrán pedir *tokens* de acceso y por lo tanto serán las únicas aplicaciones desde las cuales se podrán consumir los recursos que la institución provee desde el proyecto de tipo API RESTful llamado *CORE*, asegurando así la capa de seguridad a nivel de aplicaciones autorizadas de la institución.

Seguidamente, para que el servidor de autorización pueda proveer un *token* de acceso, como se mencionó anteriormente, la aplicación debe hacer la solicitud mediante una petición de tipo POST, enviando en el cuerpo de esta petición la información necesaria con la que el servidor de autorización pueda validar que la aplicación cliente desde la cual se está solicitando un *token* está autorizada. La estructura que espera recibir el servidor de autorización en el cuerpo de la petición POST tiene la siguiente forma (Ver figura 7).

Figura 7. **Cuerpo de la petición para obtención de *access token***

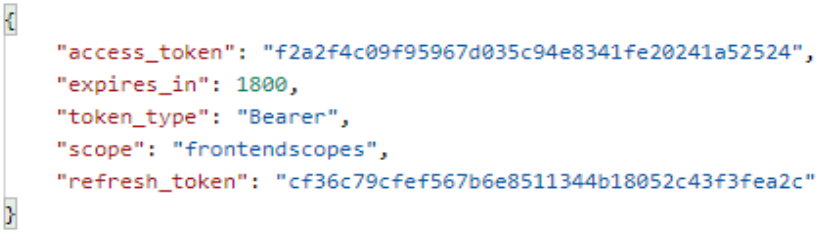


Fuente: elaboración propia, empleando online.visual-paradigm.

Por último, después de determinar que la información de la aplicación cliente que está solicitando un *token* son válidos, el servidor de autorización procede a enviar un objeto JSON como respuesta a la petición realizada, mediante su método *handleTokenRequest()->send()*, esta respuesta puede resultar con dos posibles resultados.

Si el servidor, después de hacer las verificaciones utilizando la información almacenada en la base de datos de Redis logra determinar que la aplicación es autorizada, procede a devolver el siguiente resultado satisfactorio en formato JSON. (Ver figura 8)

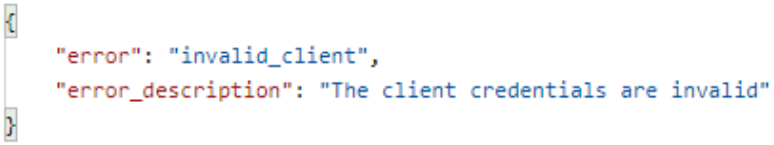
Figura 8. **Respuesta satisfactoria de petición de *access token***



Fuente: elaboración propia, empleando online.visual-paradigm.

En caso contrario de que el servidor de autorización determina que la información de la aplicación cliente con las cual se está solicitando los *tokens* de acceso son erróneos, este procede a devolver una respuesta de error en formato JSON. (Ver figura 9)

Figura 9. **Respuesta errónea de petición de *access token***



Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.2.2.2. Refrescamiento de *tokens*

La ruta definida para refrescar *access tokens* es */token*, la cual ejecuta la función *token(Request $request)* dentro del controlador *OAuthController.php* antes mencionado. Se reutiliza la ruta y función con la cual se obtiene los *access token*, la diferencia radica en el cuerpo de la petición POST que realiza la aplicación cliente, donde se le indica por parte del cliente a través del atributo *grant_type* que será de tipo *refresh_token,* a diferencia del anterior que es de tipo *password*. La estructura que espera recibir el servidor de autorización en el cuerpo de la petición POST tiene la siguiente forma (Ver figura 10).

Figura 10. **Cuerpo de la petición para refrescar *tokens***

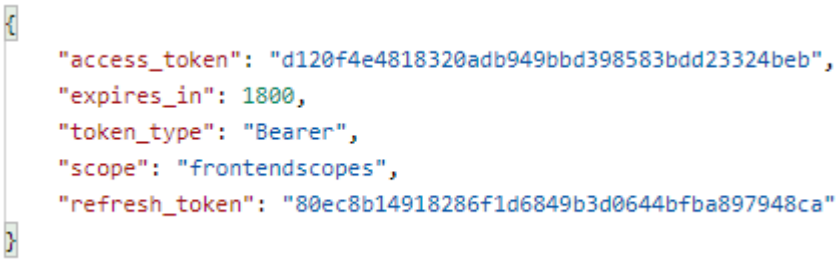
```
'form_params' => [
    'grant_type' => 'refresh_token',
    'refresh_token' => '80ec8b14918286f1d6849b3d0644bfba897948ca',
    'client_id' => 'aplicacion1',
    'client_secret' => '123',
]
```

Fuente: elaboración propia, empleando online.visual-paradigm.

El servidor al momento de determinar que se desea refrescar los *tokens*, el método *handleTokenRequest()->send()* del servidor se encarga de primeramente buscar si los *refresh token* con el que se quiere refrescar los *tokens* en realidad es válida y existe en la base de datos de Redis, y posteriormente devuelve una respuesta a la petición como un objeto de tipo JSON, la cual puede resultar en dos posibles resultados, dependiendo si el *refresh token* es correcto o no.

Si el servidor, después de hacer las verificaciones utilizando la información almacenada en la base de datos de Redis logra determinar que el *refresh token* es válido, procede a responder con un resultado satisfactorio devolviendo nuevos *tokens* de acceso a la aplicación cliente en formato JSON. (Ver figura 11)

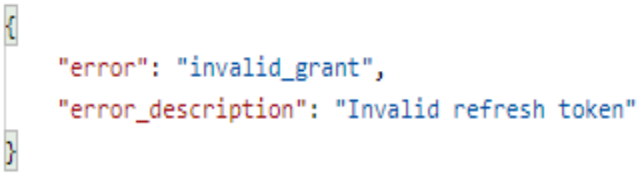
Figura 11. **Respuesta satisfactoria de refrescamiento de *tokens***



Fuente: elaboración propia, empleando online.visual-paradigm.

En caso contrario de que el servidor de autorización determina que el *refresh token* con el que se quiere extender el tiempo de autorización es erróneo, este procede a devolver una respuesta de error en formato JSON. (Ver figura 12)

Figura 12. **Respuesta errónea de refrescamiento de *tokens***



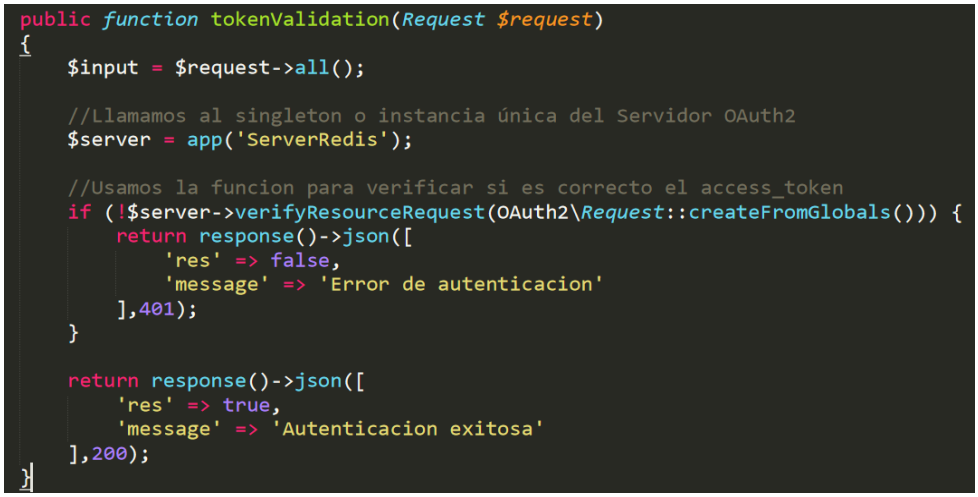
Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.2.2.3. Validación de *access token*

La ruta definida para validar *access tokens* es */validarToken*, la cual ejecuta la función *tokenValidation(Request $request)* dentro del controlador *OAuthController.php* anteriormente mencionado. La petición que recibe como parámetro debe ser de tipo POST y a diferencia de los recursos anteriores, esta ruta no necesita recibir ningún parámetro en el cuerpo de la petición, únicamente debe recibir el *bearer token* o *access token* en la cabecera *header* de la petición, con la cual el servidor de autorización procederá a validar si el *token* es correcto y no ha expirado. Este recurso a diferencia de los anteriores es consumido por la API RESTful llamado *CORE* la cual es el proyecto que contiene los recursos de la institución, y por lo tanto es este proyecto quien verifica si los *tokens* que le envían las aplicaciones cliente consumidoras son correctos, para tomar la decisión de responderle o no, los recursos que solicitan.

A continuación, se puede ver la estructura del código fuente dentro de la función *tokenValidation*(). (Ver figura 13).

