UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

PRIMER SEMESTRE 2023

LABORATORIO SOFTWARE AVANZADO

INFING-USAC

<u>INTEGRANTES</u>

CARNE	ESTUDIANTE	PARTICIPACIÓN
201901510	Pablo Daniel Rivas Marroquin	100%
201902934	German Jose Paz Cordon	100%
201903850	Adrian Samuel Molina Cabrera	100%
201900955	Diego Fernando Cortez Lopez	100%

GUATEMALA 11 DE FEBRERO DE 2023

MICROSERVICIOS

A continuación se muestran todos los aspectos de cada módulo que se asignaron a los desarrolladores para realizar sus microservicios.

DESARROLLADOR	**MÓDULO**
Pablo Daniel Rivas Marroquin	● Auth ● Pensum
German Jose Paz Cordon	● Calendario-Evento ● CRUD Docente
Adrian Samuel Molina Cabrera	● Perfil ● Comunidad
Diego Fernando Cortez Lopez	● CRUD Cursos ● CRUD Horarios de Semestre

Código de respuestas exitosas

Se utilizaron la siguiente lista de errores

Código	Descripción
200	Solicitud aceptada; la respuesta contiene el resultado

Código de respuestas fallidas

Se utilizaron la siguiente lista de errores

Código	Descripción
400	La solicitud no fue válida. Este código se devuelve cuando el servidor ha intentado procesar la solicitud
500	Se ha producido un error interno en el servidor

Cuerpo de Token

```
{
 "carne": "201901510",
 "password": "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4",
 "exp": 1677041618
}
```

LOGIN

Este microservicio fue elegido debido a que necesitamos obtener un token para validar la existencia del usuario y realizar las operaciones dentro de la plataforma.

ID: 001	**Nombre:** Microservicio acceso de usuarios
Prioridad: Alta	**Historia de usuario:** Como desarrollador quiero poder validar la autenticación de los usuarios para el ingreso a la plataforma.
Estimado: 4 puntos	
Módulo: auth	

Criterio de aceptación:
Si el usuario existe y las credenciales ingresadas son válidas debe devolver un token proveniente de JWT para la autenticación.

El servicio debe de tener la siguiente configuración

Ruta: auth/login

Método: POST

Descripción: Este endpoint su función es permitir el acceso a los usuarios.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json

Body:

Atributo	**Tipo**	**Descripción**
carne	String	Correo del usuario
password	String	Contraseña del usuario

Ejemplo de body de entrada:

```
{
	"carne" : "201901510",
	"password" : "1234"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
token	String	Token de autenticación

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"token" : "d45a98d4a6d58nfuf9837489vfsmoiksa4d54a..."
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "Usuario no encontrado"
}
```

REGISTRY

Este microservicio fue elegido porque se necesita una forma para registrar a los nuevos usuarios.

ID: 002	**Nombre:** Microservicio Formulario de registro
Prioridad: Alta	**Historia de usuario:** Como desarrollador quiero que los usuarios introduzcan sus datos personales solicitados por medio de un formulario.
Estimado: 3 puntos	
Módulo: auth	

Criterio de aceptación:
El formulario ingresado debe llevar los datos a que se inserten en la base de datos.

El servicio debe de tener la siguiente configuración

Ruta: auth/registry

Método: POST

Descripción: Este endpoint es el encargado de recopilar la información necesaria para la inserción de un usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json

Body:

Atributo	**Tipo**	**Descripción**
name	String	Nombre del usuario.
lastname	String	Apellido del usuario.
carne	Integer	Carné universitario del usuario.
cui	String	CUI del usuario.
email	String	Correo electrónico del usuario.
password	String	Contraseña del usuario.
fecha_nac	String	Fecha de nacimiento del usuario.
cel	String	Número de teléfono del usuario.

