



INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
Unidad Profesional "Adolfo López Mateos" ZACATENCO

**“SISTEMA DE GESTIÓN ESCOLAR QUE PERMITA
LA ADMINISTRACIÓN DE PERSONAL DOCENTE,
ADMINISTRATIVO Y ESTUDIANTEL DENTRO DE UNA
INSTITUCIÓN ACADÉMICA DE MANERA REMOTA”**

REPORTE TÉCNICO

**PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**

PRESENTA: CÉSAR RODRÍGUEZ CALDERÓN

ASESORES:

ING. ARTURO ALONSO HIT ESPINOSA

ING. GUILLERMO MANUEL ESCÁRCEGA CARRERA



CIUDAD DE MEXICO, DICIEMBRE 2020



INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

REPORTE TÉCNICO

QUE PARA OBTENER EL TÍTULO DE
POR LA OPCIÓN DE TITULACIÓN

INGENIERO EN COMUNICACIONES Y ELECTRÓNICA
SEMINARIO REG. DES/ESIMEZAC/S/451-33/2008-2020

DEBERA (N) DESARROLLAR

C. CESAR RODRÍGUEZ CALDERÓN

**"SISTEMA DE GESTIÓN ESCOLAR QUE PERMITA LA ADMINISTRACIÓN DE PERSONAL DOCENTE,
ADMINISTRATIVO Y ESTUDIANTEL DENTRO DE UNA INSTITUCIÓN ACADÉMICA DE MANERA REMOTA"**

**DISEÑAR UN SISTEMA INFORMÁTICO DE CONTROL ESCOLAR EN LÍNEA QUE PERMITA
ADMINISTRAR LA GESTIÓN DE PERSONAL EN UNA INSTITUCIÓN ACADÉMICA**


- ❖ MARCO CONCEPTUAL
- ❖ ANÁLISIS
- ❖ DISEÑO
- ❖ IMPLEMENTACIÓN

CIUDAD DE MÉXICO, A 19 DE DICIEMBRE DE 2020

ASESORES


ING. ARTURO A. HIT ESPINOSA


ING. GUILLERMO M. ESCARCEGA CARRERA


M. EN C. ITZALÁ RABADAN MALDA
JEFA DEL DEPARTAMENTO DE INGENIERÍA
EN COMUNICACIONES Y ELECTRÓNICA





Autorización de uso de obra

Instituto Politécnico Nacional

Presente

Bajo Protesta de decir la verdad el que suscribe **CÉSAR RODRÍGUEZ CALDERÓN**, manifiesto ser el autor y titular de los derechos morales y patrimoniales de la obra titulada **“SISTEMA DE GESTIÓN QUE PERMITA LA ADMINISTRACIÓN DE PERSONAL DOCENTE, ADMINISTRATIVO Y ESTUDIANTEL DENTRO DE UNA INSTITUCIÓN ACADÉMICA DE MANERA REMOTA.” La Tesis**” y de la cual se **adjunta copia en dos cd’s**, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor otorga a el **Instituto Politécnico Nacional**, en adelante **El IPN**, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales o impreso para consulta **“La Tesis”** por un periodo de **1 año** contando a partir de la fecha de la presente autorización dicho periodo se renovara automáticamente en caso de no dar aviso expreso a **El IPN** de su terminación.

En virtud de lo anterior **El IPN** deberá reconocer en todo momento nuestra calidad de autor de **“La Tesis”**:

Adicionalmente y en mi calidad de autor y titular de los derechos morales y patrimoniales de **“La Tesis”**, manifesté que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de **“La Tesis”**, por lo que deslindamos de toda responsabilidad a **El IPN** en caso de que el contenido de **“La Tesis”** o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumimos las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

México, Ciudad de México., a 26 de febrero de 2021.

Atentamente

César Rodríguez Calderón



ÍNDICE

OBJETIVO	6
JUSTIFICACIÓN	7
INTRODUCCIÓN	8
I. MARCO CONCEPTUAL	9
1.1 Middleware como herramienta para la comunicación entre servicios web..9	
1.2 Servicios REST	9
1.3 Interfaz de Aplicación Programable (API)	14
1.3.1 Elementos de la plataforma MuleSoft.....	16
1.4 Bases de datos	18
1.4.1 Bases de datos operacionales	18
1.4.2 Bases de datos analíticas	18
1.4.3 Bases de datos relacionales	19
1.4.4 Componentes de una base de datos.....	19
1.4.5 Tipos de relaciones en una base de datos	22
1.5 Infraestructura de portal web	23
1.5.1 HTML	23
1.5.2 NodeJs	24
1.5.3 Desarrollo del servidor con express	25
II. ANÁLISIS	26
2.1 Desventajas de los sistemas actuales de servicios remotos	26
2.2 Propuesta de solución por medio de una API	27
2.2.1 Capas lógicas de la conectividad API-Led	28



III	DISEÑO	29
3.1	Arquitectura	29
3.2	Operatividad para cada agente	30
3.3	Propuesta de portal web	31
3.4	Recursos REST propuestos y funcionalidad	35
3.4.1	Herramientas de software	37
3.5	Estructuras de datos	38
3.5.1	Diccionario de datos	39
3.5.2	Diseño de la base de datos	42
IV	IMPLEMENTACIÓN	43
4.1	Especificación RAML	43
4.1.1	Desarrollo de flujos	45
4.1.2	Enlace con base de datos	49
4.2	Desarrollo de la Base de datos	52
4.3	Desarrollo de portal web	54
	CONCLUSIONES	58
	BIBLIOGRAFIA	59



OBJETIVO

Desarrollar un sistema de control escolar en línea adaptable a cualquier tipo de institución académica pública o privada para ayudarlos al alta de profesores y/o alumnos, actualización de contactos, obtener información de las materias, asignación de profesores y permitir a los alumnos inscribirse a ellas con el fin de promover la administración de tal institución de una manera remota.



JUSTIFICACIÓN

Cada vez es más importante para las organizaciones operar en un entorno de plataforma virtual, en esta época aquellas instituciones que han emigrado la gestión de sus procesos en este sentido tienen una mayor probabilidad de adaptarse a la evolución tecnológica y a sobrevivir con el paso del tiempo gracias a que brindan mayor eficiencia en sus operaciones y a una estrategia efectiva ante problemas trascendentes como la pandemia COVID-19 en la que actualmente vivimos y en dónde se busca disminuir al máximo el tiempo que una persona está presente en el espacio público.¹

Por otra parte, este desarrollo puede ser utilizado como base práctica y metodológica por cualquier organización que busque la virtualización de sus procesos administrativos ya que podría ocupar el desarrollo práctico y estructural para adaptarlo a sus necesidades más específicas.



INTRODUCCIÓN

En este trabajo se describe cada uno de los elementos que forman la arquitectura del servicio, los cuales son el desarrollo de un portal web, un API Gateway en donde se implementa el esquema de seguridad OAuth 2.0 para mitigar vulnerabilidades en el proceso de autenticación, el desarrollo de una API usando el bus de datos MuleSoft para definir los recursos y métodos necesarios en dónde también se garantiza que la información sea correcta antes de llevar a cabo cualquier acción u operación en la base de datos por medio de validaciones de estructuras de mensajes y/o cuerpos, finalmente se detalla la conexión y desarrollo de una base de datos para el almacenamiento del servicio.

Antes de empezar a desarrollar un proyecto es importante conocer y estudiar los conceptos teóricos que están en juego, en el capítulo uno se hace un acercamiento de este tipo a los componentes de los servicios web, a las interfaces programables de aplicaciones, a los elementos primordiales de una base de datos y a las herramientas que se emplean en la construcción del portal web con el que interactúa el usuario final.

Después se hace el análisis de el por qué el uso de una interfaz programable de aplicaciones API (Application Programming Interface) tiene ventajas enfocadas hacia el futuro en comparación a las arquitecturas encapsuladas dentro de una misma aplicación.

En los capítulos tres y cuatro se abordan el diseño propuesto, desde las vistas y distribución de la página web, la arquitectura y la base de datos utilizada para almacenar la información, por último, se muestran los aspectos técnicos de cómo se desarrollan cada uno de estos elementos y su integración.



I. MARCO CONCEPTUAL

1.1 MIDDLEWARE

Un middleware es un punto donde se centraliza todas las integraciones evitando que un sistema se conecte directamente con otro sin pasar por este.

Una plataforma de integración como servicio **iPaaS** (Integration Platform-as-a-service), se encuentra alojada en la nube sin la necesidad de la gestión asociada de hardware y costos de infraestructura y, además, puede ser accesible desde cualquier lugar.

1.2 SERVICIOS REST

La transferencia de estado representacional **REST** (Representational State Transfer) es una interfaz para conectar varios sistemas basados en el protocolo de transferencia de hipertexto **HTTP** (Hyper Text Transfer Protocol) y nos sirve para obtener y generar datos en formatos muy específicos, como en notación de objetos de JavaScript **JSON** (JavaScript Object Notation).

Las principales características de esta interfaz por las cuales ha tenido mucha popularidad en los últimos años son las siguientes:

- Utiliza HTTP, un protocolo que se utiliza para las páginas web desde hace muchos años y que ya está consolidado.
- Crea una petición HTTP que contiene toda la información necesaria, es decir, una petición y en base a ella solo espera una respuesta, ósea una respuesta en concreto.

- Se apoya en los métodos básicos de HTTP:

POST: Para crear entidades que no existen en el servidor

GET: Para obtener una entidad o información en concreto

PUT: Para actualizar una entidad completamente

PATCH: Para modificar una entidad de manera parcial

DELETE: Para borrar una entidad, por ejemplo, un registro de la base de datos

- Nos permite separar el cliente del servidor. Esto quiere decir que el servidor se puede desarrollar en NodeJs y Express, y la API REST con MuleSoft.²

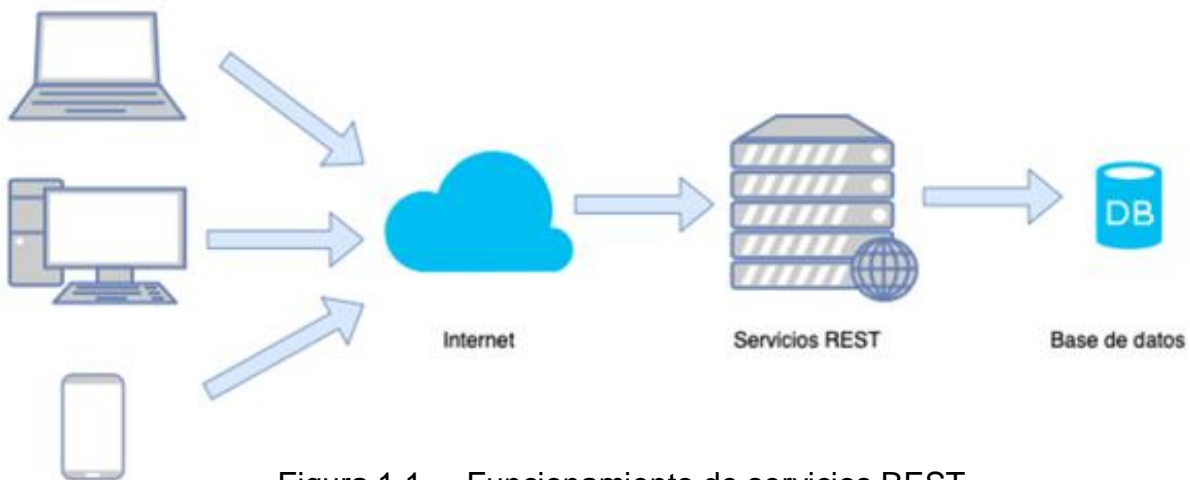


Figura 1.1 Funcionamiento de servicios REST

Los servicios REST permiten acceder y/o modificar información mediante los métodos HTTP, por lo cual se accede a ellos mediante el localizador de recursos uniforme URL (Uniform Resource Locator). Por lo general responden o reciben la información en formato notación de objetos de JavaScript **JSON**, aunque también pueden ser archivos en lenguaje de marcado extensible **XML** (Extensible Markup Language) o en formato de valores separados por comas **CSV** (Comma-Separated Values). Debido a lo sencillo de desarrollar y consumir los servicios REST son muy utilizados actualmente.