Figura 13. **Función para validación de *access token***



```php
public function tokenValidation(Request $request)
{
    $input = $request->all();

    //Llamamos al singleton o instancia única del Servidor OAuth2
    $server = app('ServerRedis');

    //Usamos la funcion para verificar si es correcto el access_token
    if (!$server->verifyResourceRequest(OAuth2\Request::createFromGlobals())) {
        return response()->json([
            'res' => false,
            'message' => 'Error de autenticacion'
        ],401);
    }

    return response()->json([
        'res' => true,
        'message' => 'Autenticacion exitosa'
    ],200);
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

En la primera línea se asegura de capturar todo el contenido de la petición que se está recibiendo, luego se hace el llamado a la instancia única del servidor de autorización para posteriormente hacer uso de la función que se encarga de verificar la validez del *bearer token*. La función encargada de hacer esta validación es *verifyResourceRequest()* el cual devuelve un valor *TRUE* en caso que el *bearer token* que se está recibiendo en la cabecera *header* es válida y aún no ha expirado, en caso contrario devuelve un *FALSE*.

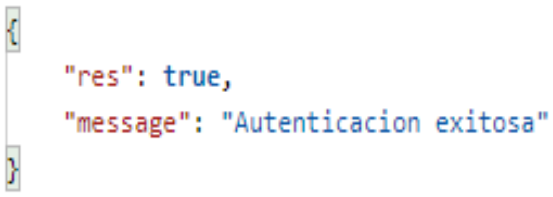
En la sentencia de flujo IF se hace la validación para devolver el resultado correspondiente utilizando la sentencia *RETURN* donde la respuesta a devolver dependerá del resultado de la validación mediante el método *verifyResourceRequest()* y con esta poder retornar el resultado compuesto con el error 200 en caso sea *token* valido o 401 una respuesta de *token* no valida.

Con esta funcionalidad, el proyecto *CORE* con los recursos de la institución y cualquier otro servicio que pudiese existir posteriormente si la institución así lo llegase a necesitar, puedan asegurarse de que los *tokens* con los cuales les consuman sus servicios sean válidos o no, y de esta manera asegurarse que únicamente aplicaciones clientes autorizadas podrán consumir los recursos de la institución que en ellas se alojan.

Por último, después de determinar si el *bearer token* que se está validando es correcto, el servidor de autorización procede a enviar un objeto JSON como respuesta a la petición realizada, donde esta respuesta puede resultar con dos posibles resultados.

Si el servidor de autorización, después de hacer la verificación utilizando la información almacenada en la base de datos de Redis logra determinar que el *access token* es válido y sigue estando activo, procederá a devolver el siguiente resultado satisfactorio en formato JSON. (Ver figura 14)

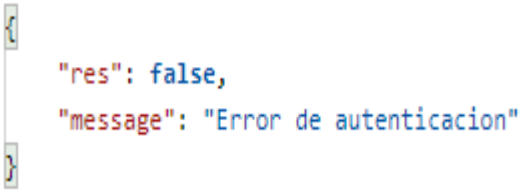
Figura 14. **Respuesta satisfactoria de validación de *access token***



Fuente: elaboración propia, empleando online.visual-paradigm.

En caso contrario, de que el servidor de autorización determina que el *access token* es incorrecto o ya ha expirado, este procederá a devolver una respuesta de error en formato JSON. (Ver figura 15)

Figura 15. **Respuesta insatisfactoria de validación de *access token***



Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.3. Servicio tipo API en el proyecto *CORE*

En este proyecto de Laravel ya existente, se harán las configuraciones necesarias para que funcione como un proyecto tipo API-RESTful, la cual estará destinado para brindar todos los recursos de la institución a diferentes tipos de aplicaciones existentes y aun no existentes de la institución, entre las aplicaciones que aún no existen se hace referencia especialmente a una aplicación móvil que la institución tiene en planes desarrollar a través de otros proyectos de EPS que están próximos a ser desarrollados. Sin embargo, esto es escalable, pudiéndose agregar más aplicaciones de cualquier tipo si así lo llegase a necesitar la institución.

Adicional a todo esto, y muy importante a mencionar, es que este proyecto hace uso de la nueva capa de seguridad del proyecto anteriormente descrito llamado OAUTH2, esto para la verificación de la autorización de las aplicaciones clientes a través de la validación de los *tokens* de acceso, con los cuales las todas las aplicaciones clientes estarán consumiendo los recursos que en este proyecto *CORE* se alojan. Para hacer esto, todos los *endpoints* de los recursos existentes en este proyecto, deberán hacer uso de un *Middleware* que se creara y se llamara *oauth2*, el cual se encarga de validar primeramente que el *bearer token* con la

cual se está solicitando el recurso desde cualquier aplicación cliente sea válido y no haya expirado, esto lo hace comunicándose con el servicio de autorización, y dependiendo del resultado de esa verificación se permitirá el consumo del recursos que se está solicitando siempre y cuando el *bearer token* sea validado y se tenga un resultado satisfactorio de parte del servidor OAUTH2.

Por lo tanto, para que una aplicación cliente pueda consumir recursos que estén alojados en este proyecto, deberán solicitar primeramente los *tokens* necesarios al proyecto OAUTH2, cualquier aplicación que no esté autorizada por la institución, no podrá obtener *tokens*, y por lo tanto no podrán consumir ningún recurso dentro de este proyecto *CORE*, poniendo así en funcionamiento la nueva capa de seguridad a nivel de autorización de aplicaciones clientes autorizados.

2.3.3.1. Validación del *access token* de una aplicación cliente

Para hacer esta validación al servidor de autorización es necesario crear algunas clases y funciones dentro de ellas que servirán para hacer todo el flujo necesario para la validación de un *access token* que un usuario o aplicación cliente provee en la cabecera *header* de un *request* al momento de solicitar algún recurso hacia el servidor *CORE* para poder consumirle de manera satisfactoria, de tal manera que sea reutilizable para todos los recursos que tendrá la institución en este proyecto.

2.3.3.1.1. *Middleware* OAuth2

Uno de los componentes más importantes, ya que con este *middleware* podremos agrupar todas las rutas o *endpoints* de todos los recursos que la institución provee en este proyecto, este será básicamente unas líneas de código

fuente que se encargaran de capturar cada petición que se haga en cualquiera de nuestros *endpoints* y posteriormente comunicarse con el servidor de autorización para la validación del *bearer token* que las aplicaciones clientes estarán enviando en la cabecera *header* de la petición o *request*. (Ver figura 16)

Figura 16. **Protección de *endpoints* a través del *middleware***

Fuente: elaboración propia, empleando online.visual-paradigm.

El *middleware* técnicamente es una clase que se crea al momento de que es ejecutada el comando "*php artisan make:middleware Oauth*", el cual es creada automáticamente en su estructura una función llamada *handle()* de tipo publica, quien se encarga de capturar todas las peticiones que se realizan en los *endpoints* que son previamente protegidos con el *middleware* que se hace mención como se puede ver en la Figura 16 donde se tienen dos rutas de tipo *get* dentro de un grupo de rutas el cual usa el *middleware* llamado "oauth2" que se detalla enseguida y que este al ser llamado ejecutara el código fuente que en ella se encuentra, en este proyecto el código fuente que se genera automáticamente dentro de la función de la clase ha sido personalizado. (Ver figura 17)

Figura 17. ***Middleware* OAuth2**

```php
namespace App\Http\Middleware;
use Closure;

class Oauth
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $response =app('OAuthClient')->tokenValidation($request);

        if(json_decode($response->getContent())->res){
            return $next($request);
        }

        return $response;
    }
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

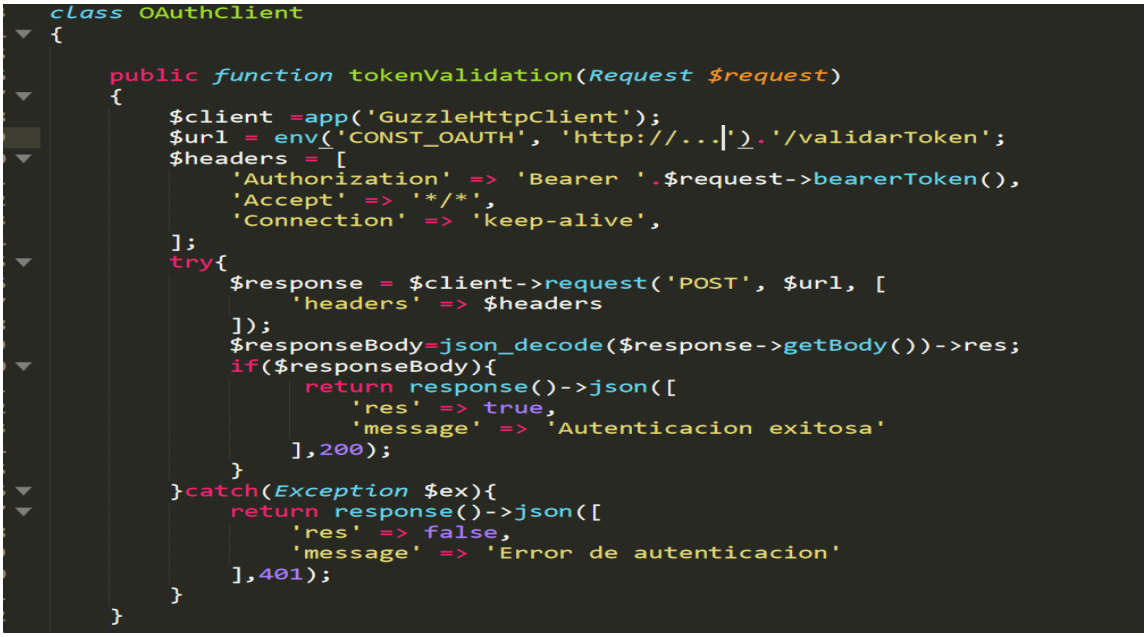
En la primera línea de la función *handle()* se hace uso de una instancia única creada con el patrón de diseño *singleton* llamada OAuthClient, el cual es una instancia de una clase también creada llamada OAuthClient.php, que hace uso de su método *tokenValidation($request),* método el cual contiene todo el código fuente que se encarga de hacer la petición y devolver el resultado booleano de la validación del *bearer token* al servidor de autorización. Seguidamente, en el caso que la validación es satisfactoria el flujo del programa entra a la sentencia IF quien se encarga de pasar el flujo del programa a la función del controlador correspondiente que devuelve los recursos solicitados a la aplicación cliente. En caso contrario se retorna la respuesta del error de autorización de la última línea de código dentro de la función.

2.3.3.1.2. Clase OAuthClient

La función *tokenValidation(Request $request)* de la instancia *singleton* de la clase *OAuthClient.php*, es la encargada de mandar el *bearer token* a través de la cabecera *header* de la petición para su análisis al servicio de autorización, y dependiendo de esa respuesta se retorna un objeto JSON personalizado que

replica el resultado devuelto por el servicio de autorización para que el *middleware* quien hace uso de esta función tome la decisión mediante la sentencia de flujo IF para ceder el flujo del programa según sea el caso. (Ver figura 18)

Figura 18. **Clase OAuthClient**



```
class OAuthClient
{

    public function tokenValidation(Request $request)
    {
        $client =app('GuzzleHttpClient');
        $url = env('CONST_OAUTH', 'http://...').'/validarToken';
        $headers = [
            'Authorization' => 'Bearer '.$request->bearerToken(),
            'Accept' => '*/*',
            'Connection' => 'keep-alive',
        ];
        try{
            $response = $client->request('POST', $url, [
                'headers' => $headers
            ]);
            $responseBody=json_decode($response->getBody())->res;
            if($responseBody){
                return response()->json([
                    'res' => true,
                    'message' => 'Autenticacion exitosa'
                ],200);
            }
        }catch(Exception $ex){
            return response()->json([
                'res' => false,
                'message' => 'Error de autenticacion'
            ],401);
        }
    }
```

Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.3.1.3. Asignación del *middleware* al *Kernel* del proyecto

Como último paso a realizar, después de tener todas las configuraciones necesarias, es registrar el nuevo *middleware* entre el array de todos los *middlewares* que utiliza el proyecto *CORE*, esto para que la aplicación reconozca de manera global la referencia que se hace al *middleware* en la agrupación realizada para proteger todos los *endpoints*.

Para esto será necesario ubicar el siguiente archivo en la siguiente ruta *app/Http/Kernel.php* y ubicar el array protegido con el nombre *$routeMiddleware[]* donde se procede a agregar y asignarle un nombre al nuevo *middleware* haciendo referencia hacia la ruta de la ubicación del archivo que representa la clase *middleware* creada anteriormente. (Ver figura 19)

Figura 19. **Asignación del *Middleware* al *Kernel***

Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.3.2. Respuesta estandarizada de la API a las aplicaciones clientes

Se ha creado una clase llamada RespuestaApiRestful ubicada en la ruta *app/Models/RespuestaApRestfuli.php* el cual se usa de manera general para todas las respuestas que devolverá la aplicación, de tal manera que exista un estándar a seguir en donde se tendrá atributos que representan metadatos para

indicar el status y una breve descripción de estos por cada respuesta a devolver, y especialmente un atributo llamado data que contendrá los datos o la información solicitada por una aplicación cliente. Esta clase tiene definida una única función llamada *setData()* que recibirá toda la información necesaria a través de sus parámetros. (Ver figura 20)

Figura 20. **Respuesta estandarizada de la API**

```php
namespace App\Models\Classes\Api;
class RespuestaApiRestful {
    var $status;
    var $message;
    var $data;
    var $errorId;
    var $errorDescription;
    /*
    Valores para $errorID:
         1   - success
         0   - error controlado
        -1   - error de sistema
    */
    function setData ($status, $message, $data, $errorId, $errorDescription)
    {
        $this->status = $status;
        $this->message = $message;
        $this->data = $data;
        $this->errorId = $errorId;
        $this->errorDescription = $errorDescription;
        if ($status == 'error') {
            if ($errorId == 0) {
                // Error controlado
                \Log::error($message);
            } else {
                // Error del sistema
                \Log::error($message . ' Error: ' . $errorDescription);
            }
        }
        return $this;
    }
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.3.2.1. Aplicación de la respuesta estandarizada

Siguiendo con los ejemplos utilizados anteriores en la utilización del *middleware* a los *endpoints*, ahora se utiliza específicamente de ejemplo el

endpoint que devuelve el historial de cursos aprobados, en donde se hace uso de la clase *RespuestaApiRestful* mediante su método *setData()* para preparar la respuesta a ser enviada a la aplicación cliente enviando todos los parámetros correspondientes donde finalmente se retorna como un objeto JSON directamente como respuesta a la petición o *request* realizada por alguna aplicación cliente. (Ver figura 21)

Figura 21. **Ejemplo del uso de la clase RespuestaApiRestful**