Ejemplo de body de entrada:

```
{
	"name" : "Pablo Daniel",
	"lastname" : "Rivas Marroquin",
	"carne" : 201901510,
	"cui" : "3657569460101",
	"email" : "pdanielr225@gmail.com",
	"password" : "DWAdsad45664s",
	"fecha_nac" : "28/02/2000",
	"cel" : "57391252"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripción de la respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"descripcion" : "Se ha registrado Correctamente"
	"status" : 200
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El carné ingresado ya se encuentra registrado"
}
```

SEND-PENSUM-USER

Este microservicio fue creado bajo la necesidad de poder registrar los cursos que el usuario se encuentra actualmente cursando durante el semestre.

ID: 003	**Nombre:** Microservicio Guardar mis cursos asignados
Prioridad: Alta	**Historia de usuario:** Como usuario quiero guardar los cursos que llevaré durante el semestre
Estimado: 2 puntos	
Módulo: pensum	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné y código de curso enviados para guardar la información del usuario

El servicio debe de tener la siguiente configuración

Ruta: pensum/sendPensumUser

Método: POST

Descripción: Este endpoint es el encargado de agregar los cursos que el usuario va a llevar durante el semestre actual.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	String	Código del curso

Ejemplo de body de entrada:

```
{
	"code_course" : "0780"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripción de la respuesta

Ejemplo de parámetros de salida exitosa:

```
{
      "descripcion" : "se ha asignado correctamente el curso"
      "status" : 200
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 400,
      "description" : "El curso ya se encuentra asociado al usuario"
}
```

GET-PENSUM-USER

Este microservicio fue creado bajo la necesidad de poder obtener todos los cursos a los que el usuario se encuentra asignado actualmente durante el semestre.

ID: 004	**Nombre:** Microservicio Obtener mis cursos asignados
Prioridad: Baja	**Historia de usuario:** Como usuario quiero obtener los cursos que tengo asignados para el semestre actual.
Estimado: 2 puntos	
Módulo: Pensum	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné enviado para obtener la lista de cursos asignados.

El servicio debe de tener la siguiente configuración

Ruta: pensum/getPensumUser

Método: GET

Descripción: Este endpoint es el encargado de devolver todos los cursos que el usuario haya marcado como asignados.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
courses	[String]	Lista de cursos asignados

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "courses" : [
                    {
                      "code_course" : 0780,
                      "name_course" : "Software Avanzado"
                    }
                  ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 400,
      "description" : "El usuario no ha sido encontrado"
}
```

GET-ALL-DOCENTE

Este microservicio fue creado bajo la necesidad de poder obtener la información de todos los docentes que se encuentran registrados.

ID: 005	**Nombre:** Microservicio obtener los docentes
Prioridad: Alta	**Historia de usuario:** Como desarrollador quiero obtener una lista con los docentes registrados.
Estimado: 1 punto	
Módulo: crudDocente	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto podrá obtener la lista de cursos asignados.

El servicio debe de tener la siguiente configuración

Ruta: crudDocente/getAllcrudDocente

Método: GET

Descripción: Este endpoint permite obtener a todos los docentes registrados.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
lista_docente	[String]	Lista de docentes registrados

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "lista_docente" : [
           {
              "id" : 2
              "email" : "germanpc9@gmail.com",
              "name" : "German José",
              "lastname" : "Paz Cordón"
           }
      ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 400,
      "description" : "El usuario no ha sido encontrado"
}
```

ADD-DOCENTE

Este microservicio fue creado bajo la necesidad de poder registrar la información de un nuevo docente.

ID: 006	**Nombre:** Microservicio Agregar Docentes
Prioridad: Alta	**Historia de usuario:** Como desarrollador quiero un formulario donde se puedan registrar los datos del docente
Estimado: 3 puntos	
Módulo: crudDocente	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará los parámetros enviados para registrar a un docente.

El servicio debe de tener la siguiente configuración

Ruta: crudDocente/addcrudDocente

Método: POST

Descripción: Este endpoint permite la creación de nuevos docentes.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
email	String	Correo electrónico del docente.
name	String	Nombres del docente.
lastname	String	Apellidos del docente

Ejemplo de body de entrada:

```
{
	"email" : "germanpc90@gmail.com",
	"name" : "German José",
	"lastname" : "Paz Cordón"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripción de la respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200
	"descripcion": "Se ha insertado el docente correctamente”
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El docente ya existe."
}
```

REMOVE-DOCENTE

Este microservicio fue creado bajo la necesidad de eliminar a un docente existente.

ID: 007	**Nombre:** Microservicio Remover Docentes
Prioridad: Media	**Historia de usuario:** Como desarrollador quiero una opcionalidad donde se pueden eliminar docentes registrados
Estimado: 2 puntos	
Módulo: crudDocente	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto se debe de remover de la base de datos.

El servicio debe de tener la siguiente configuración

Ruta: crudDocente/removecrudDocente

Método: DELETE

Descripción: Este endpoint permite eliminar docentes registrados.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
email	String	Correo electrónico del docente.

Ejemplo de body de entrada:

```
{
	"email" : "germanpc9@gmail.com"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripción de la respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion": "Se ha eliminado el docente correctamente”
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El docente no existe"
}
```

GET-ALL-EVENT

Este microservicio fue creado bajo la necesidad de poder obtener todos los eventos que se encuentran registrados y calendarizados.

ID: 008	**Nombre:** Microservicio Obtener eventos
Prioridad: Media	**Historia de usuario:** Como usuario quiero obtener los eventos que están calendarizados.
Estimado: 2 puntos	
Módulo: calendarioEvento	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto devolverá todos los eventos que están registrados en base de datos.

El servicio debe de tener la siguiente configuración

Ruta: calendarioEvento/getAllEvent

Método: GET

Descripción: Este endpoint permite obtener todos los eventos creados.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
list_eventos	[String]	Lista de eventos creados

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "list_eventos" : [
           {
              "id" : 1,
              "carne" : 201902934,
              "title" : "Conferencia SOA",
              "msg" : "Se realizará una conferencia hablando sobre…",
              "fecha" : "11/02/2023"
           }
      ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Falla de conexión con el servidor"
}
```

SEND-EVENT

Este microservicio fue creado bajo la necesidad de poder registrar un nuevo evento creado por un usuario.

ID: 009	**Nombre:** Microservicio registrar evento
Prioridad: Alta	**Historia de usuario:** Como usuario quiero enviar un evento por medio de un formulario que permita registrar los datos solicitados.
Estimado: 2 puntos	
Módulo: calendarioEvento	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto debe de registrar el evento por medio del formulario e insertarse en base de datos.

El servicio debe de tener la siguiente configuración

Ruta: calendarioEvento/sendEvent

Método: POST

Descripción: Este endpoint permite la creación de nuevos eventos.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
title	String	Titulo del evento
msg	String	Mensaje del evento
fecha	String	Fecha del evento

Ejemplo de body de entrada:

```
{
    "title" : "Conferencia SOA",
    "msg" : "Se realizará una conferencia hablando sobre...",
    "fecha" : "11/02/2023"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripción de la respuesta

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "descripcion": "Se agrego el evento correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 400,
      "description" : "El carnet no existe."
}
```

GESTIONAR-EVENTO

Este microservicio fue creado bajo la necesidad de realizar dos gestiones entre los eventos y usuarios. La primera enfocada a la asignación de un usuario a un evento y la otra gestión la desasignación de un usuario a un evento.

ID: 010	**Nombre:** Microservicio gestión de eventos
Prioridad: Alta	**Historia de usuario:** Como usuario quiero asignar o desasignar eventos.
Estimado: 1 puntos	
Módulo: calendarioEvento	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto se debe registrar en base de datos que se haya asignado un usuario a un evento existente.

El servicio debe de tener la siguiente configuración

Ruta: calendarioEvento/gestionarEvent

Método: POST

Descripción: Este endpoint permite asignar o desasignar un evento a un usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
id_evento	Integer	Identificador del evento.
carne	Integer	carne del usuario.
opcion	Integer	Operación a realizar (1: asignar, 0: desasignar)

Ejemplo de body de entrada:

```
{
	"id_evento" : 1,
	"carne" : 201902934,
	"opcion" : 1
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de retorno

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200
	"descripcion" : "asignado al evento"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "No existe el evento"
}
```

ELIMINAR-EVENTO

Este microservicio fue creado bajo la necesidad de poder eliminar un evento que se encuentra creado.