Algunos ejemplos de servicios REST los podemos estudiar en la figura 1.2:

Figura 1.2 Ejemplos de servicios REST

Verbo	URI	Descripción
GET	<i>http://localhost:5000/api/Cientes</i>	Obtiene todos los clientes
GET	<i>http://localhost:5000/api/Ciente/1</i>	Obtiene los datos del cliente 1
POST*	<i>http://localhost:5000/api/Ciente</i>	Agrega un nuevo cliente
PUT*	<i>http://localhost:5000/api/Ciente/1</i>	Modifica todos los datos cliente 1
PATCH*	<i>http://localhost:5000/api/Ciente/1</i>	Modifica algunos datos del cliente
DELETE	<i>http://localhost:5000/api/Ciente/1</i>	Borra el cliente 1

**Estos recursos requieren la información que se desea crear o actualizar y se debe enviar en el cuerpo de la petición, la información se puede representar en formatos JSON, CSV, XML etc.*



CÓDIGOS DE ESTADO

Los servicios REST manejan códigos de estatus para indicar a grandes rasgos si una petición es exitosa o fallida y las razones de la falla, si la petición se realizó correctamente, empiezan a partir del código 200 de lo contrario representa un error. Si empiezan con 400 por lo general es porque la información enviada en la petición es errónea (por ejemplo, se envía el ID de un usuario que no existe en el sistema) y si empieza con 500 el error esta del lado del servidor (por ejemplo, un servidor al que se le está dando mantenimiento y aparece suspendido temporalmente).

En la figura 1.3 podemos conocer los códigos de estado exitosos para cada verbo del protocolo HTTP.³

Figura 1.3 Códigos de estado HTTP

Acción	Código de estatus	Descripción
GET	200	OK
POST	201	Creado
PUT	204	Sin Contenido
DELETE	204	Sin Contenido

JSON

Es un formato ligero de representación de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto de lenguaje de programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que el formato JSON sea ideal para el intercambio de datos.

Resumiendo todo lo anterior un servicio REST se compone principalmente de la URL, un verbo HTTP y si el verbo lo requiere, un cuerpo. La petición es interpretada



por el servidor y responderá en base a la información que viaja con la petición. En la figura 1.4 se ejemplifica una petición REST y cada uno de los elementos.

The screenshot displays a REST client interface with the following elements:

- Method:** GET
- URL:** http://localhost:8081/api/alumno/19950227-SERROD-CA17
- Verbo HTTP:** GET
- Response Status:** 200 OK
- Response Time:** 179 ms
- Response Size:** 579 B
- Response Body (JSON):**

```
{
  "Nombre": "Sergio",
  "apellidoPaterno": "Rodriguez",
  "apellidoMaterno": "Calderon",
  "fechaNacimiento": "1995-02-27",
  "direccion": {
    "calle": "Sao Paulo",
    "numeroInterior": "71",
    "tipoUnidad": "Departamento",
    "numeroExterior": "21",
    "ciudad": "Cuautla",
    "estado": "Morelos",
    "colonia": "Empleado"
  },
  "contacto": {
    "telefono": "5524508130",
    "email": "fable2548@gmail.com"
  },
  "carrera": "Antropologia"
}
```

Figura 1.4 Ejemplo de petición REST



1.3 INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API)

Una API nos permite generar comunicación y lógica entre servicios, sin necesidad de saber cómo están implementados cada uno de ellos.

Una API se considera como un contrato, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

También nos permite habilitar el acceso a sus recursos y al mismo tiempo a mantener la seguridad y el control.

Este proyecto está desarrollado con una API para una satisface una sección de los múltiples procesos que llegue a tener una institución académica, el alcance se revisará más adelante.

La API permitirá que un cliente web (portal web, aplicación móvil, herramienta para trabajadores internos, etc.) acceda a información sobre materias, profesores, y/o alumnos.

Existen tres enfoques respecto a las políticas de una API:

Privado

La API solo se puede consumir internamente, así que las organizaciones tienen un mayor control sobre ellas. Esto les da a las organizaciones mayor control sobre sus APIs.

Partners

La API se comparte con otras organizaciones específicas, lo cual puede ofrecer flujos de integración adicionales e implementaciones bilaterales, sin comprometer la calidad. Cómo habilitar el acceso y a quiénes dependerá del desarrollador de la



API. Un ejemplo claro de este tipo de integración lo encontramos en empresas como Uber, Rappi, Waze que contratan el servicio de la plataforma de mapas de Google, en donde se ofrecen rutas, mapas y geolocalización en tiempo real, estas aplicaciones se basan en el uso de la API de Google para dar soporte a su negocio. Claramente Google tiene una tarifa por su servicio y es una relación de negocio, aunque también puede haber casos de asociaciones en donde no necesariamente existen intereses de ganancias económicas, pero sí de colaboración.

Público

En este caso todos tienen acceso a la API, así las organizaciones pueden desarrollar soluciones y se convierte en una fuente de innovaciones. Esto permite que terceros desarrollen aplicaciones que interactúan con su API. Actualmente existen APIs abiertas públicamente con diferentes funcionalidades y tópicos, una muy interesante es la *Open Weather Map* de Rapid API que provee las características del tiempo (temperatura, humedad, pronóstico, probabilidades de tormenta etc.) de un punto geográfico dado, este punto se manda en la petición a la API en forma de coordenadas geográficas (latitud, altitud) o también por el nombre de la ciudad, estado, etc.⁴

El alcance en políticas de este proyecto está previsto para ser privado de una organización académica con sus desarrolladores para exponer los recursos en una aplicación o portal web, o también tener alguna organización gubernamental como socio para que por ejemplo se integre con alguno de sus servicios y que la Clave Única de Registro de Población CURP que provee un estudiante sea validada en las bases de datos del Registro Nacional de Población antes de darlo de alta en el sistema de la institución.

Estas características permiten un uso dinámico de esta API y que consumiendo la información y lógica que accede se pueda integrar con nuevas APIs que implementan otras funcionalidades de la institución de una manera más eficiente.



1.3.1 ELEMENTOS DE LA PLATAFORMA MULESOFT

API DESIGNER

Es una herramienta que permite el diseño, la documentación, y pruebas de una especificación de interfaz. La nomenclatura que se utiliza en MuleSoft para la definición de una API es el lenguaje de modelado de APIs REST **RAML** (REST API Modeling Language) que permitirá la definición de métodos, recursos y parámetros, así como los otros componentes HTTP. También permite el modelado de los cuerpos de entrada y salida de una API y validación de información antes de que esta entre al servicio.⁵

Anypoint Studio

Es la herramienta en donde se implementa la lógica de la API que previamente hemos diseñado, para ello, MuleSoft nos provee de Anypoint Studio, un ambiente de desarrollo integrado **IDE** (Integrated Development Enviroment) que por medio de una interfaz gráfica iremos arrastrando los componentes necesarios para implementar la lógica necesaria.

Anypoint Management Center

Es la herramienta utilizada para el control y monitorización de todas aquellas aplicaciones que hemos implementado en Anypoint Studio. Dentro del conjunto de servicios que ofrece esta herramienta, podemos destacar los dos más importantes.⁵

Runtime Manager: para gestionar el despliegue de la aplicación, existen dos tipos de despliegue:

Despliegue On Premise: La aplicación se despliega en los servidores de la organización.



Despliegue en Cloudhub: La aplicación se despliega en la nube de MuleSoft.

API Manager: es una herramienta que permite realizar toda la gestión de las APIs desplegadas, entre las diferentes funcionalidades que ofrece, destacamos:

- Configuración de un proxy para nuestra API

- Configuración de acuerdos de nivel de servicios **SLA** (Service Level Agreement)

- Configuración de políticas de autenticación y seguridad

- Configuración de alertas basadas en **SLA** o políticas definidas

- Revisión de informes de actividad de uso de la API

En este proyecto se utiliza MuleSoft para la construcción de la API ya que es una herramienta para la orquestación de las comunicaciones entre múltiples sistemas, permite disponer de un punto común donde gestionar todas las integraciones de nuestro ecosistema de aplicaciones sin necesidad de tener conocimientos técnicos avanzados, abstrayéndose en cierta medida, de la tecnología utilizada.



1.4 BASES DE DATOS

Una base de datos es una colección organizada de información usada para modelar algún tipo de proceso organizacional. No importa que se use papel o un programa computacional para recolectar datos. Se considera una base de datos siempre que se haga colección y guardado de información organizadamente para propósitos específicos.

1.4.1 BASES DE DATOS OPERACIONALES

Las bases de datos operacionales son la base de muchas organizaciones e instituciones en la actualidad. Los datos guardados son dinámicos, lo que significa que están en constante cambio y además actualizados en cada momento. Organizaciones como tiendas, empresas de manufactura, hospitales o clínicas usan bases de datos operacionales porque sus datos están en constante cambio.

1.4.2 BASES DE DATOS ANALÍTICAS

En contraste, una base de datos analítica guarda información histórica. Es muy valiosa para determinar tendencias, visualización de estadísticas a través de largos periodos de tiempo, o para hacer proyecciones de negocio. Los datos que se guardan son estáticos, lo que significa que esta información prácticamente nunca se modifica. Laboratorios químicos, empresas geológicas, y firmas de análisis de marketing son ejemplos de organizaciones que usan bases de datos analíticas.

Nótese que la información encontrada en una base de datos analítica es usualmente recogida de una base de datos operacional. Por ejemplo, el historial mensual de ventas debe ser resumido y guardado en una base de datos analítica.