```php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\RespuestaApiRestful;

class ApiController extends Controller
{

    public function historialCursosAprobados(Request $request)
    {
            $respuesta = new RespuestaApiRestful();

            try{
                $consulta = DB::select(
                    "SELECT
                        .
                        .
                        .
                    from ..."
                );

                $respuesta->setData('sucess', 'Historial de cursos aprobados obtenido satisfactoriamente', $consulta, 1, null);
                return response()->json($respuesta);

            }catch(\Exception $e){
                $respuesta->setData('error', 'No se pudo obtener lista de historial de cursos aprobados', null, -1, $e->getMessage());
                return response()->json($respuesta);
            }
    }
```

Fuente: elaboración propia, empleando online.visual-paradigm.

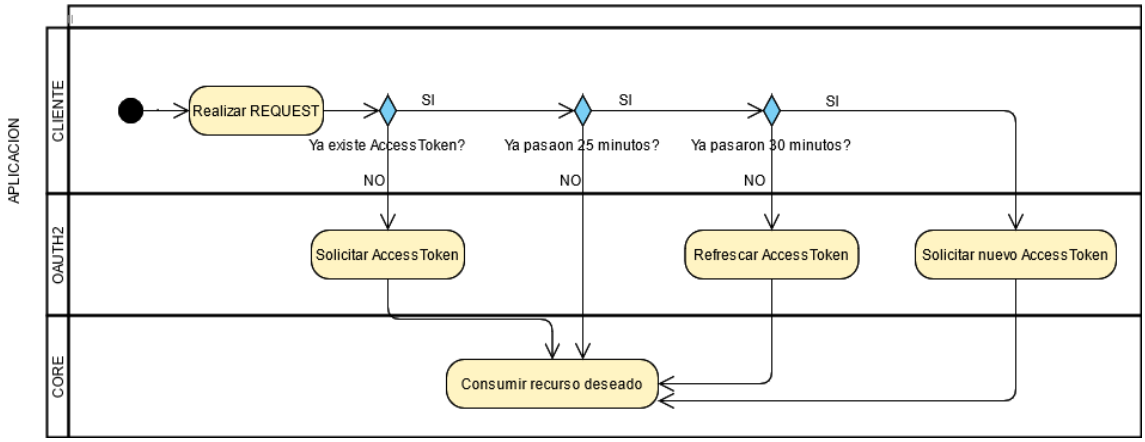
2.3.3.2.2. Consumo de recursos desde un cliente

Siguiendo con el ejemplo anterior, ahora se muestra el consumo de esa información desde una aplicación cliente. Antes que nada, es importante recordar

que para que una aplicación cliente pueda consumir satisfactoriamente algún recurso es necesario primeramente solicitar un *access token* al servicio de autorización OAUTH2 y posteriormente enviar este *access token* en la cabecera *header* de la petición que se hace al servicio *CORE* donde se solicita el recurso de interés.

Por lo tanto, debe existir un algoritmo programado en la aplicación cliente que se encargue de solicitar los *tokens* correspondientes previo al consumo de los recursos deseados, algoritmo que además de solicitar, también refresca y/o solicitar nuevos *tokens* dependiendo del caso. A continuación, se presenta el siguiente algoritmo el cual se recomienda programar para dar solución a este flujo. (Ver figura 22)

Figura 22. **Algoritmo para solicitud y refrescamiento de *tokens***



Fuente: elaboración propia, empleando online.visual-paradigm.

Finalmente, se ve en funcionamiento el consumo del recurso de historial de cursos aprobados desde la aplicación cliente web llamada *FRONTEND* donde se ha creado una clase con un método llamado *obtenerToken()* el cual ejecuta todo el algoritmo recomendado anteriormente utilizando variables de sesión.

Seguidamente, se arma la petición que se hará al proyecto *CORE* donde se le envía el *access tok*en proveído por el servicio de autorización OAUTH2 gracias a la ejecución del método *obtenerToken().* Posteriormente se obtiene los datos solicitados de interés y se guardan en una variable que finalmente es enviada a la vista y presentado de manera agradable al usuario final. (Ver figura 23)

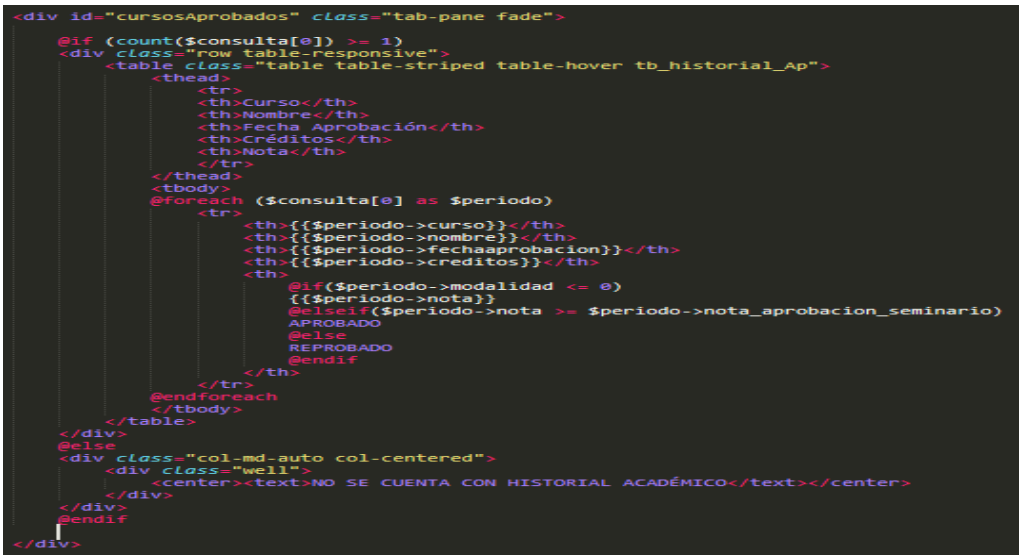
Figura 23. **Consumo del recurso historial de cursos aprobados**

```php
class EstudianteController extends Controller {

    public function HistorialAcademico() {
        app('OAuthClientController')->obtenerToken();//Se obtiene tokens
        $httpClient=app('GuzzleHttpClient');//consume recurso al CORE

        try{
            $url=env('CONST_COREURL', 'https://...').'/historialCursosAprobados?usuarioid='.Auth::user()->usuarioid.'&idcarrera='.Auth::user()->idcarrera;

            $response = $httpClient->request('GET', $url, [
                'headers' => [
                    'Accept' => 'application/json',
                    'Authorization' => 'Bearer '.session('atoken'),
                ],
            ]);

            $consulta[0] = json_decode($response->getBody())->data;//Se guadra los datos del objeto JSON devuelto

        }catch(\Exception $e){
            error_log("EXCEPCION OBTENCION DE CURSOS APROBADOS: " . $e->getMessage());
        }
        return view('directorio.estudiantes.HistorialAcademico',compact('consulta'));
    }
```

Fuente: elaboración propia, empleando online.visual-paradigm.

Después de tener la variable con los datos de los recursos solicitados, se procede a recorrer todos los elementos que contiene dentro de la vista, de tal manera que se imprimen y se presenten de manera agradable a la vista del usuario final quien interactúa desde la aplicación cliente independientemente si es de tipo web o móvil. (Ver figura 24 y figura 25)

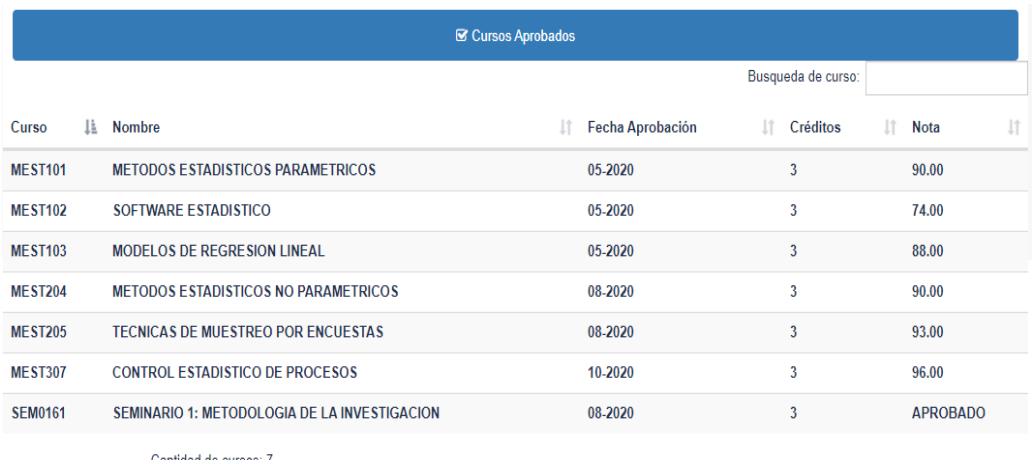
Figura 24. **Iteración de los elementos dentro de la vista**



```
<div id="cursosAprobados" class="tab-pane fade">
    @if (count($consulta[0]) >= 1)
    <div class="row table-responsive">
        <table class="table table-striped table-hover tb_historial_Ap">
            <thead>
                <tr>
                <th>Curso</th>
                <th>Nombre</th>
                <th>Fecha Aprobación</th>
                <th>Créditos</th>
                <th>Nota</th>
                </tr>
            </thead>
            <tbody>
            @foreach ($consulta[0] as $periodo)
                <tr>
                    <th>{{$periodo->curso}}</th>
                    <th>{{$periodo->nombre}}</th>
                    <th>{{$periodo->fechaaprobacion}}</th>
                    <th>{{$periodo->creditos}}</th>
                    <th>
                        @if($periodo->modalidad <= 0)
                        {{$periodo->nota}}
                        @elseif($periodo->nota >= $periodo->nota_aprobacion_seminario)
                        APROBADO
                        @else
                        REPROBADO
                        @endif
                    </th>
                </tr>
            @endforeach
            </tbody>
        </table>
    </div>
    @else
    <div class="col-md-auto col-centered">
        <div class="well">
            <center><text>NO SE CUENTA CON HISTORIAL ACADÉMICO</text></center>
        </div>
    </div>
    @endif
</div>
```

Fuente: elaboración propia, empleando online.visual-paradigm.

Figura 25. **Visualización de los elementos por el usuario final**



Cursos Aprobados

Busqueda de curso:

Curso	Nombre	Fecha Aprobación	Créditos	Nota
MEST101	METODOS ESTADISTICOS PARAMETRICOS	05-2020	3	90.00
MEST102	SOFTWARE ESTADISTICO	05-2020	3	74.00
MEST103	MODELOS DE REGRESION LINEAL	05-2020	3	88.00
MEST204	METODOS ESTADISTICOS NO PARAMETRICOS	08-2020	3	90.00
MEST205	TECNICAS DE MUESTREO POR ENCUESTAS	08-2020	3	93.00
MEST307	CONTROL ESTADISTICO DE PROCESOS	10-2020	3	96.00
SEM0161	SEMINARIO 1: METODOLOGIA DE LA INVESTIGACION	08-2020	3	APROBADO

Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.4. Módulo de seguimiento y atención de casos estudiantiles

Este es un nuevo módulo desarrollado que cuenta con dos partes que interactúan entre si desde diferentes proyectos; el primero consiste en la creación del submódulo de ingreso de casos estudiantiles, que implica extender las funcionalidades dentro del proyecto existente llamado *FRONTEND* enfocado para los usuarios de tipo aspirantes, estudiantes y docentes, el segundo consiste en extender la funcionalidades dentro del proyecto existente llamado CONTROL enfocado para el personal administrativo y que servirá para la generación de reportes de interés para la toma de decisiones de la institución.

Estas nuevas funcionalidades serán posibles a través del consumo de recursos dentro de la API anteriormente configurada dentro del proyecto llamado *CORE*, el cual, alojara todos los recursos que requiere este nuevo módulo, y que estarán protegidos haciendo uso del *middleware oauth* creado en la primera parte de este proyecto que se refiere a la capa de seguridad a través del uso de *tokens* de acceso. Para solventar todas las necesidades de la Institución, los recursos dentro del proyecto *CORE* a su vez consumirán servicios dentro de una API de un proyecto de GitLab especificado por la institución, que será utilizado para darle seguimiento y atención a los casos estudiantiles ingresados desde el portal de la Institución, como *issues* dentro del proyecto de GitLab.

2.3.4.1. Recurso de ingreso de casos estudiantiles

Para el manejo de este recurso, es necesario la creación de un controlador con todas las funciones necesarias y el modelo de datos que servirá para tener los registros de todos los casos ingresados en la base de datos, ya que, a partir de esta información almacenada se conoce quien ingreso el caso, la fecha en la

que se registró y cuál es el identificador del caso o *issue* proporcionada por la API de GitLab, ya que a partir de este identificador se le puede dar seguimiento en GitLab. Por lo tanto, antes de registrar el caso en la tabla de la base de datos de la institución (*Ver apéndice 2*), se debe asegurar primeramente el ingreso satisfactorio de este caso como un *issue* en el proyecto de G*itLab*, a través del uso de la API que ofrece.

Figura 26. **Función en *CORE* para el recurso de registro del caso estudiantil**