ID: 011	**Nombre:** Microservicio eliminar evento
Prioridad: Media	**Historia de usuario:** Como usuario quiero eliminar eventos que estén existentes
Estimado: 1 puntos	
Módulo: calendarioEvento	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto remueve un evento se elimine el registro de la base de datos.

El servicio debe de tener la siguiente configuración

Ruta: calendarioEvento/eliminarEvent

Método: DELETE

Descripción: Este endpoint permite eliminar un evento.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
id_evento	Integer	Identificador del evento.

Ejemplo de body de entrada:

```
{
	"id_evento" : 1
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "evento eliminado correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "No existe el evento"
}
```

UPDATE-USER-INFO

Este microservicio fue creado bajo la necesidad de poder actualizar la información almacenada de un usuario.

ID: 012	**Nombre:** Microservicio Actualizar mi información
Prioridad: Media	**Historia de usuario:** Como usuario quiero poder actualizar mi información para visualizarla en mi perfil.
Estimado: 2 puntos	
Módulo: Perfil	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará los parámetros enviados en el body para realizar la actualización de información

El servicio debe de tener la siguiente configuración

Ruta: perfil/updateUser

Método: PUT

Descripción: La función de este endpoint es permitir editar la información del usuario..

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
name	String	Nombre del usuario
lastname	String	Apellido del usuario
email	String	Email del usuario
fecha_nac	String	Fecha de nacimiento del usuario
cel	String	Número de celular del usuario
password	String	Contraseña del usuario

Ejemplo de body de entrada:

```
{
	"name" : "Adrian",
	"lastname" : "Molina",
	"email" : "admin@gmail.com",
	"fecha_nac" : "20/12/2000",
	"cel" : "55555151",
	"password" : "admin1234"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "usuario actualizado correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "Usuario no existente"
}
```

GET-USER-INFO

Este microservicio fue creado bajo la necesidad de obtener la información sobre un usuario que fue registrada en su creación.

ID: 013	**Nombre:** Microservicio Obtener mi información
Prioridad: Baja	**Historia de usuario:** Como usuario quiero obtener la información que ingresé al registrarme.
Estimado: 1 puntos	
Módulo: Perfil	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné enviado para obtener la información del usuario

El servicio debe de tener la siguiente configuración

Ruta: perfil/getUser

Método: GET

Descripción: La función de este endpoint es obtener la información del usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
carne	Integer	Carnet del usuario
name	String	Nombre del usuario
lastname	String	Apellido del usuario
email	String	Email del usuario
fecha_nac	String	Fecha de nacimiento del usuario
cel	String	Número de celular del usuario
cui	String	Dpi del usuario

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200
	"carne" : 201903800,
	"name" : "Adrian",
	"lastname" : "Molina",
	"email" : "admin@gmail.com",
	"fecha_nac" : "20/12/2000",
	"cel" : "55555151",
	"cui" : "3020721280105"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "Usuario no existente"
}
```

GET-CURSOS-APROBADOS

Este microservicio fue creado bajo la necesidad de poder obtener el listado de todos los cursos que el usuario ha marcado como aprobados.

ID: 014	**Nombre:** Microservicio obtener mis cursos aprobados
Prioridad: Baja	**Historia de usuario:** Como usuario quiero obtener un listado de todos los cursos que he marcado como aprobados.
Estimado: 1 punto	
Módulo: Perfil	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné enviado para obtener las lista de cursos aprobados del usuario.

El servicio debe de tener la siguiente configuración

Ruta: perfil/getCursos

Método: GET

Descripción: La función de este endpoint es obtener un listado de cursos ya aprobados por el usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
total_credits	Integer	Número total de créditos
courses	[String]	listado de cursos aprobados por el usuario

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200
      "total_credits" : 9,
      "courses" : [
                {"code" : 101, "name" : "Deportes 1", "credits": 1},
                {"code" : 348, "name" : "Química", "credits": 3},
                {"code" : 147, "name" : "Física", "credits": 5},
                ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 400,
      "description" : "Usuario no existente"
}
```

SET-CURSO-APROBADO

Este microservicio fue creado bajo la necesidad de que los usuarios puedan seleccionar y almacenar qué cursos han aprobado.