1.4.3 BASES DE DATOS RELACIONALES

De acuerdo con el modelo relacional, los datos en una base de datos relacional son almacenados con relaciones, que se perciben por el usuario como tablas. Cada relación está compuesta de duplas (registros o renglones) y atributos (campos o columnas).

1.4.4 COMPONENTES BASICOS DE UNA BASE DE DATOS

TABLAS

Las tablas son la estructura principal en una base de datos. Cada tabla siempre representa un tema u objeto en específico. Cada tabla contiene al menos una columna (conocida como *llave primaria*) que identifica cada uno de sus renglones.

El tema que se le da a una tabla puede ser desde un objeto hasta un evento. Cuando el tema es un objeto, la tabla representa algo tangible, como una persona, un lugar, o una cosa. Independientemente del tipo cada objeto tiene características que pueden ser almacenadas como datos. Productos, máquinas, edificios, estudiantes o herramientas son ejemplos de objetos que pueden ser representados en una tabla.

Cuando el tema de una tabla es un evento, la tabla representa algo que ocurre en un determinado punto en el tiempo y tiene características que se desean guardar. Estas características pueden ser guardadas y después ser procesadas de la misma manera que una tabla que representa un objeto en específico. Ejemplos de eventos que se pueden representar en una base de datos son: pruebas de laboratorio, encuestas geológicas, distribución de fondos, visitas de pacientes en un hospital etc.



COLUMNAS

Las columnas son la estructura de datos más pequeña en una base de datos, y representa una característica del tema de la tabla a la cual pertenece. Las columnas son las estructuras que guardan datos.

Cada columna en una base de datos correctamente diseñada contiene un y único valor, y su nombre identifica el tipo de valor que contiene. Esto hace que la inserción de datos sea muy intuitiva. Si se pueden observar columnas con nombres como: Nombre, Apellido, Ciudad, Estado se encuentra exactamente qué tipo de valor va en cada columna. También será fácil encontrar datos por Estado o filtrar a las personas que se apelliden Rodríguez.

RENGLONES

Los renglones representan una única instancia del tema de una tabla. Están compuestos por un conjunto de columnas. Por la manera en que la base de datos sea definida, cada renglón es identificado por un único valor en la columna de *llave primaria*.

LLAVES

Las llaves son columnas especiales que juegan un rol importante en una tabla, aunque una tabla puede contener muchos tipos de llaves, solo se hablará de las dos más importantes:

- Llave primaria
- Llave foránea

Las llaves primarias consisten en una columna que identifican cada renglón de manera única en cada tabla. La llave primaria es la más importante por dos razones:

- Su valor identifica un renglón dentro de toda la base de datos
- Las llaves primarias refuerzan la integridad a nivel de tablas y ayudan a establecer relaciones con otras tablas. Cada tabla en la base de datos debe tener una llave primaria.



La matrícula de un alumno es un buen ejemplo de llave primaria por que identifica a cada alumno dentro de la tabla Alumnos (figura 1.5) y asegura la integridad evitando que se dupliquen los renglones o en este caso estudiantes.

Figura 1.5 Tabla “Alumnos”

	MATRICULA	NOMBRE	APELLIDO_PATERO	APELLIDO_MATERNO	CARRERA	FECHA_NACIMIENTO	FEC_ALTA
1	19900212-ARTRIV-GO14	Arturo	Rivera	Gonzalez	Actuaria	1990-02-12	2020-12-19
2	19950227-CESCAL-RO17	Cesar	Calderón	Rodríguez	Antropología	1995-02-27	2020-12-19

Las llaves foráneas se usan cuando se determina que un par de tablas tienen una relación, típicamente se establece esta relación tomando una copia de la llave primaria de la primera tabla e insertándola en la segunda tabla, en dónde se convierte en la llave foránea. El termino foránea se deriva del hecho de que la segunda tabla ya tiene una llave primaria *per se*, y que la llave primaria que se introduce de la primera tabla es foránea en la segunda tabla (Figura 1.6).

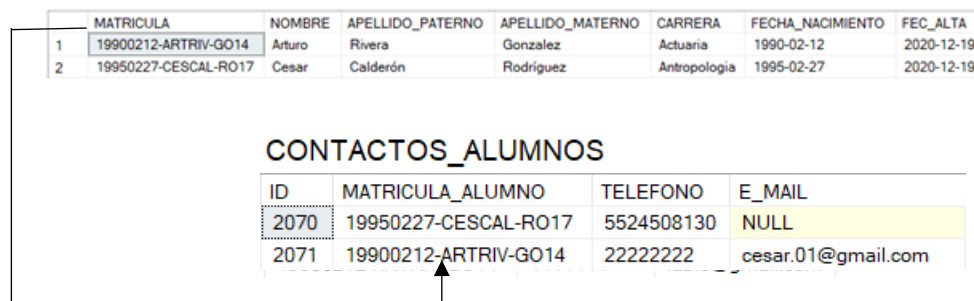


Figura 1. 6 Ejemplo de llave foránea



1.4.5 TIPOS DE RELACIONES EN UNA BASE DE DATOS

Cuando los renglones en una tabla pueden ser asociados de alguna manera con renglones de otra tabla, se dice que estas tablas mantienen una relación. Existen tres tipos de relaciones: uno a uno (one-to-one), uno a muchos (one-to-many).

ONE-TO-ONE

Un par de tablas tienen una relación one-to-one cuando un renglón de la primera tabla está relacionado a un único renglón en la segunda, y cuando el renglón en la segunda tabla está relacionado a un único renglón en la primera tabla. La relación se establece tomando la llave primaria de la primera tabla e insertándola en la segunda tabla, en donde como se explicó previamente se convierte en una llave foránea.

ONE-TO-MANY

Cuando un par de tablas tienen una relación one-to-many, un simple renglón en la primera tabla está relacionado con muchos renglones en la segunda tabla, pero un simple renglón en la segunda tabla estará relacionado únicamente a un renglón de la primera tabla. Esta relación se logra tomando la llave primaria de la primera tabla e insertándola en la segunda convirtiéndose en llave foránea, pero agregando más de un renglón en la segunda tabla que estarán relacionados a únicamente un renglón de la primera tabla.⁶



1.5 INFRAESTRUCTURA DE PORTAL WEB

En este proyecto se hace una representación interactiva de los recursos que nuestra API está exponiendo de una manera entendible para un usuario final (alumno, profesor, personal administrativo) con esta finalidad se emplea un desarrollo de vista front-end con HTML y un entorno en tiempo de ejecución multiplataforma NodeJs para crear el portal web.

1.5.1 HTML

El lenguaje de marcado de hiper texto **HTML** (Hypertext Markup Language) hace referencia a un lenguaje de marcado para la elaboración de páginas web, permite definir estructuras básicas para la visualización de contenido en una página web, como texto, imágenes, videos entre otros.

Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final.

1.5.2 NODEJS

NodeJs es un entorno de ejecución de código abierto y a su vez una herramienta para desarrollar back-end que se ejecuta por medio de código JavaScript, otro lenguaje de programación que después de Python es el más usado en la actualidad.

Tanto JavaScript como NodeJs se ejecutan en el motor de tiempo de ejecución JavaScript V8 (V8 es el nombre del motor de JavaScript que alimenta Google Chrome. Es lo que toma el código JavaScript y lo ejecuta mientras navega con Chrome). Este motor toma el código JavaScript y lo convierte en un código de máquina más rápido. El código de máquina es un código de nivel más bajo que la computadora puede



ejecutar sin necesidad de interpretarlo primero, ignorando la compilación y por lo tanto aumentando su velocidad.⁷

¿Para qué nos sirve NodeJs?

NodeJs utiliza un modelo de entrada y salida sin bloqueo controlado por eventos que lo hace ligero y eficiente (con entrada nos referimos a solicitudes y con salida a respuestas). Puede referirse a cualquier operación, desde leer o escribir archivos de cualquier tipo hasta hacer una solicitud HTTP.

La idea principal de Node.js es usar el modelo de entrada y salida sin bloqueo y controlado por eventos para seguir siendo liviano y eficiente frente a las aplicaciones en tiempo real de uso de datos que se ejecutan en los dispositivos. Es una plataforma que no dominará el mundo del desarrollo web, pero si satisface las necesidades de una gran mayoría de programadores.

La finalidad de NodeJs no tiene su objetivo en operaciones intensivas del procesador, de hecho, usarlo para programación de más peso eliminará casi todas sus ventajas. Donde NodeJs realmente brilla es en la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad.⁸

La escalabilidad es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.



1.5.3 Desarrollo de servidor con Express

Es un framework de nodeJs que nos permite crear aplicaciones web del lado del servidor, cuando se trabaja en una aplicación web se usa HTML, hojas de estilo en cascada **CSS** (Cascading Styles Sheets) y JavaScript que son enviados por el servidor web y también se pueden recibir datos de usuarios, imágenes, videos etc. Al final del lado del servidor se controla la lógica que se va a mostrar al usuario en la página web y para lograr todo esto se necesita el lenguaje de programación como PHP o Ruby, pero a partir del año 2010 se empezó a popularizar el concepto de JavaScript para normalizar el uso de un solo lenguaje de programación del lado del servidor y express es una herramienta que nos puede ayudar a construir una aplicación real utilizando JavaScript puro.

A su vez express es un módulo de terceros, esto quiere decir que en lugar de escribir código puro desde cero en el back-end utilizando JavaScript, también puedes utilizar código JavaScript de otras personas para poder crear el servidor. Este tipo de contribuciones es algo bastante común en la industria del software ya que muchas aplicaciones se complementan de módulos de terceros. Esta característica convierte a express en una herramienta bastante productiva.

Express brinda todas las herramientas para crear el servidor web, esto quiere decir que se complementa con otros módulos de back-end, en el caso de este proyecto se complementa con la comunicación a los recursos de la API que se explicarán más adelante, pero también existen módulos enfocados a conectarse a una base de datos directamente, subir imágenes, videos etc.

II. ANÁLISIS

2.1 Desventajas de los Sistemas Actuales de Servicios Remotos

Desde hace algunos años los sistemas de servicios y portales web siguen una arquitectura descrita en la figura 2.1, en donde notamos que el Portal web esta enganchado a una lógica y funcionalidad directamente, explicado de otra manera es un portal con acoplamiento estrecho a su back-end. Esta arquitectura tiene más desventajas que ventajas enfocadas hacia el futuro, como única ventaja se tiene que es un desarrollo de implementación rápido. Sin embargo, se tienen oportunidades de reutilización de lógica reducida debido al acoplamiento estrecho y esto lo hace también difícil de reutilizar en el momento en que otra entidad quiera acceder a los datos de usuarios que por ejemplo están almacenados en Salesforce.