```php
//POST
public function store(CasosEstudiantilesStoreRequest $request)
{
    $respuesta = new RespuestaApiRestful();
    try {
        $issue=$this->registrar_issue($request);//Primero registramos el caso como un issue en GitLab

        $input = $request->all();
        $input['issue'] = $issue->iid;//Agregamos el id del issue en nuestros campos para almacenar en la BD

        $caso = CasosEstudiantiles::create($input);
        $respuesta->setData('success', 'Caso estudiantil se ha registrado correctamente.', $caso, 1, null);

    } catch (Exception $ex) {
        $respuesta->setData('error', 'Ocurrio un error para almacenar el caso estudiantil.', null, 0, $ex->getMessage());
    }

    return response()->json($respuesta);
}
```

Fuente: elaboración propia, empleando online.visual-paradigm.

2.3.4.2. Consumo del recurso de ingreso de casos estudiantiles

El consumo de este recurso se hace desde un nuevo módulo dentro proyecto llamado *FRONTEND*, en el cual se ha creado un formulario que puede ser vista y utilizada por los tres diferentes tipos de usuario que interactúan en este proyecto, los cuales son: aspirantes, estudiantes y docentes. Cada uno de

estos usuarios funcionan como roles, y a estos roles se les asigna el permiso de acceso al nuevo módulo de ingreso de casos estudiantiles, de tal manera que en cualquiera de los tres tipos de sesiones se tenga acceso al menú y así poder ingresar un nuevo caso estudiantil. Para acceder a este formulario principal, se ha agregado un nuevo menú con el nombre “Solicitudes Administrativas” y únicamente se debe dar clic sobre el submenú “Mis solicitudes”.

Figura 27. **Formulario en *FRONTEND* para el registro de un caso estudiantil**

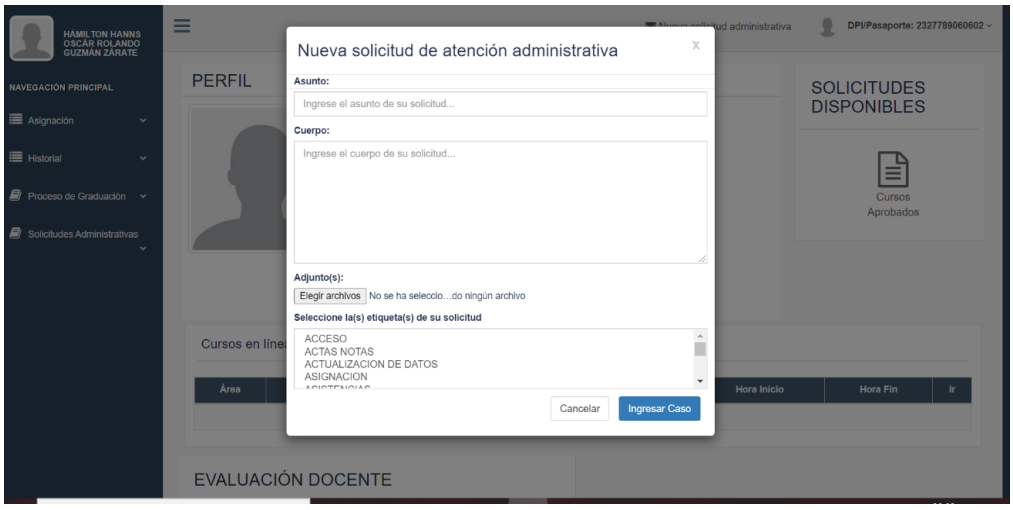
Fuente: elaboración propia, empleando online.visual-paradigm.

Tanto el campo *Asunto* como el campo *Cuerpo* son obligatorios, a diferencia de los otros dos campos, ya que, en el campo *Adjunto*, si se desea se puede adjuntar uno, varios o ningún archivo, de cualquier tipo. Así también. en el campo de las *Etiquetas*, se puede seleccionar todas las etiquetas con las cuales se desea etiquetar un caso a ingresar, con la diferencia que, en este caso, como mínimo se debe seleccionar una etiqueta.

Por tanto, al presionar el botón Ingresar Caso, esta aplicación procederá con el flujo ya estandarizado y que se debe seguir para el resto de consumo de recursos que se mencionaran en el documento, en donde primero se procederá a solicitar los *tokens* de acceso al servidor de autorización, con el fin de poder consumir el recurso alojado en el proyecto *CORE* para registrar el nuevo caso estudiantil, ya que de no tener estos *tokens* de acceso, el servidor devolverá un error de autorización como ya se ha mencionado anteriormente.

También, se ha desarrollado una venta emergente para el ingreso de estos casos, de tal manera que el usuario final pueda ingresar un caso estudiantil desde cualquier vista o ventana de la aplicación, dicha ventana modal se puede acceder a través de clic sobre un nuevo botón que colocado en la esquina superior derecha de cualquier vista en la aplicación de *FRONTEND*, como se puede alcanzar a ver en la figura 27, con la descripción de “Nueva solicitud administrativa”.

Figura 28. **Ventana modal para el registro de un caso estudiantil**



Fuente: elaboración propia, empleando online.visual-paradigm.

Todos los campos de esta ventana modal siguen la misma lógica de requerimientos del formulario principal, como se ha descrito anteriormente y que se refiere a la figura 27.

2.3.4.3. Resultado del ingreso de un caso estudiantil

El primer paso del flujo de un ingreso de caso estudiantil es la creación del caso como un *issue* dentro de un proyecto de GitLab. Esto se hace a través de la API que ofrece, la cual cuenta con varios recursos que permite interactuar con los proyectos alojados a través de este servicio. En esta ocasión, se utiliza el recurso que ofrece para la creación de un *issue*, junto con el recurso para subir un archivo en el proyecto. Ambos recursos combinados para satisfacer las necesidades que cumplen con el ingreso de un caso estudiantil desde el portal de la institución.

Tabla I. **Recurso de GitLab para creación de nuevo *issue***

POST /projects/:id/issues			
Atributo	**Tipo**	**Requerido**	**Descripción**
title	string	si	Título o asunto del issue
description	string	si	Descripción o cuerpo del issue
labels	string	no	Cadena separada por comas de las etiquetas que representan el issue

Fuente: elaboración propia, empleando Microsoft Word.

En esta y las próximas tablas se menciona las rutas y los tipos de peticiones que se deben hacer para poder consumir los recursos de GitLab que se utilizan en este proyecto. Uno de los valores que se puede ver en la cadena de la ruta es la sub-cadena “:id”, esta se refiere al código único de identificación del proyecto en GitLab a donde se desea ingresar los nuevos *issues*. Normalmente es una

secuencia de números enteros que genera GitLab para identificar un nuevo proyecto y lo hace llamar Project ID. Este recurso retorna un objeto JSON con toda la información del *issue* que se creó, con la finalidad de recibir el identificador del *issue* para posteriormente registrarlo en la base de datos de la institución.

Tabla II. **Recurso de GitLab para cargar archivos**

POST /projects/:id/uploads			
Atributo	**Tipo**	**Requerido**	**Descripción**
file	string	si	El archivo que se desea cargar
id	string o entero	si	El id del proyecto de GitLab

Fuente: elaboración propia, empleando Microsoft Word.

Este recurso se utiliza para subir archivos adjuntos. La cual retorna un objeto JSON con las rutas relativas con las cuales se puede acceder a los archivos dentro del proyecto de GitLab a donde se sube; proyecto que se hace referencia indicándolo en el atributo "*id*" que se envía en el cuerpo de la petición que se realiza al momento de cargar un archivo.

Seguidamente, se concatena esta ruta relativa del archivo cargado en el atributo "*description*" que se envía en el cuerpo de la petición para crear un nuevo *issue*, como se describe en la Tabla I. Esta cadena por concatenar debe seguir la siguiente sintaxis especifica:

Entre corchetes se debe colocar el nombre del archivo cargado, el cual es devuelto en el objeto JSON en el atributo "alt" y seguidamente sin ningún espacio entre medio, colocar entre paréntesis la ruta relativa hacia el archivo dentro del proyecto, y el cual también es devuelto en el objeto JSON en el atributo "url".

Ejemplo: *[fotoPrueba](/uploads/2f484137e4ee312/fotoPrueba.jpg).*

Al momento de lograr registrar un caso estudiantil como un *issue* en el proyecto de GitLab, rápidamente se registra el caso estudiantil en la base de datos de la institución, para tener registro del usuario que lo ingreso desde el portal, el tipo de usuario y el número de *issue* que generó. Esto con el fin de poder filtrar la información dependiendo del usuario final y mostrarle únicamente sus casos en su perfil dentro del portal del proyecto de *FRONTEND*.