ID: 015	**Nombre:** Microservicio Guardar curso aprobado
Prioridad: Alta	**Historia de usuario:** Como usuario quiero guardar cursos que tengo aprobados**.**
Estimado: 2 puntos	
Módulo: Perfil	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné y código de curso enviados para guardar la información de cursos aprobados del usuario.

El servicio debe de tener la siguiente configuración

Ruta: perfil/setCurso

Método: POST

Descripción: La función de este endpoint es almacenar los cursos ya aprobados por el usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	Integer	Código del curso

Ejemplo de body de entrada:

```
{
	"code_course" : 101
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "Se ha asignado el curso correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El curso no existe"
}
```

DELETE-CURSOS

Este microservicio fue creado bajo la necesidad de que los usuarios puedan eliminar cursos de su propia lista de cursos que previamente se han seleccionado como aprobados.

ID: 016	**Nombre:** Eliminar curso aprobado
Prioridad: Alta	**Historia de usuario:** Como usuario quiero eliminar cursos aprobados de mi lista.
Estimado: 2 puntos	
Módulo: Perfil	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto usará el carné y codigo de curso enviados para eliminar la información de la lista de cursos.

El servicio debe de tener la siguiente configuración

Ruta: perfil/deleteCurso

Método: DELETE

Descripción: La función de este endpoint es eliminar un curso que se encuentre en la lista de cursos aprobados por el usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	Integer	Código del curso

Ejemplo de body de entrada:

```
{
	"code_course" : 101
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "Se ha eliminado el curso correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El curso no se encuentra asociado al usuario"
}
```

SET-POST

Este microservicio fue creado bajo la necesidad de poder registrar y almacenar las nuevas publicaciones que los usuarios crean.

ID: 017	**Nombre:** Crear Posts para la comunidad
Prioridad: Alta	**Historia de usuario:** Como usuario quiero crear publicaciones para que los demás lean mis consejos y avisos.
Estimado: 2 puntos	
Módulo: Comunidad	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto usarán los parámetros enviados para guardar los post del usuario.

El servicio debe de tener la siguiente configuración

Ruta: comunidad/setPost

Método: POST

Descripción: La función de este endpoint es almacenar los post creados por el usuario.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
title	String	Titulo del post
msg	String	Cuerpo del post
tag	String	Etiqueta principal del post

Ejemplo de body de entrada:

```
{
	"title" : "Iniciar Redes de Computadoras 1 de mejor forma",
	"msg" : "Tener instalado gns3, aprender lo básico sobre dispositivos cisco, …",
	"tag" : "Redes"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "Se ha creado el post correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 400,
	"description" : "El usuario no existe"
}
```

GET-POST

Este microservicio fue creado bajo la necesidad de poder obtener todos los post que se han creado y registrado que cumplan con un tema en específico.

ID: 018	**Nombre:** Obtener los post creados
Prioridad: Baja	**Historia de usuario:** Como usuario quiero ver todos los post creados y buscar por un tema en específico.
Estimado: 1 punto	
Módulo: Comunidad	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto podrá obtener la información sobre los post creados.

El servicio debe de tener la siguiente configuración

Ruta: comunidad/getPost

Método: GET

Descripción: La función de este endpoint es obtener los post creados por los usuarios registrados..

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
tag	String	Parámetro opcional para recuperar únicamente posts con el tag principal solicitado.

Ejemplo de body de entrada:

```
{
    "tag" : "Redes"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
posts	[String]	Lista de post creados por los usuarios

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "posts" : [
                  {
                    "carne" : "201903800"
                    "title" : "Iniciar Redes de Computadoras 1
                               de mejor forma",
                    "msg" : "Tener instalado gns3, aprender lo
                             básico sobre dispositivos cisco, …",
                    "tag" : "Redes"
                  }
                ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Ha fallado la conexión"
}
```

SET-CURSOS

Este microservicio fue creado bajo la necesidad de poder crear y agregar nuevos cursos, almacenando toda su información.