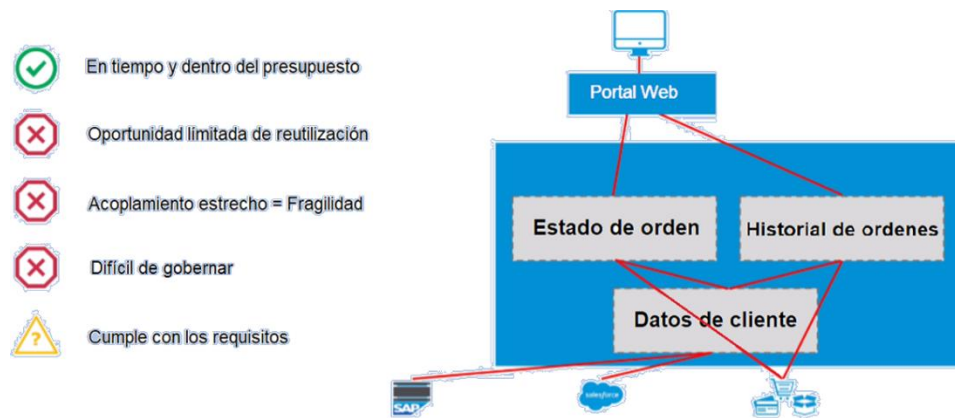


Figura 2.1 Modelo de desarrollo encapsulado

Cuando la organización desee añadir una aplicación móvil para acceder a los servicios desde un teléfono celular, la arquitectura que se tendría que seguir se muestra en la figura 2.2.

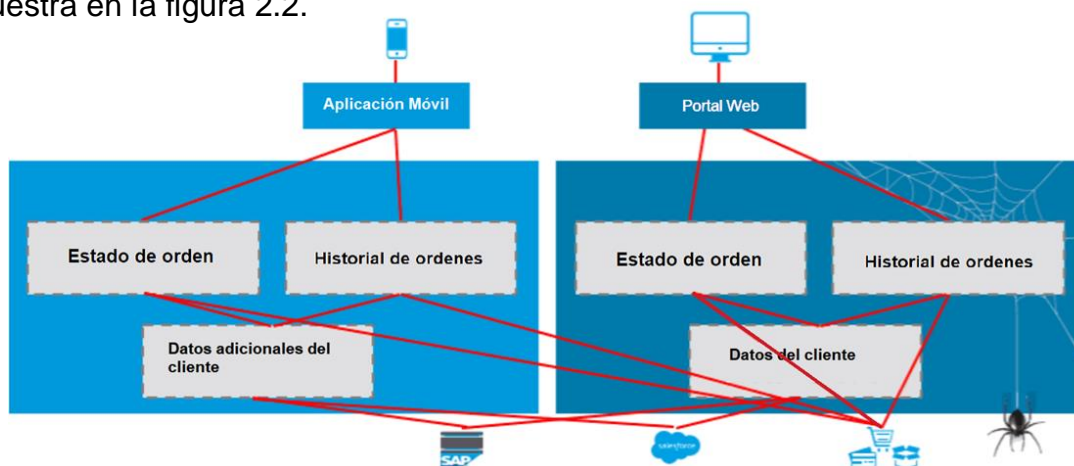


Figura 2.2 Implementación de proyecto para la aplicación móvil

Esto implica desarrollar un sistema totalmente nuevo al que se engancharía la aplicación móvil, esto representa costos de desarrollo nuevamente y tiempo. A su

vez la aplicación Móvil no puede hacer uso de las funcionalidades que el desarrollo del Portal Web implementa, siguiendo este modo de trabajo encontramos que es ineficiente y sin dinamismo.

2.2 Propuesta de solución por medio de una Interfaz de Aplicación Programable

En los últimos años el auge de las APIs es debido a que las organizaciones están buscando un mayor dinamismo y versatilidad en sus sistemas, tanto beneficios de reutilización internos (producir y diseñar para un consumo fácil y reutilizable) como a los expuestos públicamente, que se puedan comunicar y generar lógica con otros sistemas, y que también sus implementaciones puedan ser de utilidad a otras organizaciones por medio de políticas de relación establecidas o a todos los equipos de desarrollo internos. Esta nueva tecnología se basa en la conectividad API-Led (véase figura 2.3) y ayudará a la organización a sostener las demandas sobre departamento de TI (Tecnologías de la Información), en el momento en que las diferentes áreas de la organización busquen generar innovaciones, aplicaciones, servicios simplemente buscando en un catálogo de APIs la lógica, información o comunicación requerida.

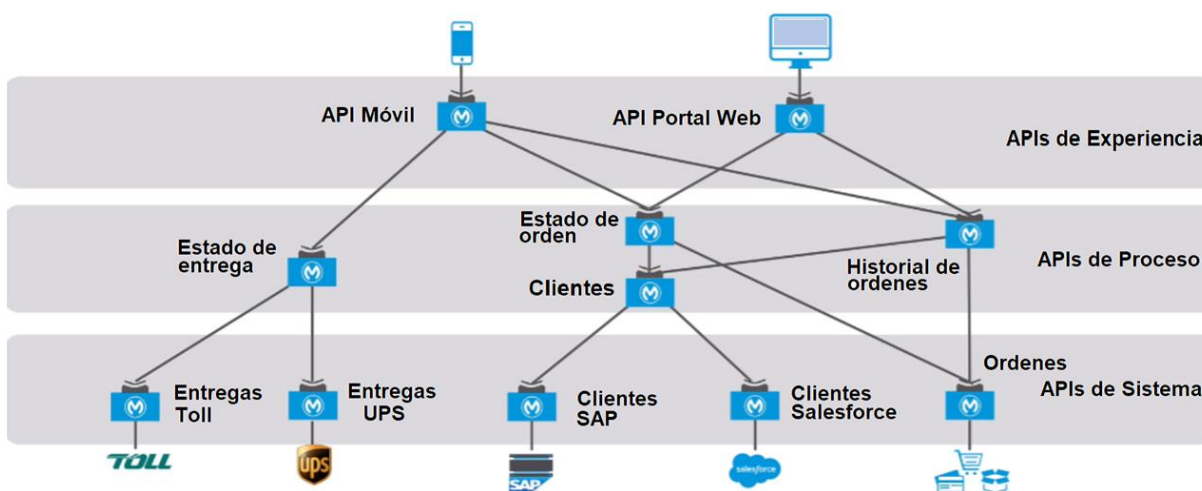


Figura 2.3 Capas del modelo de operaciones API-Led



2.2.1 Capas lógicas de la conectividad API-Led

Sistema:

Básicamente las APIs de Sistema exponen información sensible de una organización. No deben ser expuestas para su uso público.

Proceso:

El objetivo principal de las APIs de Proceso es encapsular lógica de negocio o procesos independientemente de los sistemas fuente (APIs de sistema) desde donde la información se origina. Lógicas de negocio o de proceso pueden ser compartidas con toda la organización.

Experiencia:

En este punto ya se tiene la información de una organización expuesta de una manera privada en las APIs de sistema, y también la orquestación de esta información para generar lógica en las APIs de proceso. Las APIs de experiencia son aquellas que se exponen de una manera pública para su consumo. Son el producto final de la organización. Además, en este nivel es en donde se pueden aplicar políticas de consumo muy personalizadas.⁹

El desarrollo de este proyecto emplea en su back-end una API desarrollada con MuleSoft con el objetivo de aprovechar las bondades que nos brinda este tipo de arquitectura hacia el futuro, ya que siempre se podrá reutilizar el trabajo presente para poder enriquecer sistemas de gestión escolar, además de que es altamente adaptable a las necesidades específicas de políticas de información que a la institución le interese mantener.

III. DISEÑO

3.1 ARQUITECTURA

El diagrama de bloques de alto nivel de este proyecto se puede estudiar con ayuda de la figura 3.1. El flujo de ejecución del sistema es de izquierda a derecha, cuando la API ejecuta alguna operación en la base de datos responde el contenido con el estatus correspondiente a la capa de presentación para ser descrita a los usuarios finales, dando un resultado exitoso o también la propagación de un error debidamente manejado y descrito para ser entendido por el usuario.

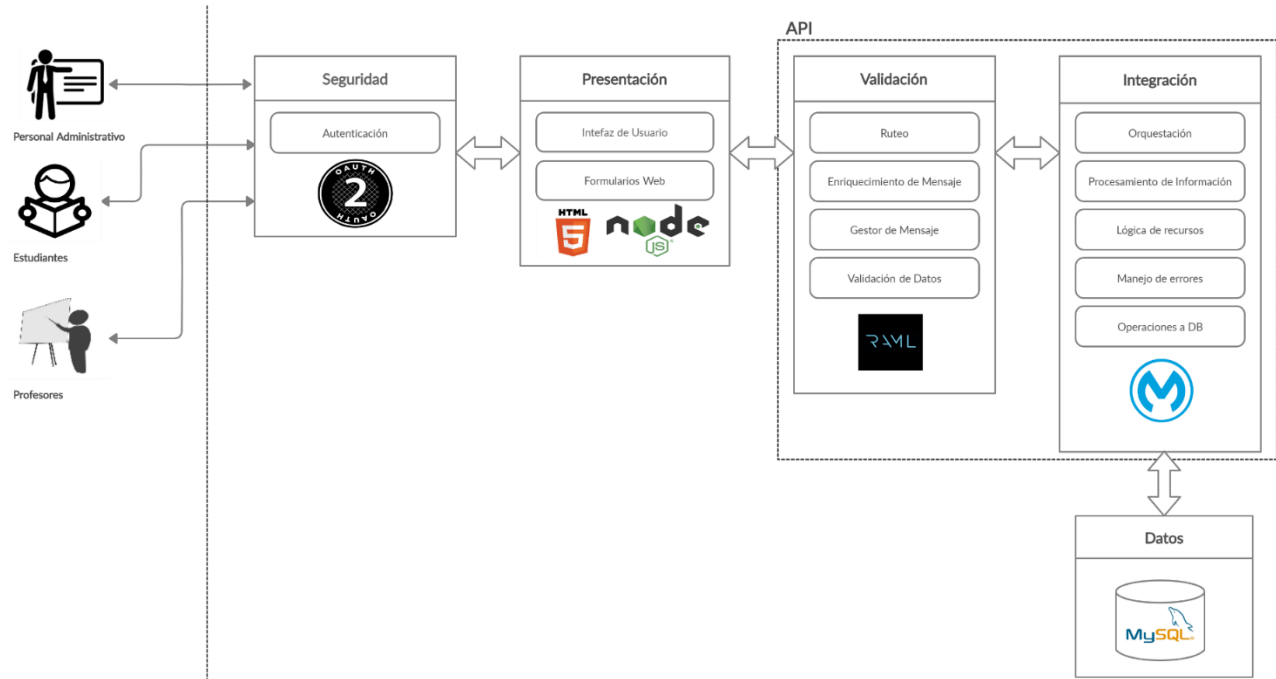


Figura 3.1 Diagrama de bloques del diseño general del proyecto



3.2 OPERATIVIDAD PARA CADA AGENTE

El alcance de las funcionalidades que abarca este proyecto se puede conocer en la figura 3.2, personalizado para cada agente.