El resultado final de este registro de caso estudiantil es el nuevo *issue* dentro de la lista o bandeja de entrada del proyecto en GitLab (Ver figura 29), proyecto que únicamente el personal administrativo tendrá acceso para darle seguimiento y atención a estos casos, siguiendo los procesos de negocio que son establecidos por los encargados de la jerarquía más alta del personal administrativo.

A continuación, en la siguiente imagen se observan seis *issues* creados desde el portal, donde se puede ver el título o asunto del *issue*, las etiquetas con la cual fue ingresada que puede ser uno más etiquetas, la fecha en la que se creó, el o las personas asignadas al caso, y algo muy importante; el *bot* con el cual se está registrando los *issues* el cual se ha configurado con el nombre “IssuesPostgradoBot” que es el que utilizara el portal de la institución para registrar los casos estudiantiles como *issues* dentro del proyecto de GitLab.

Figura 29. **Vista de la bandeja de entrada de *issues* en el proyecto de GITLAB para el personal administrativo**

Fuente: elaboración propia, empleando online.visual-paradigm.

En cuanto a la vista que tiene el usuario final para ver el historial de sus casos estudiantiles ingresados, se ha desarrollado una vista en donde cada usuario según su tipo puede ver tanto sus casos estudiantiles cerrados o dados por finalizados, así como también sus casos que actualmente siguen abiertos o activos.

El acceso a esta vista es a través del menú con el nombre “Solicitudes Administrativas”, en su submenú “Mis solicitudes”, en la parte baja de la vista se podrá filtrar entre los casos abiertos y cerrados (Ver figura 30). También se podrá ver información relevante, como el número de ticket que le es asignado, en este caso hace referencia al número de *issue* con el cual se registró en GitLab, así como el asunto con el que el usuario ingreso su caso, la fecha de creación, la última fecha de modificación para que el usuario sepa cómo va siendo atendido su caso, el estado, y el personal administrativo el cual tiene asignado el caso. Si el caso recién fue ingresado, la columna personal asignado aparecerá como “Pendiente”. En la última columna llamada Operación, se puede abrir en una

nueva ventana el caso seleccionado con mayores detalles y otras funcionalidades adicionales (Ver figura 31).

Figura 30. **Vista de los casos ingresados por un usuario final dentro del portal de la institución**

Fuente: elaboración propia, empleando online.visual-paradigm

2.3.4.4. Seguimiento de un caso de un usuario final desde el portal de la institución

Una de las funcionalidades más importantes es el poder darle seguimiento a un caso en específico seleccionado por un usuario final, esta vista fue desarrollado específicamente para que un usuario pueda ver todo el detalle que se tiene respecto un caso seleccionado, dicha vista es alimentada primeramente por la información que se tiene en la base de datos de la institución, y complementada con información que se consume a través de la API de GitLab, gracias a que sabemos cuál es el número de *issue* el cual el usuario desea ver y posteriormente mediante este número solicitarle a la API de GitLab toda la información referente a esta.

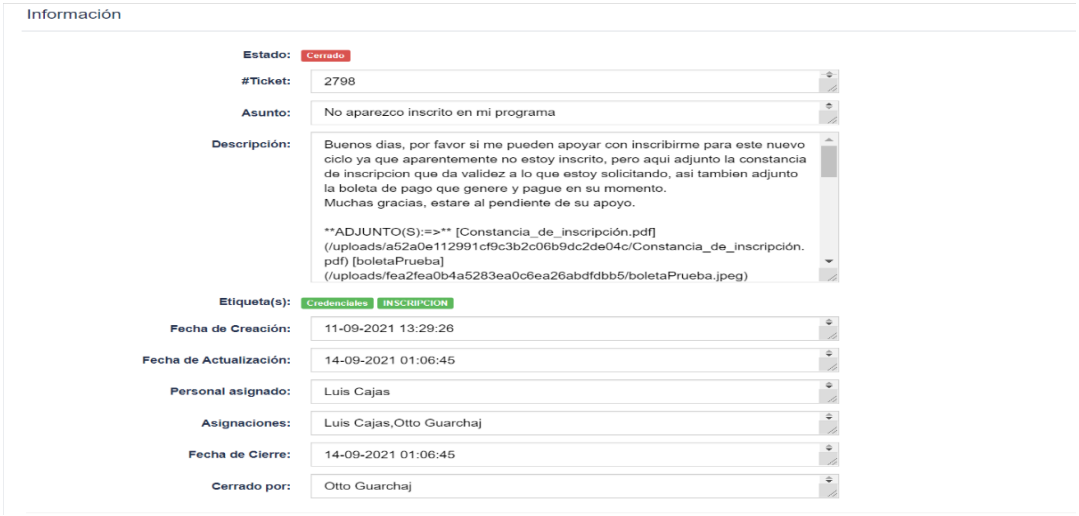
En esta vista, se puede ver información detallada sobre un caso abierto o cerrado, la cual está dividida en dos partes que a continuación se detalla.

La primera muestra información importante que le permite al usuario final saber el estado actual de la atención que se lleva respecto a su caso, donde se puede ver si el caso está abierto o cerrado, la fecha en la que se registró el caso, las etiquetas con las cuales fue creada, el asunto y el cuerpo así como el número de *ticket* o *issue* con el cual se identifica de manera única un asunto estudiantil y por supuesto también se puede ver el personal administrativo el cual tiene asignado o quien está atendiendo el caso y la última fecha de modificación que ha tenido. (Ver figura 31)

La segunda permite ver todos los registros de notas y actividades sobre la atención que se le ha estado dando al caso de parte del personal administrativo desde el lado de GitLab, en estas notas y registros de actividades el usuario final también puede participar, permitiendo así, en caso fuera necesario, tener una comunicación o interacción de manera directa entre el personal asignado al caso y el usuario final a través de dos aplicaciones diferentes, desde el portal de la institución del lado del usuario final(estudiante, aspirante o docente) y desde el proyecto de GitLab para el personal administrativo de diferentes niveles de jerarquías (Difusión, Información, Control académico, Tesorería e informática). (Ver figura 32)

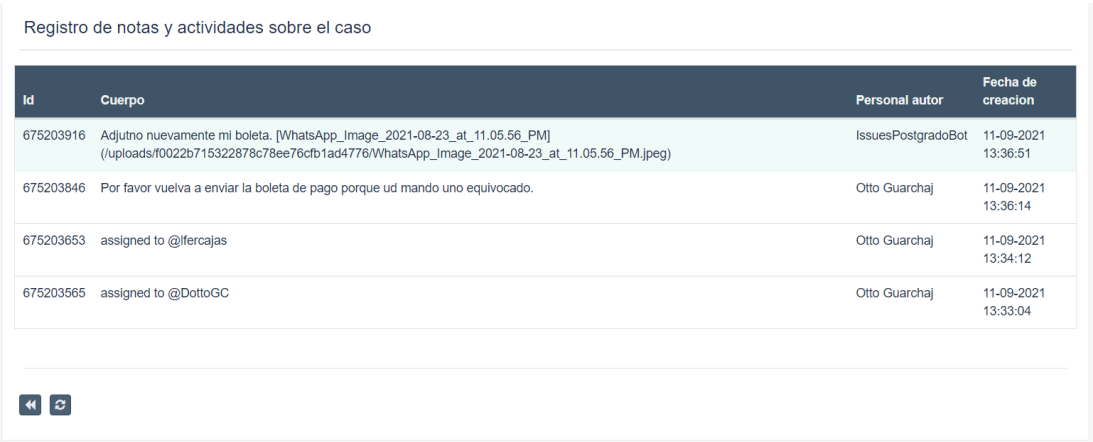
Para fines de ilustración en las siguientes dos imágenes se puede ver los detalles descritos anteriormente sobre un caso estudiantil de ejemplo el cual ya ha sido cerrado o atendido.

Figura 31. **Vista detallada de la información de un caso para su seguimiento desde el portal de la institución**



Fuente: elaboración propia, empleando online.visual-paradigm.

Figura 32. **Vista detallada de las notas y actividades de un caso, para su seguimiento desde el portal de la institución**



Fuente: elaboración propia¸ empleando online.visual-paradigm.

La única diferencia entre la vista detallada entre un caso abierto y cerrado, es que en el caso cerrado (Ver figura 32), no existe ninguna opción habilitada para responder sobre las notas o actividades del caso, a diferencia de los casos abiertos donde sí se tiene habilitado la funcionalidad de responder a las notas y actividades (Ver figura 33), de tal manera que el usuario tenga interacción con el personal administrativo asignado si así fuera requerido, por ejemplo, cuando para atender el caso es necesario algún dato o documento faltante, el personal hace una nota o comentario sobre el caso desde el proyecto de GitLab solicitándole al usuario mediante las notas y actividades que se pueden registrar sobre un caso, y el usuario podrá responderlo adjuntando o no algún documento desde el portal de la Institución.

Figura 33. **Vista de las notas y actividades de un caso abierto, para su seguimiento desde el portal de la institución**

Fuente: elaboración propia, empleando online.visual-paradigm.

De esta manera, se da por resulto las necesidades de la institución de centralizar el ingreso de los casos estudiantiles o asuntos administrativos desde

el portal de la institución, permitiendo así agilizar los procesos, de tal manera que el usuario final sepa el estado y el historial de sus casos ingresados.

2.3.4.5. Atención de un caso del personal administrativo desde el proyecto de GitLab

En el lado del personal administrativo, se tiene la vista detallada de un *issue* según el que se seleccione entre la lista de los casos abiertos que se pueden ver en la vista que se mencionó anteriormente (Ver figura 29).

Al dar doble clic sobre un caso en específico, se abrirá otra vista para poder darle toda la atención necesaria, y se podrá ver además del asunto, etiquetas y numero de *issue*, el cuerpo del caso ingresado desde el portal de la institución.