<table>
<tr><td>ID: 019</td><td>Nombre: Registrar cursos a la plataforma</td></tr>
<tr><td>Prioridad: Alta</td><td rowspan="3">Historia de usuario:
Como administrador quiero poder agregar cursos, con su información, a la plataforma.</td></tr>
<tr><td>Estimado: 3 puntos</td></tr>
<tr><td>Módulo: CRUD Cursos</td></tr>
<tr><td colspan="2">Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto se podrá registrar un curso en la base de datos

El servicio debe de tener la siguiente configuración:
Ruta: crudCursos/setcurso
Método: POST
Descripción: La función de este endpoint es registrar los cursos a utilizar en la plataforma.

Formato de entrada: JSON
Header:
<table>
<tr><th>Atributo</th><th>Tipo</th><th>Descripción</th></tr>
<tr><td>Content type</td><td>header</td><td>application/json</td></tr>
<tr><td>Token</td><td>header</td><td>token <TOKEN></td></tr>
</table>
Body:
<table>
<tr><th>Atributo</th><th>Tipo</th><th>Descripción</th></tr>
<tr><td>code_course</td><td>String</td><td>Código del curso</td></tr>
<tr><td>name_course</td><td>String</td><td>Nombre del curso</td></tr>
<tr><td>credit_course</td><td>Integer</td><td>Créditos del curso</td></tr>
<tr><td>pre_courses</td><td>[String]</td><td>Lista de códigos pre-requisito</td></tr>
<tr><td>optional</td><td>Integer</td><td>Parámetro para verificar si el curso es obligatorio.</td></tr>
</table>
</td></tr>
</table>

Ejemplo de body de entrada:

```
{
    "code_course" : "0780",
    "name_course" : "Software Avanzado",
    "credit_course" : 8,
    "pre_courses" : ["708", "901"],
    "optional" : 1
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "descripcion" : "Se ha creado el curso correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Ha fallado la conexión"
}
```

DELETE-CURSOS

Este microservicio fue creado bajo la necesidad de eliminar cursos que se encuentran creados y registrados en la plataforma.

ID: 020	**Nombre:** Eliminar un curso en la base de datos
Prioridad: Media	**Historia de usuario:** Como administrador quiero eliminar cursos en la plataforma.
Estimado: 2 puntos	
Módulo: CRUD Cursos	

Criterio de aceptación:

Se enviará un token para validar la autenticidad del usuario, y de ser correcto podrá eliminar un curso registrado en la base base de datos

El servicio debe tener la siguiente configuración:

Ruta: crudCursos/deleteCurso

Método: DELETE

Descripción: La función de este endpoint elimina cursos registrados.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	String	Código del curso

Ejemplo de body de entrada:

```
{
    "code_course" : "0780"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "Se ha eliminado el curso correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 500,
	"description" : "Ha fallado la conexión"
}
```

GET-ALL-CURSOS

Este microservicio fue creado bajo la necesidad de poder obtener todos los cursos registrados así como también toda la información que contiene.

ID: 021	**Nombre:** Agregar Cursos a plataforma
Prioridad: Baja	**Historia de usuario:** Como usuario quiero obtener la lista de los cursos registrados en la plataforma con la información de cada uno.
Estimado: 2 puntos	
Módulo: CRUD Cursos	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto devolverá todos los cursos que están registrados en la base de datos.

El servicio debe tener la siguiente configuración:
Ruta: crudCursos/getAllCursos
Método: GET
Descripción: La función de este endpoint devuelve todos los cursos registrados.