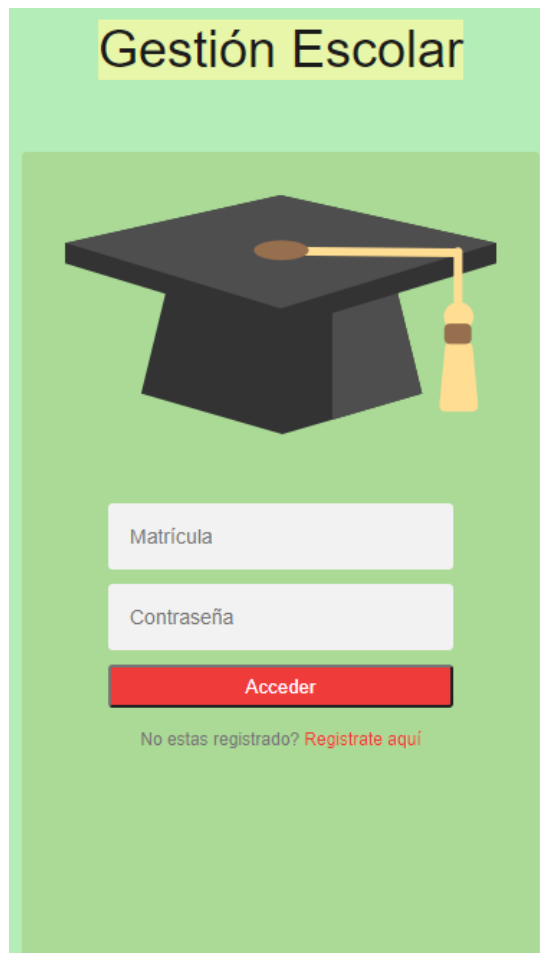
Figura 3.2 Funcionalidad de los agentes

Agentes	Funciones
Personal Administrativo	<ul style="list-style-type: none">• Alta de un profesor• Alta de un alumno• Actualización de un contacto• Actualización de una dirección• Asignar un profesor a un curso
Profesores	<ul style="list-style-type: none">• Obtener lista de cursos disponibles• Consultar información sobre un curso disponible• Inscribir alumno a un curso• Remover alumnos inscritos a un curso• Actualizar el programa de un curso
Estudiantes	<ul style="list-style-type: none">• Obtener lista de cursos disponibles• Consultar información sobre un curso disponible• Inscribirse a un curso

3.3 PROPUESTA DE PORTAL WEB

En esta sección se describe el portal web propuesto, este portal web hace las peticiones a la API por medio de un servidor que se construye con Express de NodeJs. El diseño de este portal web es sencillo de utilizar y también puede ser personalizado en un futuro por la institución académica que lo implemente.

En la figura 3.3 tenemos la vista para ingresar al sistema. Aquí el sistema hace una petición con los datos capturados a la API, la API se encarga de validar estos datos (Matrícula, Contraseña) sean correctos en la base de datos y efectivamente existan, si todo va bien se genera una sesión interna y dejará entrar al usuario al portal web, figura 3.5. Cuando se trata de un alumno que necesita inscribirse, tendrá que registrarse con sus datos a través del enlace *Regístrate aquí* en letras rojas que lo llevarán a la página de Registro, figura 3.4. Cuando el alumno se registra el sistema construye su matrícula en base a la información personal. Finalmente, con la matrícula y contraseña podrá ingresar al portal.



The image shows a login interface for a system titled "Gestión Escolar". It features a green background. At the top, the title "Gestión Escolar" is displayed in a yellow box. Below the title is a large illustration of a black graduation cap with a gold tassel. Underneath the illustration are two white input fields: the first is labeled "Matrícula" and the second is labeled "Contraseña". Below these fields is a red button with the text "Acceder". At the bottom of the form, there is a link that says "No estas registrado? [Regístrate aquí](#)" in red text.

Figura 3. 3 Pantalla de entrada



Registro



Formato: AAAA-MM-DD

Dirección

Contacto

Carrera

Contraseña

Nota: Con la matrícula que te entregue este formulario y tu contraseña podrás ingresar al sistema.

[Regresar](#)

Figura 3. 4 Pantalla para registrarse



Cuando el alumno se ha identificado exitosamente, ingresa al portal en donde verá dos cabeceras principales con diferentes funcionalidades, en la primera que tenemos en la parte superior, encontraremos enlaces a Alumnos, Profesores, Cursos y Ayuda. Cada una de ellas nos llevará a una nueva pantalla y dependiendo el tema (Alumnos, Profesores etc.) habrá otra cabecera en la parte inferior personalizada con las funcionalidades que se asocian a la cabecera superior.

En la figura 3.5, se observa que un alumno puede consultar sus datos por medio de su matrícula simplemente escribiéndola en el formulario y presionando el botón 'Consultar' color rojo.

Alumnos

[Alumnos](#) [Profesores](#) [Cursos](#) [Ayuda](#)

[Inscripción a Curso](#) [Baja de curso](#) [Actualizar Contacto](#) [Actualizar Dirección](#)

Consultar mis datos

19950227-RODSAN-HE17

Consultar

Nombre:	Rodrigo
Apellido Paterno:	Sandoval
Apellido Materno:	Hernandez
Fecha de Nacimiento:	1995-02-27
Carrera:	Actuaria

Contacto

Teléfono:	5524508130
Email:	ejemplo@gmail.com

Dirección

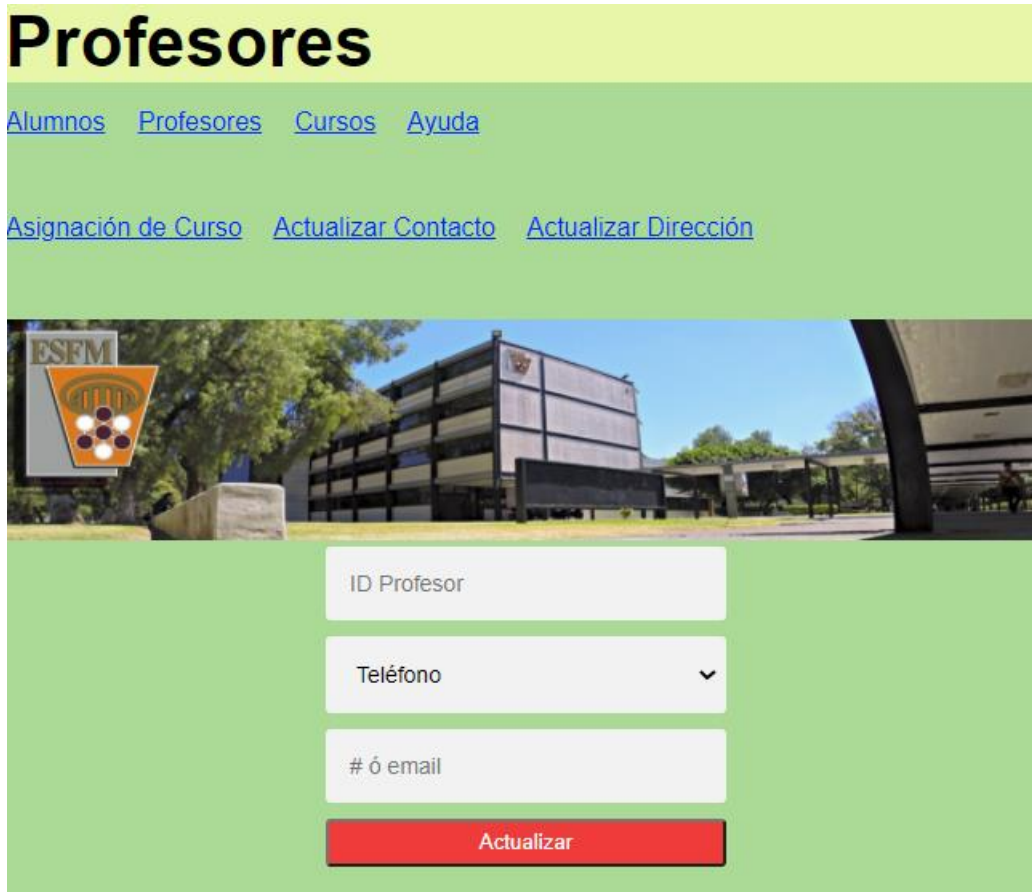
Calle:	Montes de Oca
Numero Interior:	12
Tipo Unidad:	Departamento
Numero Exterior:	12
Ciudad:	Cuautla
Estado:	Morelos
Colonia:	Empleado Municipal

Cursos

BIOL-04

Figura 3.5 Vista para consultar los datos de algún alumno por medio de su matrícula escolar

En la figura 3.6 estamos situados en la sección de *Profesores* y en la subsección *Actualizar Contacto*. Cuando el profesor tiene la necesidad de actualizar su contacto debe proveer su ID, seleccionar el tipo de contacto y escribir el nuevo valor, posteriormente presionar el botón Actualizar para ejecutar la operación.



The screenshot displays the 'Profesores' (Professors) section of the ESFM system. At the top, there is a navigation bar with links for 'Alumnos', 'Profesores', 'Cursos', and 'Ayuda'. Below this, a sub-navigation bar includes 'Asignación de Curso', 'Actualizar Contacto', and 'Actualizar Dirección'. The main content area features a banner image of the ESFM building. Overlaid on this is a form with three input fields: 'ID Profesor', 'Teléfono' (with a dropdown arrow), and '# ó email'. A red 'Actualizar' button is positioned at the bottom of the form.

Figura 3.6 Vista para actualizar el contacto de algún profesor por medio de su ID y el Email o Teléfono

Cuando se presiona el botón Actualizar la vista despliega 'Actualización exitosa' dependiendo si la operación es válida o en caso contrario describe el error.