También se mostrará el acceso a los archivos adjuntos si hubiera, y por supuesto, una de las ventajas que se tiene de enviar los asuntos estudiantiles o *issues* desde el portal de la institución, es el acceso a toda la información referente al usuario final que se encuentra almacenada en la base de datos de la institución, por lo tanto, se muestra también el tipo de usuario, dpi, registro académico, nombre completo, carrera, entre otros (Ver figura 34).

Estos datos varían según el tipo de usuario, pero la idea es que el personal administrativo tenga la información básica del usuario para poder darle una atención más rápida y precisa, y solo en casos especiales solicitarle al usuario final más información a través de las notas y actividades sobre el caso.

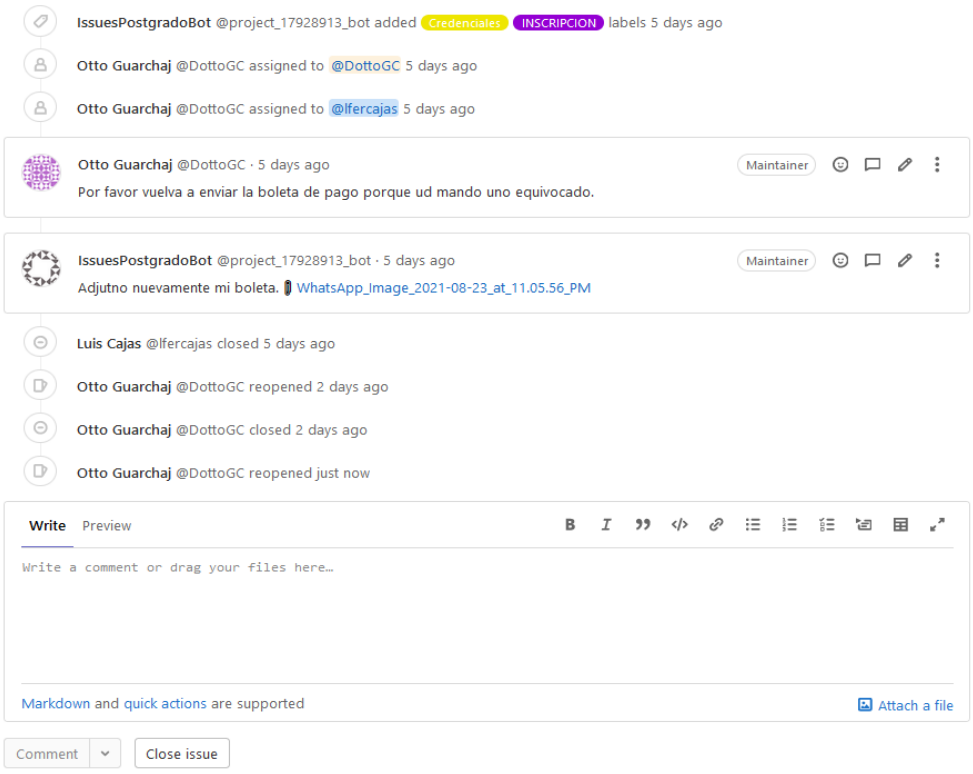
Figura 34. **Vista detallada de la información de un *issue* para su atención dentro del proyecto de GitLab**

Fuente: elaboración propia, empleando herramienta de recortes de Windows.

En el lado derecho de la vista (Ver figura 34), se tiene la opción de asignar el personal que atenderá el caso, en la imagen de ejemplo, se puede ver que tiene 2 personas asignadas, esta funcionalidad le servirá al personal administrativo para asignar el personal según lo que requiera el caso, ya que existen diferentes niveles de jerarquía y dependiente del caso, se puede atender desde la jerarquía más baja que es Difusión, hasta llegar a la jerarquía más alta que es Informática, si el caso lo llegara a necesitar.

Estas asignaciones que se realizan se van registrando en el historial del *issue* y esta información es la que el usuario final puede ver desde el portal de la institución y de esta manera pueda saber cómo va el proceso de la atención de su caso, desde el momento en que se registró, quien o quienes lo han atendido, hasta el momento en que se da por atendida o finalizada el caso.

Figura 35. **Vista de las notas y registro de actividades sobre un *issue* dentro del proyecto de GitLab**



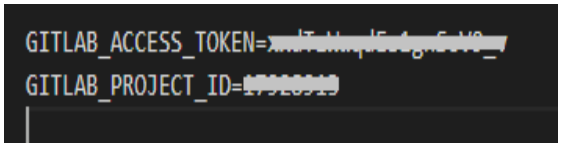
Fuente: elaboración propia, empleando herramienta de recortes de Windows.

Todas las actividades que se tengan sobre un *issue* se van registrando (Ver figura 35), desde el momento en que se ingresó el caso y se creó el *issue*, pasando por la primera persona asignada y posteriormente como la primera persona asignada se lo asigno a otra persona, esto simulando que la primera persona no haya podido atender el caso debido a que el caso requería una atención en un nivel de jerarquía más alta.

Posteriormente, se puede ver como la persona asignada le solicita al usuario final que vuelva a enviar un documento, seguidamente el usuario envía

el documento solicitado a través del portal de la institución, y como se podrá ver, todas las respuestas que se hagan desde el portal de la institución se registran en GitLab con el identificador del *bot* configurado con el nombre "IssuesPostgradoBot". Esta configuración es parametrizable en el proyecto *CORE* de la institución, a través del archivo *.env (*Ver figura 36*)*, donde el primero hace referencia al *token* de acceso generada en GitLab por un usuario administrador del proyecto para el *bot*, permitiendo así registrar cualquier actividad con el *token* sea registrado por un usuario llamado "IssuesPostgradoBot" que la institución ha decidido asignar para este proyecto. El otro parámetro es el identificador del proyecto de GitLab, que nos permite asignar el proyecto que será destinado a recibir todos los *issues* o casos estudiantiles para poder atenderlos.

Figura 36. **Parámetros de configuración de GitLab en el proyecto *CORE***



Fuente: elaboración propia, empleando Sublime 3.

2.3.4.6. Generación de reportes por rango de fechas

Se ha creado un nuevo módulo para la generación de los reportes de casos cerrados y abiertos por rango de fechas dentro del proyecto llamado CONTROL, el cual es un proyecto que forma parte del portal de la institución dedicada específicamente para el personal administrativo.

Estos reportes son generados dentro del proyecto *CORE* de tipo API RESTful, y es consumida por el proyecto CONTROL dedicado al personal administrativo, donde serán visualizadas según sea necesario por el usuario que tenga acceso al módulo.

Esta interacción entre los dos proyectos siguiendo el mismo proceso explicado con anterioridad, a través del uso de *tokens* de acceso de parte del proyecto CONTROL para poder consumir y obtener la información a través de objetos JSON de parte del proyecto *CORE*, ya que, de no ser así, no habrá comunicación por falta de autorización.

Para tener acceso a este módulo, se debe buscar del lado izquierdo el menú con el nombre “Reportes Generales” y dar clic sobre el submenú “Solicitudes Administrativas GitLab”. A continuación, se verá la vista en donde se podrá filtrar primero por los tipos de reportes disponibles, y posteriormente, por el rango de fecha que se desea analizar. (Ver figura 37)

Figura 37. **Módulo de generación de reportes de casos estudiantiles en el proyecto CONTROL**

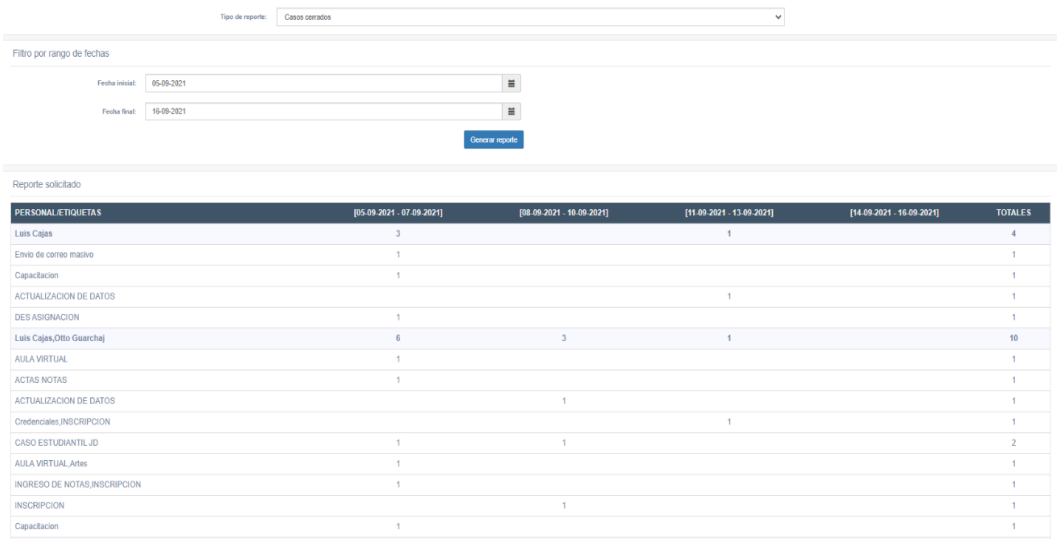
Fuente: elaboración propia, empleando herramienta de recortes de Windows.

2.3.4.6.1. Reporte de casos cerrados

En este reporte se muestra los casos cerrados por personal administrativo, durante un rango de fechas según el filtro seleccionado y el cual es dividido dentro de otros cuatro sub-rangos de fechas, de tal manera que se puedan ver los subtotales por columna, así como también entre esos casos cerrados, cuáles son las etiquetas de esos casos que fueron cerrados por el personal administrativo. Así, el personal interesado por el reporte puede saber cuáles son los casos que más se han estado atendiendo durante un periodo de tiempo, y quien es el personal administrativo que ha estado atendiendo más casos.

La fecha tomada en cuenta para este reporte es la fecha en que el *issue* fue cerrado en GitLab, y para esto primero se tuvo que filtrar para obtener únicamente los casos con el estado '*closed*'.

Figura 38. **Reporte de casos cerrados por rango de fechas**



PERSONAL/ETIQUETAS	[05-09-2021 - 07-09-2021]	[08-09-2021 - 10-09-2021]	[11-09-2021 - 13-09-2021]	[14-09-2021 - 16-09-2021]	TOTALES
Luis Cajas	3		1		4
Envio de correo masivo	1				1
Capacitacion	1				1
ACTUALIZACION DE DATOS			1		1
DES ASIGNACION	1				1
Luis Cajas,Otto Guarchaj	6	3	1		10
AULA VIRTUAL	1				1
ACTAS NOTAS	1				1
ACTUALIZACION DE DATOS		1			1
Credenciales,INSCRIPCION			1		1
CASO ESTUDIANTIL JD	1	1			2
AULA VIRTUAL,Artes	1				1
INGRESO DE NOTAS,INSCRIPCION	1				1
INSCRIPCION		1			1
Capacitacion	1				1

Fuente: elaboración propia, empleando herramienta de recortes de Windows.

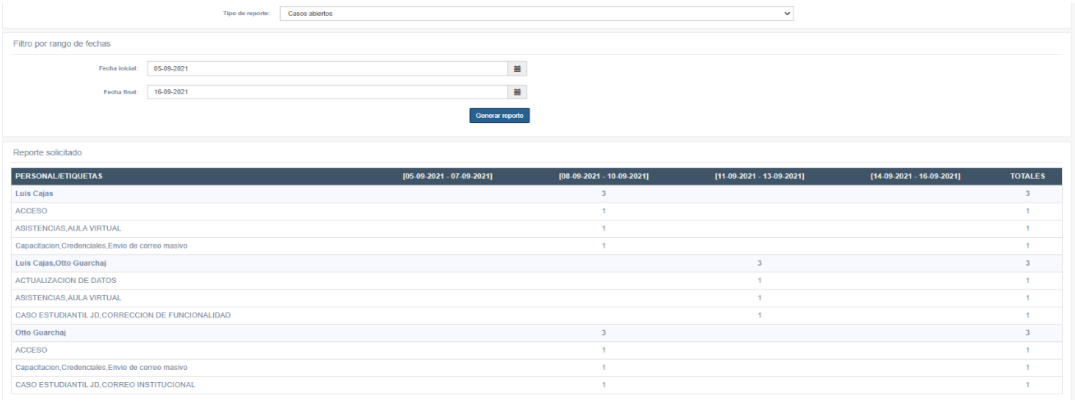
2.3.4.6.2. Reporte de casos abiertos

Este reporte muestra los casos que actualmente siguen abiertos o que no han sido cerrados por algún personal administrativo, esto también según el rango de fechas del filtro seleccionado que a si mismo también es dividido en cuatro sub rangos, de tal manera que se pueda ver los subtotales por columnas, de la misma manera en que se muestra en el primer reporte, con la diferencia que en este caso se refiere a los casos que aún se están atendiendo, ya sea que tenga una persona asignada o aun no, pero que aún no se han cerrado.