Formato de entrada: JSON
Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
courses	[String]	Lista de los cursos almacenados

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "course" : [
                  {
                    "code_course" : 0780,
                    "name_course" : "Software Avanzado",
                    "credit_course" : 8,
                    "pre_course" : ["0785"],
                    "optional" : 1
                  }
                ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Ha fallado la conexión"
}
```

ADD-HORARIO-CURSOS

Este microservicio fue creado bajo la necesidad de poder asignar horarios para los cursos en un respectivo semestre.

ID: 022	**Nombre:** Agregar horarios a los cursos del semestre
Prioridad: Alta	**Historia de usuario:** Como administrador quiero agregar un horario para los cursos del semestre.
Estimado: 3 puntos	
Módulo: CRUD Horarios de Semestre	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto inserta los horarios con los parámetros enviados.

El servicio debe tener la siguiente configuración:
Ruta: horariosSemestre/addHorarioCurso
Método: POST
Descripción: La función de este endpoint es la de agregar horarios a los cursos para el semestre.

Formato de entrada: JSON
Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	Integer	Código del curso
code_doce	Integer	Código del docente
section	String	Sección del curso
hour_init	String	La hora en que inicia la clase
hour_finish	String	La hora en que finaliza la clase
dates	[String]	Lista de días de la seman el cual se impartirá el curso

Ejemplo de body de entrada:

```
{
	"code_course" : "0780",
	"code_doce" : 4,
	"section" : "N",
	"hour_init" : "19:00",
	"hour_finish" : "20:40",
	"dates" : ["Lunes","Miercoles"]
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
	"status" : 200,
	"descripcion" : "Se ha agregado el horario correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	**Tipo**	**Descripción**
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
	"status" : 500,
	"description" : "Ha fallado la conexión"
}
```

REMOVE-HORARIO-CURSOS

Este microservicio fue creado bajo la necesidad para poder remover y eliminar los horarios que le fueron asignados a los cursos.

ID: 023	**Nombre:** Remover horario de curso del semestre
Prioridad: Medio	**Historia de usuario:** Como administrador quiero eliminar el horario de un curso en el semestre.
Estimado: 2 puntos	
Módulo: CRUD Horarios de Semestre	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto podrá eliminar un horario para un curso en el semestre.

El servicio debe tener la siguiente configuración:

Ruta: horariosSemestre/removeHorarioCurso

Método: DELETE

Descripción: La función de este endpoint es eliminar los cursos en la lista de horarios de cursos para el semestre.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**
code_course	Interger	Código del curso
section	String	Sección del curso

Ejemplo de body de entrada:

```
{
    "code_course" : "0780",
    "section" : "N"
}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
descripcion	String	Descripcion de respuesta

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "descripcion" : "Se ha eliminado el horario correctamente"
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Ha fallado la conexión"
}
```

GET-HORARIO-CURSOS

Este microservicio fue creado bajo la necesidad de obtener el listado de horarios para los cursos que se encuentran registrados en el semestre.

ID: 024	**Nombre:** Obtener horarios de cursos del semestre
Prioridad: Baja	**Historia de usuario:** Como usuario quiero obtener un listado de horarios de los cursos registrados en el semestre.
Estimado: 3 puntos	
Módulo: CRUD Horarios de Semestre	

Criterio de aceptación:
Se enviará un token para validar la autenticidad del usuario, y de ser correcto devolverá todos los horarios de cursos que están registrados en el semestre.

El servicio debe tener la siguiente configuración:

Ruta: horariosSemestre/getHorariosCursos

Método: GET

Descripción: La función de este endpoint tiene la de devolver todos los cursos registrados en los horarios para el semestre.

Formato de entrada: JSON

Header:

Atributo	**Tipo**	**Descripción**
Content type	header	application/json
Token	header	token <TOKEN>

Body:

Atributo	**Tipo**	**Descripción**

Ejemplo de body de entrada:

```
{

}
```

Formato de salida: JSON

Código de respuesta exitosa: HTTP 200

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
courses	[String]	Lista de los cursos almacenados

Ejemplo de parámetros de salida exitosa:

```
{
      "status" : 200,
      "course" : [
                  {
                    "code_course" : "0780",
                    "name_course" : "Software Avanzado",
                    "credit_course" : 8,
                    "pre_course" : ["0785"],
                    "optional" : FALSE
                  }
                ]
}
```

Formato de salida: JSON

Código de respuesta fallida: HTTP 400, HTTP 500

Atributo	Tipo	Descripción
status	Integer	Código de respuesta
description	String	Descripción del error.

Ejemplo de parámetros de salida fallida:

```
{
      "status" : 500,
      "description" : "Ha fallado la conexión"
}
```