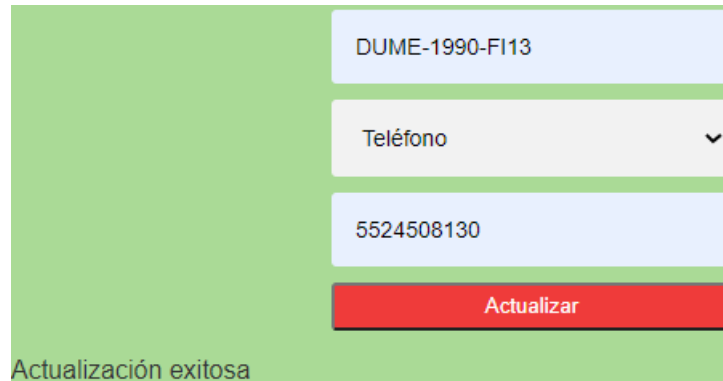


Figura 3.7 Operación exitosa

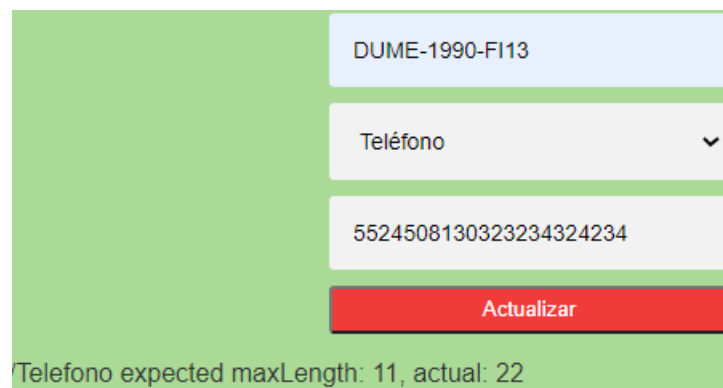


Figura 3.8 Operación fallida (en Teléfono solo se permiten 11 caracteres)

3.4 RECURSOS REST PROPUESTOS Y FUNCIONALIDAD

/cursos

GET: Este método permite obtener la lista de cursos

/cursos/{clave}

GET: Permite obtener información de un curso en específico con la clave del curso

/cursos/{clave}/alumno

PUT: Permite a un alumno inscribirse a un curso con la clave del curso



DELETE: Permite dar de baja a un alumno de un curso con la clave del curso

/profesor

POST: Este recurso permite dar de alta un profesor

/profesor/{numeroEmpleado}

GET: Permite obtener la información de un profesor con su numero de empleado

/profesor/{numeroEmpleado}/dirección

PUT: Permite actualizar la dirección de un profesor con su número de empleado

/profesor/{numeroEmpleado}/contacto

PUT: Permite actualizar el contacto de un profesor con su número de empleado

/profesor/{numeroEmpleado}/asignar/{clave}

PUT: Permite asignar un profesor a un curso con la clave del curso y con su número de empleado

/alumno

POST: Este recurso permite dar de alta un alumno

/alumno/{matricula}

GET: Este recurso permite obtener los datos de un alumno

/alumno/{matricula}/contacto

PUT: Permite actualizar el contacto de un alumno por medio de su matricula

/alumno/{matricula}/direccion

PUT: Permite actualizar la dirección de un alumno con su matricula



3.4.1 HERRAMIENTAS DE SOFTWARE



HTTP

RAML



{ REST }



DataWeave



RegEx



3.5 ESTRUCTURAS DE DATOS

En la figura 3.9 se puede hacer un análisis de las estructuras de datos que utiliza este proyecto. Esta es la información de los usuarios que abarca el proyecto, en donde se pueden hacer generalizaciones o especialidades de datos u objetos, por ejemplo un profesor o un alumno se pueden generalizar como una persona, e inversamente un profesor es una especialidad del grupo que engloba a las personas y que tiene una propiedad exclusiva que lo identifica (*Numero Empleado*), a su vez el objeto persona está relacionado con otros objetos de diferente naturaleza como una dirección y un contacto, esto permite a nivel de desarrollo la reutilización de estructuras de datos que están relacionadas y al mismo tiempo son independientes, en base a esta estructura se desarrolló la base de datos.

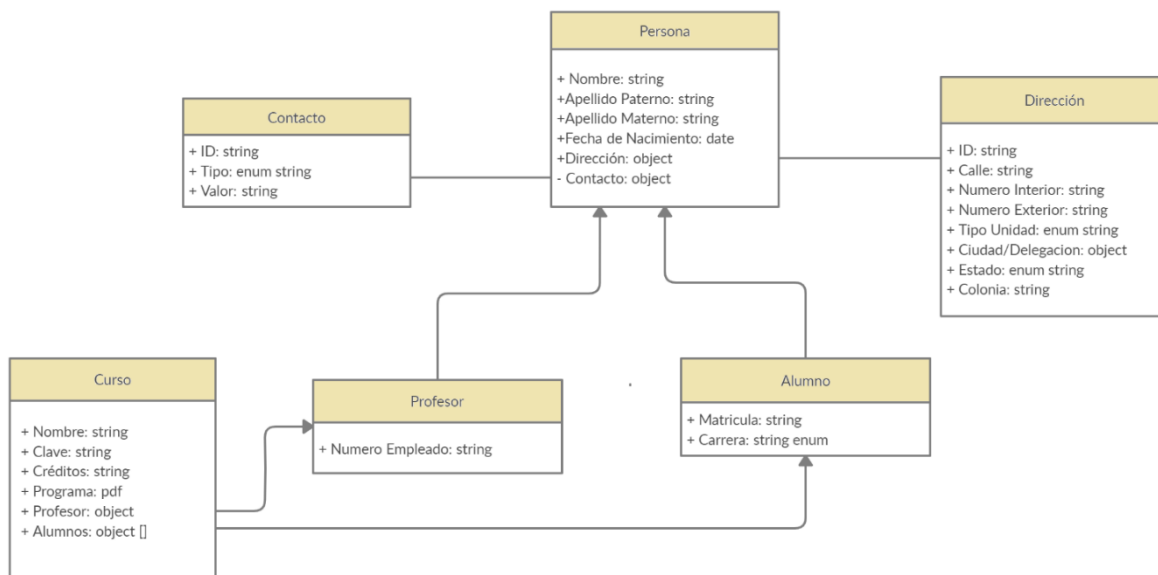


Figura 3.9 Estructura de datos



3.5.1 DICCIONARIO DE DATOS

El objetivo en esta sección es documentar y describir un diccionario de datos en el que se basan las validaciones de entrada/salida de la especificación RAML para cada propiedad de cada objeto. Marcar reglas de sintaxis, contenido y de nomenclaturas ayuda a tener un mejor control de la información que entra/sale de la REST API y evitar por ejemplo guardar información basura. El diccionario de datos se basa en las estructuras de datos descritas en el capítulo anterior. Este diccionario es una propuesta y es la que utiliza el presente proyecto lo que significa que es personalizable en un futuro por una institución académica.

Persona

Atributo	Tipo	Requerido	Nota
Nombre	String	Sí	No se aceptan números. Longitud Máxima: 60
Apellido Paterno	String	Sí	No se aceptan números. Longitud Máxima: 30
Apellido Materno	String	Sí	No se aceptan números. Longitud Máxima: 30
Fecha de Nacimiento	String	Sí	Formato: yyyy-MM-dd
Dirección	Objeto	Sí	
Contacto	Objeto		

Profesor

Atributo	Tipo	Requerido	Nota
Número Empleado	String	Sí	Los números de empleados se forman de la siguiente manera: XXXX-9999-XX99 Donde: X representa una letra 9 representa un dígito

**Alumno**

Atributo	Tipo	Requerido	Nota
Matricula	String	Sí	La matrícula de un alumno esta compuesta de la siguiente manera: 99999999-XXXXXX-XX99 Donde: X representa una letra 9 representa un dígito
Carrera	String	Si	La institución ofrece las siguientes carreras: <ul style="list-style-type: none">• Actuaría• Administración• Agropecuaria• Antropología• Arquitectura• Arquitectura de Paisaje• Arte y Diseño• Artes Visuales

Dirección

Atributo	Tipo	Requerido	Nota
Calle	String	Sí	Longitud mínima de 1 y máxima de 80 caracteres, donde solo se permiten letras y números
Numero Exterior	String	Si	Longitud mínima de 1 y máxima de 30, solo se permiten dígitos y letras
Tipo de Unidad	String	Si	Limitado a los siguientes valores: <ul style="list-style-type: none">• Departamento• Edificio• Casa
Numero interior	String	No	Longitud mínima de 1 y máxima de 30, solo se permiten dígitos y letras
Ciudad	String	No	Longitud mínima de 1 y máxima de 30, solo se permiten dígitos y letras
Estado	String	Si	Limitado a los 32 estados de la república Mexicana
Colonia	String	Si	Longitud máxima de 60 caracteres



Contacto

Atributo	Tipo	Requerido	Nota
Tipo	String	Sí	Limitado a: <ul style="list-style-type: none">• Email• Teléfono
Valor	String	Si	Valor del contacto

Curso

Atributo	Tipo	Requerido	Nota
Clave	String	Sí	Las claves de cursos tendrán la siguiente estructura: XXXX-9999
Nombre	String	Si	Máximo 120 caracteres de longitud, donde solo se permiten letras y números
Créditos	String	Si	Numero de créditos del curso
Profesor	Objeto	Si	Un curso puede tener solo un profesor asignado
Alumnos	Objeto	Si	Un curso puede tener un máximo de 22 alumnos inscritos

3.5.2 DISEÑO DE LA BASE DE DATOS

En la figura 3.10 se puede observar el diseño de la base de datos de este proyecto, cada una de las tablas tiene relación con otra por medio de llaves foráneas con relaciones de renglón uno a uno y uno a muchos. Como ejemplo de una relación uno a uno se tiene entre la tabla **ALUMNOS** y **CONTACTOS_ALUMNOS**, en dónde una matrícula en la columna **MATRICULA** de **ALUMNOS** estará relacionada únicamente con una matrícula de la columna **MATRICULA_ALUMNO** de **CONTACTOS_ALUMNOS**, la columna **MATRICULA_ALUMNO** es llave foránea de la columna **MATRICULA** en **ALUMNOS** y permite asegurar que se asocien únicamente contactos a matrículas existentes en la tabla de **ALUMNOS**, en este caso se tienen almacenados el teléfono y correo electrónico del alumno en el sistema. Dos ejemplos de relación de renglones uno a muchos se puede observar en la tabla **CURSO_ALUMNO** en donde se consigue que un alumno esté inscrito en diferentes cursos gracias a esta relación, esto quiere decir que una matrícula de un alumno en **ALUMNOS** estará presente en repetidas ocasiones en **CURSO_ALUMNO** asociándola a diferentes claves de cursos.