De esta manera, el personal interesado por este reporte podrá saber cuáles son los casos que se siguen atendiendo durante un periodo de tiempo, y quien o quienes son las personas que los tienen asignados.

La fecha tomada en cuenta para este reporte es la fecha en que el *issue* fue creado desde el portal de la institución por un usuario final, y también se tuvo que filtrar para obtener únicamente los casos con el estado '*opened'*.

Figura 39. **Reporte de casos abiertos por rango de fechas**



PERSONAL/ETIQUETAS	[05-09-2021 - 07-09-2021]	[08-09-2021 - 10-09-2021]	[11-09-2021 - 13-09-2021]	[14-09-2021 - 16-09-2021]	TOTALES
Luis Cajas		3			3
ACCESO		1			1
ASISTENCIAS,AULA VIRTUAL		1			1
Capacitacion,Credenciales,Envio de correo masivo		1			1
Luis Cajas,Otto Guarchaj			3		3
ACTUALIZACION DE DATOS			1		1
ASISTENCIAS,AULA VIRTUAL			1		1
CASO ESTUDIANTIL JD,CORRECCION DE FUNCIONALIDAD			1		1
Otto Guarchaj		3			3
ACCESO		1			1
Capacitacion,Credenciales,Envio de correo masivo		1			1
CASO ESTUDIANTIL JD,CORREO INSTITUCIONAL		1			1

Fuente: elaboración propia, empleando herramienta de recortes de Windows.

2.4. Costos del proyecto

A continuación, se elaboró un presupuesto de lo que podría ser el costo del desarrollo de este proyecto de EPS para la institución.

Tabla III. **Presupuesto del proyecto**

Recursos	Cantidad	Costo Unitario	Subtotal
Sueldo de 8 horas al mes, por asesoría de Escuela de Ciencias y Sistemas.	6 meses	Q 2 000,00	Q 12 000,00
Sueldo por desarrollo de *software* para analista y programador (Epesista), por 20 horas a la semana.	6 meses	Q 9 000,00	Q 54 000,00
Servicio de Internet.	6 meses	Q 400,00	Q 2 400,00
Servicio de electricidad.	6 meses	Q 350,00	Q 2 100,00
Servicio de mantenimiento y limpieza de equipo de cómputo.	3 veces	Q 250,00	Q 750,00
Costo por creación/edición de video tutoriales y capacitación de personal.	1 vez	Q 4 000,00	Q 4 000,00
		TOTAL	**Q 75 250,00**

Fuente: elaboración propia, empleando Microsoft Excel

2.5. Beneficios del proyecto

Los beneficios se presentan como soluciones a las necesidades obtenidas de los requerimientos que fueron recibidos por parte de los representantes de la institución. previo al desarrollo de este proyecto y las nuevas necesidades que

fueron surgiendo en el transcurso del tiempo del desarrollo que se fueron adaptando y solventando para ayudar a la institución. Los beneficios principales son:

- Un sistema centralizado del seguimiento y atención de casos estudiantiles, donde tanto el personal administrativo como el usuario final puedan monitorear el estado de un caso, tener interacción entre ellos hasta dar por finalizado un caso.
- Un nuevo diseño de arquitectura de servicios para que la institución pueda empezar a desarrollar aplicaciones móviles capaz de consumir recursos de un proyecto centralizado de manera segura, a través de la integración de un nuevo servicio de autorización para el uso de *tokens* de acceso para aplicaciones autorizadas por la institución. Especialmente el consumo de todos los recursos que tiene que ver con el nuevo módulo de atención de casos estudiantiles.
- Disponibilidad de la documentación con los estándares a seguir para migrar todos los recursos de la institución al proyecto que servirá como el núcleo todos los recursos de la institución protegidos con la capa de seguridad de autorización.
- Mejoras en el tiempo de atención de los casos estudiantiles, así como una mejor experiencia por la creación de vistas donde fácilmente pueden interactuar los usuarios, ver información respecto el estado del caso, y enviar documentos de ser requerido.

3. FASE ENSEÑANZA APRENDIZAJE

3.1. Capacitación propuesta

La capacitación se realiza de manera virtual a través de video tutoriales que estarán disponibles de manera pública a través de la red social YouTube con la cuenta de la Institución. Este video será únicamente para usuario finales de tipo estudiantes, aspirantes y docentes.

3.2. Material elaborado

A continuación, se describen los materiales elaborados para llevar a cabo el proyecto.

3.2.1. Manual técnico para la implementación del servicio de autorización

Este es un documento en formato PDF entregado a la institución que contiene detalles técnicos sobre la creación del nuevo proyecto llamado OAUTH2. Incluye instrucciones precisas de cómo ir agregando o autorizando más aplicaciones si la institución así lo llegara a necesitar.

3.2.2. Manual técnico para la implementación del módulo de atención de casos estudiantiles

Este también es un documento en formato PDF entregado a la institución que contiene detalles técnicos sobre todas la lógica seguida y las funciones

principales que pueden ser de interés para futuras extensiones que desea realizar la institución y que le dan vida a este nuevo módulo. Incluye instrucciones precisas de cómo ir filtrando las etiquetas obtenidas de GitLab que se deben mostrar a los usuarios finales según su tipo, así como cuales son los recursos habilitados.

CONCLUSIONES

1. La implementación del nuevo diseño de arquitectura de las aplicaciones orientadas a servicios como una API, le permite a la institución escalar sus recursos hacia el soporte de todo tipo de aplicaciones de una manera segura, gracias la integración con un nuevo servicio de autorización mediante el uso de *tokens* de acceso, restringiendo así únicamente hacia todas las aplicaciones que la institución desee autorizar.

2. La creación del nuevo módulo de atención de casos estudiantiles del lado del portal de la institución y en conjunto con el uso de la API de GitLab permitirá una mejor atención a todas las solicitudes administrativas incluidas los casos estudiantiles, permitiendo así dar un mejor seguimiento a estos tanto del lado del usuario final como del lado del personal administrativo.

3. Es muy importante la estandarización de algunos procesos, para la correcta adaptación y/o extensión de los servicios de la institución. De tal manera que este documento contenga todo los recursos teóricos y técnicos necesarios para que cualquier persona con conocimientos de informática pueda comprender y replicar la solución.

RECOMENDACIONES

1. Insistir que cada aplicación cliente nueva debe poseer sus propias credenciales y agregarlas dentro de la lista de aplicaciones clientes registradas que abarca los alcances de este proyecto de EPS, de tal manera que el servicio de autorización pueda reconocerlo para poder proveerle los *tokens* de acceso y así poder consumir los recursos que la institución ofrece.

2. Mencionar que no existe un límite de proyectos de GitLab que se pueden ir agregando, sin embargo, hay que tomar en cuenta que conforme vayan creciendo los proyectos, irán creciendo el número de recorridos que se deben hacer al momento de obtener la información de los *issues* en cada uno de los proyectos, afectando así directamente el tiempo de reacción de la aplicación del lado del usuario final.