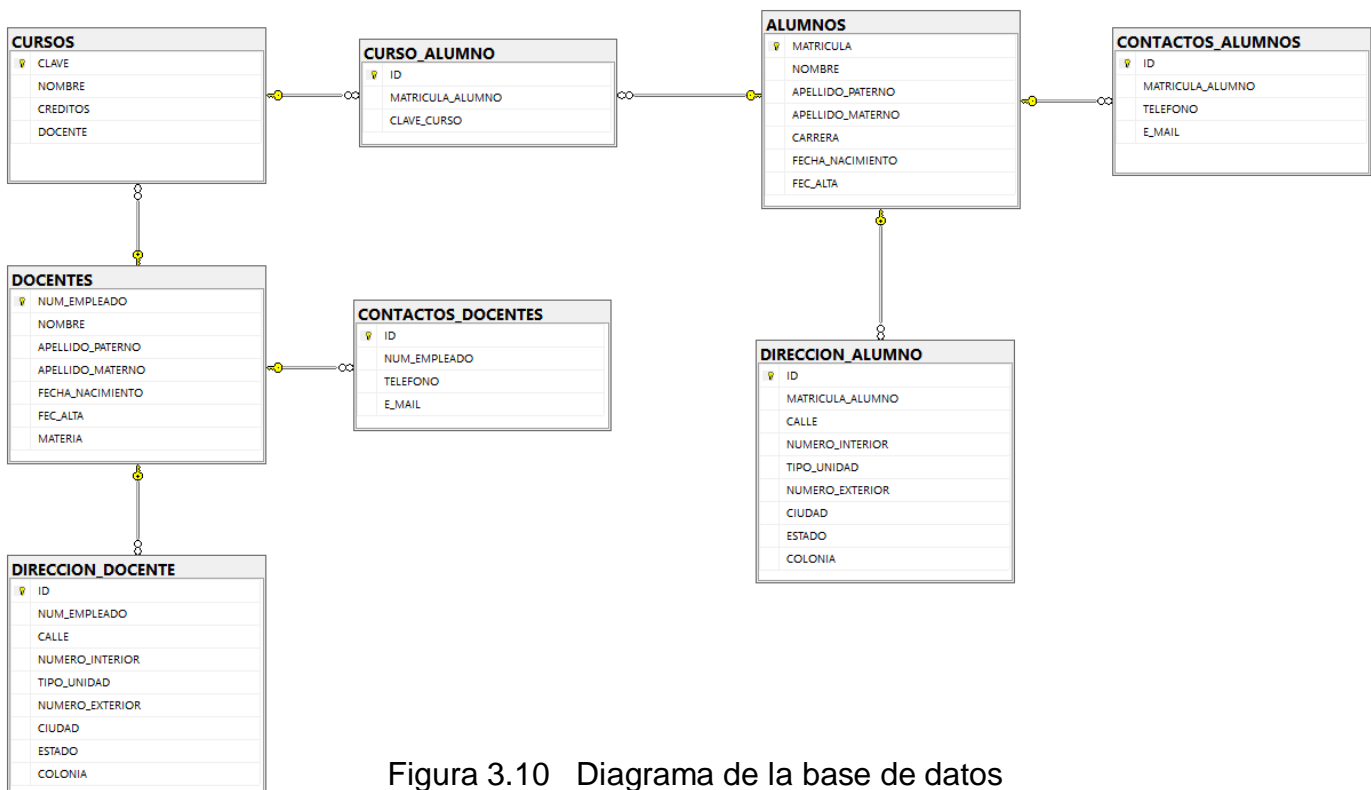


Figura 3.10 Diagrama de la base de datos



IV. IMPLEMENTACIÓN

4.1 ESPECIFICACIÓN RAML

El siguiente fragmento de RAML sirve para ejemplificar la definición de dos recursos que brinda la API. Como se estudió anteriormente lo primero que se define es el nombre del recurso (p. ej. */alumno*) después el verbo HTTP que este recurso ejecuta (POST). En el caso del verbo HTTP POST se ejecuta con la finalidad de crear una entidad nueva en el servidor (en este caso a un alumno), por lo tanto, este recurso espera un cuerpo de entrada para procesarlo y almacenarlo siguiendo la estructura de datos **Alumno**. Finalmente, el código de respuesta HTTP que se espera en caso de que la ejecución POST sea exitosa es un *201*, también devuelve un objeto con la propiedad **matricula** que se construye en la API en base a información del cuerpo de entrada y sirve para generar un vector de información en la base de datos asociada a este valor.

Dentro del recurso */alumno* tenemos un segundo recurso anidado que permite el filtrado de un alumno por medio de un parámetro en la URI llamado **matricula** y que el servidor ocupa para filtrar el contenido específico de esa matricula en la base de datos. Cuando el contenido es obtenido de la base de datos el API construye un cuerpo de respuesta tomando esa información y devolverla al usuario cliente, como respuesta se recibe una estructura de datos tipo **Alumno**.

```
/alumno:
  post:
    is: [errores-comunes]
    description: Este recurso permite dar de alta a un alumno.
    body:
      type: type.Alumno
      example: !include examples/alumno.json
    responses:
      201:
        description: Alta exitosa.
        body:
          type: object
          properties:
            matricula: string
          example:
            { "matricula": "19950227-SERROD-CA17" }
  /{matricula}:
    description: Permite actualizar el contacto de un alumno con su matricula.
    uriParameters:
      matricula:
        description: Identificador de alumno.
        pattern: "[0-9]{8}-[a-zA-Z]{6}-[a-zA-Z]{2}[0-9]{2}"
    get:
      responses:
        200:
          body:
            type: type.Alumno
            example: !include examples/alumno.json
```



Resumiendo, el fragmento anterior es la especificación RAML que el API utiliza para construir los siguientes recursos:

POST /alumno

GET /alumno/{matricula}

El nodo *pattern* contiene una expresión regular **REGEX** (Regular Expression) para validar la sintaxis definida en el diccionario de datos en el campo 'matrícula', si en la petición el campo 'matrícula' no cumple con esta sintaxis, el recurso propaga un código de error **400 Bad Request** (petición no valida).

En la figura (q) tenemos la estructura de datos **Alumno**, la especificación RAML completa disponible en los Anexos o en el repositorio Git están se encuentran todas las estructuras que se emplean como **Profesor**, **Curso**, **Dirección** etc.

Aquí se puede observar la reutilización de estructuras de datos en el tipo **Alumno** que es del tipo **Persona** y también tiene la propiedad **carrera** que lo identifica, la propiedad **carrera** tiene un nodo **enum** que permite restringir los valores que **carrera** acepta y gracias a esto se evita guardar información basura en la base de datos, también se personaliza en base a las carreras que ofrece una institución educativa. A la vez **Persona** contiene las propiedades de **dirección** y **contacto** que ocupan las respectivas estructuras de datos y se pueden revisar con detalle en la especificación RAML.

```
types:
Alumno:
  type: Persona
  properties:
    carrera:
      enum: ["Actuaria","Administracion","Administracion
Agropecuaria","Administracion de Archivos y Gestion
Documental","Antropologia","Arquitectura","Arquitectura de Paisaje","Arte y
Diseno","Artes Visuales"]

Persona:
  properties:
    Nombre:
      maxLength: 30
    apellidoPaterno:
      maxLength: 30
    apellidoMaterno:
      maxLength: 30
    fechaNacimiento: date-only
    direccion: Direccion
    contacto?: Contacto
```

4.1.1 DESARROLLO DE FLUJOS

El desarrollo de la API esta implementado con el IDE de la plataforma MuleSoft, Anypoint Studio 7.7. Es un IDE bastante práctico con una vista gráfica y otra en XML para hacer el desarrollo de flujos internos de la API.

En el flujo principal se tiene configurado un HTTP Listener, a través del puerto 8081 y con una dirección a la máquina local. Este elemento crea el nodo de entrada que espera peticiones por parte de algún cliente consumidor. En la figura 4.1 podemos observar gráficamente el flujo principal (Administracion-main) y el nodo HTTP Listener.

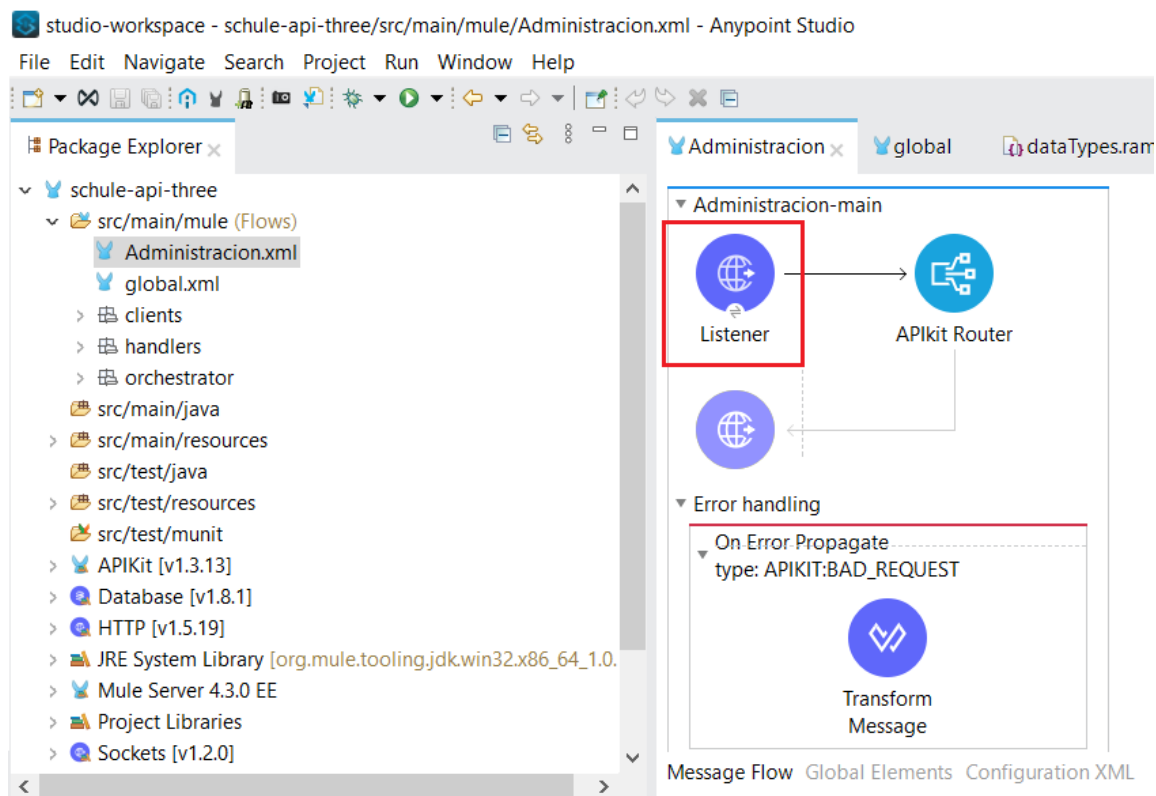


Figura 4.1 Directorio de flujos y flujo principal en dónde se encuentra configurado el recurso HTTP principal y un ruteador APIkit

En la figura 4.2 podemos observar la configuración del nodo de entrada HTTP Listener.