3. Resaltar al personal administrativo que tiene acceso al módulo de generación de reportes la importancia de la selección correcta del filtro para el rango de fechas, tanto para la generación de los reportes de los casos estudiantiles abiertos como para la generación de los reportes de los casos estudiantiles cerrados. Tratando la manera de que el rango no sea excesivamente pequeño o al menos se recomienda que no sea menor a 8 días, o una semana, esto debido a la división que se hace de ese valor en otros 4 sub-rangos de manera automatizada, de tal manera que se siempre se presenten en cuatro columnas según indicaciones recibidas por el asesor de la institución.

4. Analizar todo este documento para adaptar cualquier otra aplicación o servicio a la arquitectura de servicios existente, ya que contiene toda la documentación técnica y teoría necesaria a conocer para poder seguir escalando sin ningún problema.

BIBLIOGRAFÍA

1. EOI, W. *Gestión de riesgos en gestión de proyectos* [en línea]. <https://www.eoi.es/wiki/index.php/GESTI%C3%93N_DE_RIESGOS_en_Gesti%C3%B3n_de_proyectos>. [Consulta: 17 Sep. 2020].

2. Muy Agile. *Product Backlog y Sprint Backlog*. [en línea]. <https://muyagile.com/product-backlog-y-sprint-backlog>. [Consulta: 17 marzo 2021].

3. Sistema de Estudios de Postgrado. *Catálogo de programas de postgrado*. [en línea] <https://sep.usac.edu.gt/cg/graphics/g/catalogo.pdf>. [Consulta: 5 de mayo 2021].

4. Paradigma. *¿Cómo securizar tus APIs con OAuth?* [en línea] <https://www.paradigmadigital.com/dev/oauth-2-0-equilibrio-y-usabilidad-en-la-securizacion-de-apis>. [Consulta: 3 de junio 2021].

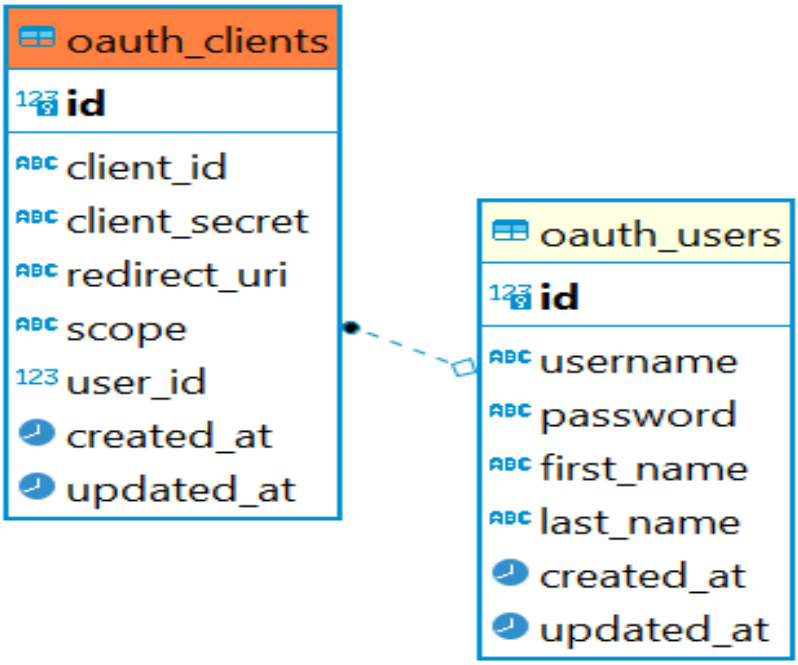
5. GitLab Docs. *Issues API.* [en línea] <https://docs.gitlab.com/ee/api/issues.html>. [Consulta: 28 de junio de 2021].

6. SHAFERM, Brent. OAuth 2.0 Server PHP. [en línea] <https://bshaffer.github.io/oauth2-server-php-docs/>. [Consulta: 9 de agosto de 2021].

7. KRUSS, Till. Predis. [en línea] <https://github.com/predis/predis [Consulta: 9 de agosto de 2021].

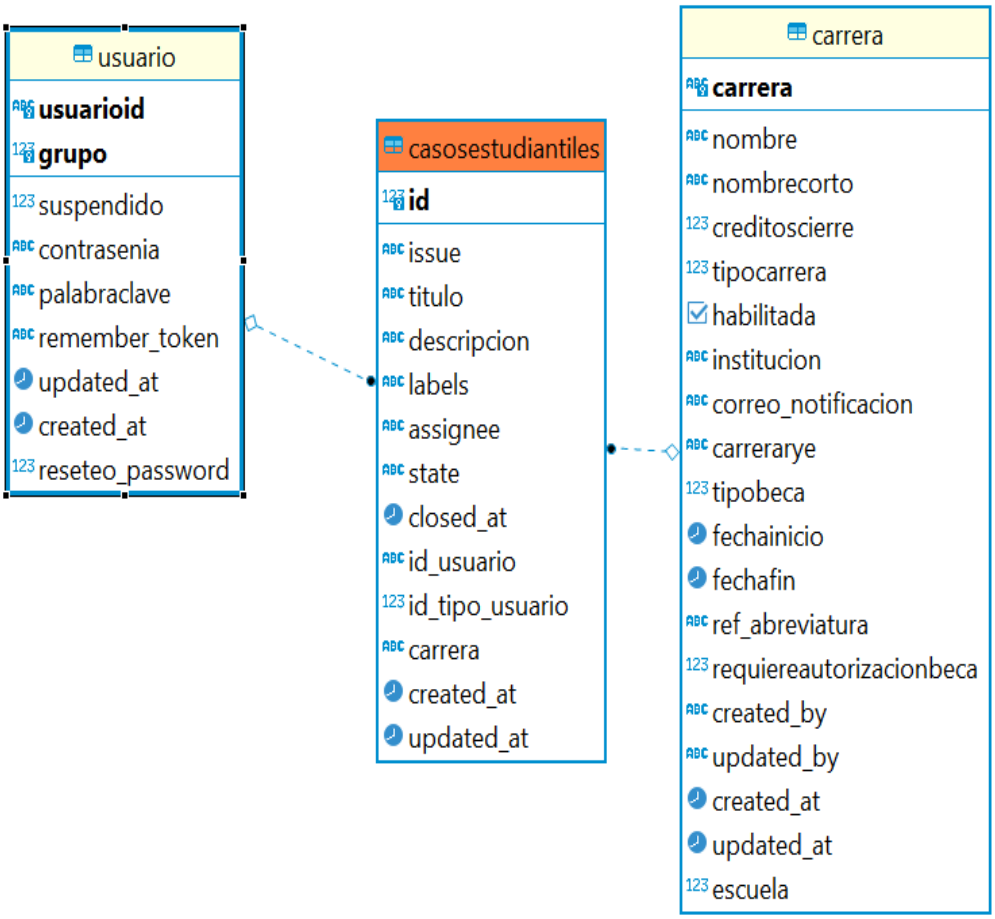
APÉNDICES

Apéndice 1. **Entidad relación para el registro de aplicaciones y usuarios del servicio de OAuth2**



Fuente: elaboración propia, empleando DBeaver.

Apéndice 2. **Entidad relación para el registro de los casos estudiantiles ingresados desde el portal de la institución**



Fuente: elaboración propia, empleando DBeaver.

Apéndice 3. **Resumen de tiempos actuales de atención de casos en el área de difusión**

DIFUSION	
Temporadas	
Noviembre	1.Dudas inscripción(10min) [1.1 primer ingreso(10mins) 1.2 reingreso(5mins)] 2. Uso de los sistemas(15mins) 3. Actualizaciones(5mins) 4. Cambios (Se recurre a informática. 12horas)
Diciembre	1.Dudas inscripción(10min) 2. Uso de los sistemas(12horas) 3. Actualizaciones(12horas) 4. Cambios (Se recurre a informática. 12horas)
Enero	1.Dudas inscripción(10min) 2. Uso de los sistemas(15mins) 3. Actualizaciones(5mins) 4. Cambios (Se recurre a informática. 12horas)
Quincena de febrero	1.Dudas inscripción(10min) 2. Uso de los sistemas(15mins) 3. Actualizaciones(5mins) 4. Cambios (Se recurre a informática. 12horas)

Fuente: elaboración propia, empleando Microsoft Word.

Apéndice 4. **Resumen de tiempos actuales de atención de casos en el área de información**

INFORMACION	
Temporadas	
Octubre/Noviembre	1. Cierres de pensum(12.5dias) 2. defensas tesis (Traslado a control académico[30mes]) 3. Predefensas de tesis (LO MISMO) 4. Certificados de cursos(24horas)
Enero/Febrero	1. Cierres de pensum(12.5dias) 4. Certificados de cursos(24horas) 3. Reincorporaciones (Traslado a control académico (3dias)), 4. Retiro maestría temporal (Traslado a Tesorería y luego a CA(15dias)) 5 Retiro maestría definitivo(LO MISMO)
Mayo/Agosto	1. Cierres de pensum(12.5dias) 4. Certificados de cursos(24horas) 3. Retiro maestría temporal (Traslado a Tesorería y luego a CA(15dias)) 5 Retiro maestría definitivo(LO MISMO), 5. regularización(Ingreso a CA luego regresa y tiempo de 15dias).

Fuente: elaboración propia, empleando Microsoft Word.

Apéndice 5. **Resumen de tiempos actuales de atención de casos en el área de control académico**

CONTROL ACADEMICO	
Temporadas	
Enero/Febrero	1. Primer ingreso(10mins) 2. Entrega de diplomas de especialización(5mins)
Resto del año	1. Seguimiento de diplomas de especialización de primer año de maestría(5mins), 2. seguimiento para actos de graduación(30dias), 3. Primer Ingreso(3mins)
Octubre/Noviembre	1. Seguimiento para el protocolo de graduación de especializaciones(45dias), 2. Graduaciones de maestría(14dias)

Fuente: elaboración propia, empleando Microsoft Word.

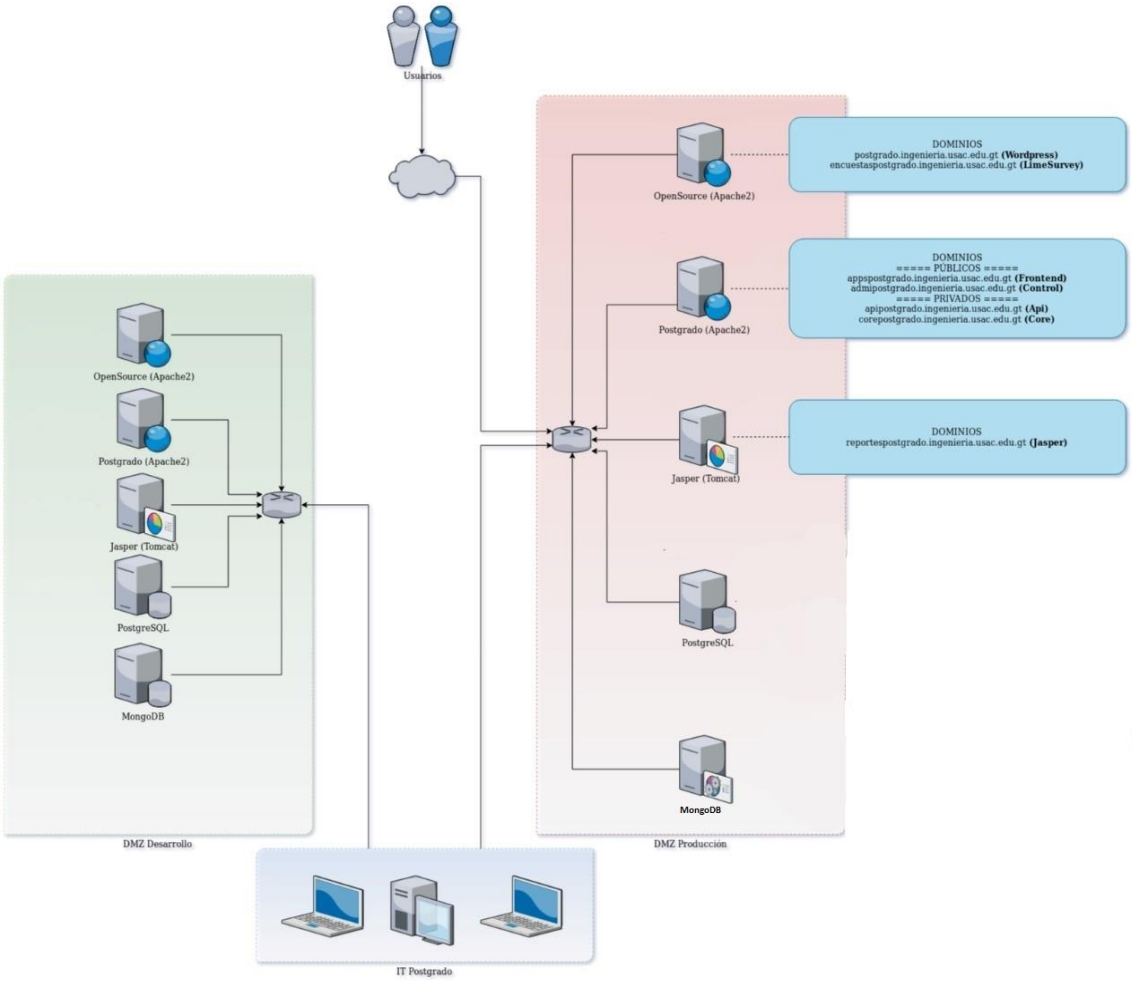
Apéndice 6. **Resumen de tiempos actuales de atención de casos en el área de Tesorería**

TESORERIA	
Temporadas	
Enero/Febrero/Marzo	1. Cierre de pensum(24horas), 2. defensas de tesis(24horas), 3. inscripciones de pregrado y postgrado(24horas) 4. Validaciones(36horas) 5. asignación extemporánea(24horas) 7. consolidación de pagos(24horas), 8. Estados de cuenta(24horas), 9. Solvencias(48horas)
Enero/Febrero/Marzo	se suman 1 semana para todos los tipos

Fuente: elaboración propia, empleando Microsoft Word.

ANEXOS

Anexo 1. **Infraestructura de la institución previo al proyecto de EPS**



Fuente: CAJAS, Luis, *Infraestructura de servidores de la institución previo a la implementación del proyecto*. Ciudad de Guatemala, USAC. Consulta: 21 de agosto de 2020.

Anexo 2. **Formulario físico de actual uso para el ingreso de solicitud de casos estudiantiles.**

FACULTAD DE INGENIERÍA USAC
ESCUELA DE ESTUDIOS DE POSTGRADO

Escuela de Estudios de Postgrado, Edificio S-11, primer nivel
Teléfono: 2418-9142 / 2418000 ext. 1382
Email: epostgrado@ingenieria.usac.edu.gt
Web: https://epostgrado.ingenieria.usac.edu.gt

SOLICITUD DE
ASUNTOS ESTUDIANTILES

Cuadro de Clasificación
EP.E-006

ASUNTO | FECHA | NRO. DE REGISTRO

DATOS DEL SOLICITANTE

NOMBRE COMPLETO | TELÉFONO

PROGRAMA | NRO. DE DPI | CARNÉ

CORREO ELECTRÓNICO

SOLICITUD
(REALIZADO POR EL ESTUDIANTE)

FIRMA DEL SOLICITANTE

Fuente: CAJAS, Luis. *Formulario para ingreso de casos estudiantiles*. Ciudad de Guatemala, USAC. Consulta: 31 de agosto de 2020.

Anexo 3. **Ejemplo de reporte de casos estudiantiles cerrados generados manualmente**

Cuenta de Description	Etiquetas de columna				
Etiquetas de fila	24/07/2021 - 30/07/2021	31/07/2021 - 6/08/2021	7/08/2021 - 13/08/2021	14/08/2021 - 19/08/2021	Total general
⊞	2	11	7	1	21
⊞ Ana MarÃ-a Morales	1	1			2
⊞ Arleni Escobar	5	6	33	28	72
⊞ David Estuardo Morales Ajcot		1	4	1	6
⊞ David Estuardo Morales Ajcot, Rene Monge			1		1
⊞ EDGAR ALVAREZ COTI	1	2	2		5
⊞ Eduardo Soto	1	4	3	6	14
⊞ Flor de MarÃ-a Morales		28	48	27	103
⊞ Gabriela Guzman			1		1
⊞ geilin osiris Escobar Montes	1		1	2	4
⊞ GitLab Support Bot	1				1
⊞ Lissa Maria Corzantes Martinez	1	9	4	3	17
⊞ Luis Cajas	2	3	3		8
⊞ Luis Cajas, David Estuardo Morales Ajcot			1		1
⊞ Luis Ricardo Hernandez		3			3
⊞ MANUEL DE JESUS ELIAS TEJAX		4	4	3	11
⊞ MANUEL DE JESUS ELIAS TEJAX, Arleni Escobar		1			1
⊞ Pamela Gonzalez				1	1
⊞ Rene Monge	1				1
Total general	16	73	112	72	273

Fuente: CAJAS, Luis. *Ejemplo de reporte de casos estudiantiles generados manualmente en excel*. Ciudad de Guatemala, USAC. Consulta: 25 de agosto de 2021.