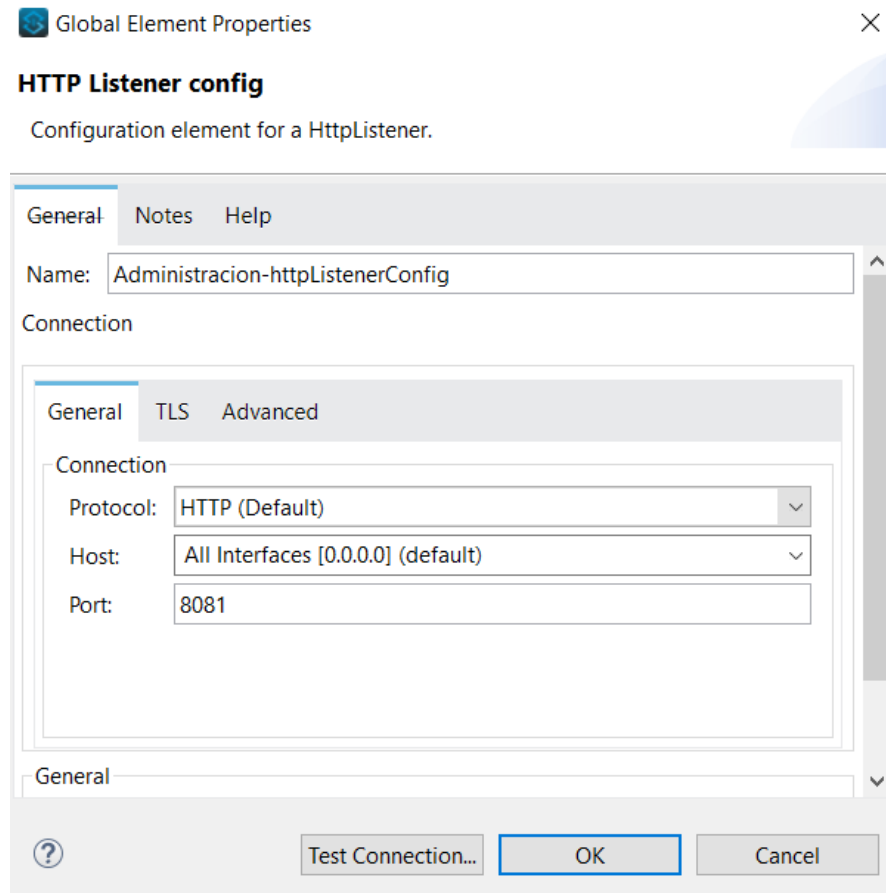
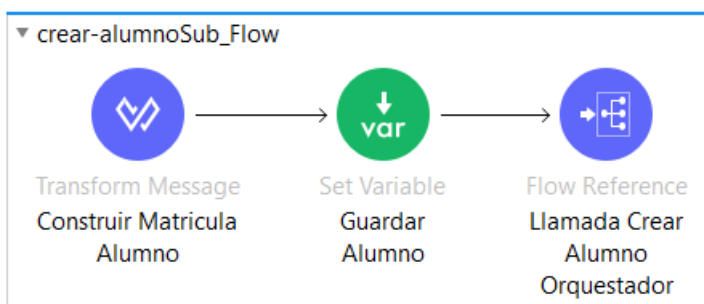
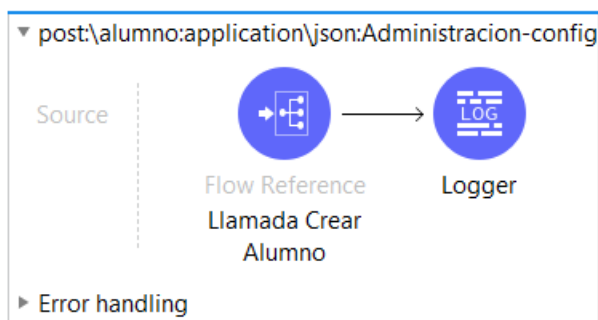


Figura 4. 2 Configuración de HTTP Listener

Después del nodo HTTP Listener en la figura 4.1 podemos observar el elemento APIKit Router, su función es tomar la especificación RAML y generar los flujos correspondientes a cada verbo dentro de los recursos que existen en la especificación. El APIKit Router además de rutear al momento de que recibe un mensaje, hace las validaciones de información de entrada declaradas en la especificación RAML. En caso de que la petición no siga las especificaciones, el APIKit Router propagará un error **400 Bad Request** regresando una descripción de error en un objeto JSON. Si la petición (Verbo HTTP, Cabeceras, Cuerpos, Datos, Dirección) es válida, el APIKit router encauza la ejecución hacia el flujo correspondiente.

En la figura 4.1 observamos una estructura de directorios del lado izquierdo, la ruta **src/main/mule** contiene todos los flujos configurados. Adentro de esta ruta existen otros directorios, **handlers**, **clients**, **orchestrator**. Esta estructura facilita la implementación de las funcionalidades, en un flujo **handler** se hacen guardados de variables o transformaciones comúnmente. En un **orquestador** se hacen peticiones a diferentes clientes para implementar lógica, enriquecer mensajes, etc. y por último un flujo en **clients** se encargará únicamente de hacer una llamada a un servicio externo, por ejemplo, en este proyecto los flujos en **clients** se conectan a la base de datos, estos flujos son llamados clientes porque desde la perspectiva de la base de datos representan un cliente consumidor.

En la figura 4.3 tenemos la secuencia de flujos para crear un alumno, cuando el APIKit Router ha validado la petición, este dirige la petición al flujo *post:\alumno*, posteriormente este flujo hará una llamada al sub-flujo *crear-alumnoSub_Flow*, dentro de este flujo se genera en base a datos del alumno su matrícula y también se guarda el cuerpo con la información del alumno, finalmente el handler hace una llamada al orquestador. El orquestador hace la llamada al flujo que contiene la operación a la base de datos y construye una respuesta que contiene la matrícula y se devolverá al cliente, en este acaso al alumno nuevo. Como último flujo tenemos el cliente de la base de datos, dentro tenemos la operación INSERT que guarda la información del alumno en las tablas correspondientes. La matrícula generada en el handler ayuda a relacionar la información del alumno en las diferentes tablas de la base de datos.



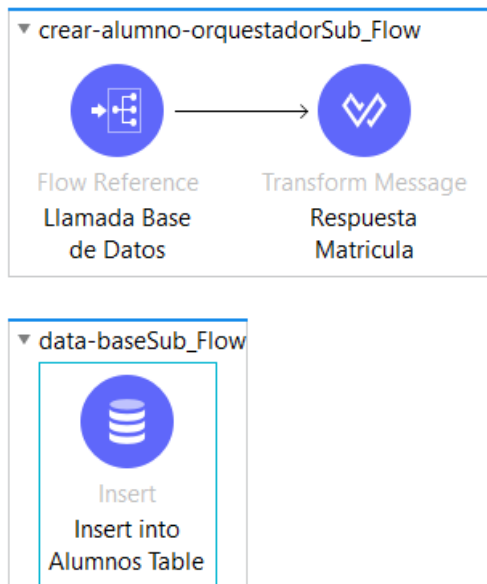
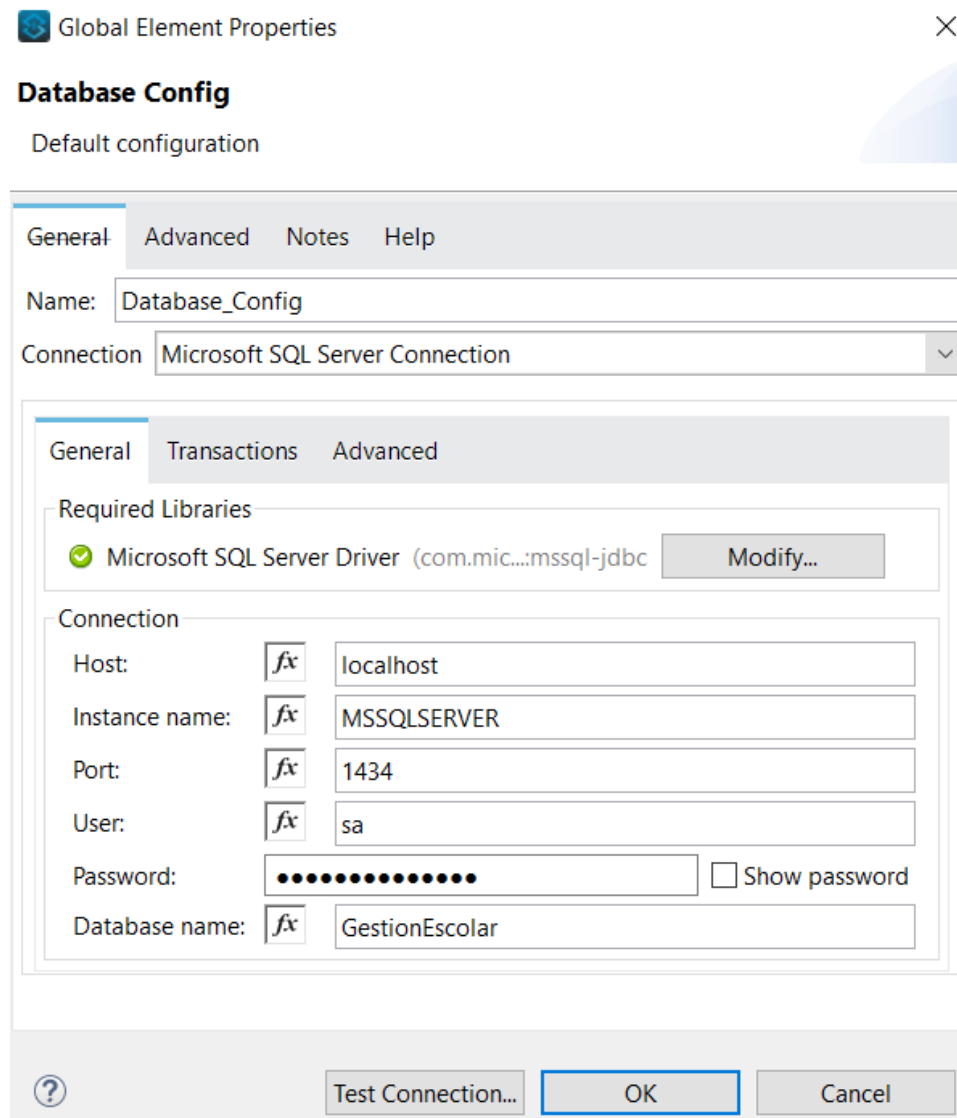


Figura 4.3 Flujos para la creación de un alumno

De esta manera se implementan las otras funcionalidades que contiene la API, cambiando en lógica, guardado de variables, decisiones de ruta etc. La API puede ser descargada en mi repositorio Git remoto o en el disco.

4.1.2 ENLACE CON BASE DE DATOS

La configuración para la conexión a la base de datos se hace por medio de una configuración global dentro de la API para especificar una interfase de acceso a base de datos **JDBC** (Java Database Connectivity). En la figura 4.4 se muestra la configuración para la interacción con la base de datos **GestionEscolar**.



Global Element Properties

Database Config

Default configuration

General Advanced Notes Help

Name: Database_Config

Connection: Microsoft SQL Server Connection

General Transactions Advanced

Required Libraries

- Microsoft SQL Server Driver (com.mic...:mssql-jdbc) Modify...

Connection

Host: localhost

Instance name: MSSQLSERVER

Port: 1434

User: sa

Password: Show password

Database name: GestionEscolar

? Test Connection... OK Cancel

Figura 4.4 Configuración para enlazar la API con la base de datos



La clase del controlador java (**mssql-jdbc**) es una dependencia de maven que se agrega en los Objetos Modelos de Proyecto **POM** (Project Object Model) para ser incorporada al momento de compilar la API (Figura 4.5).

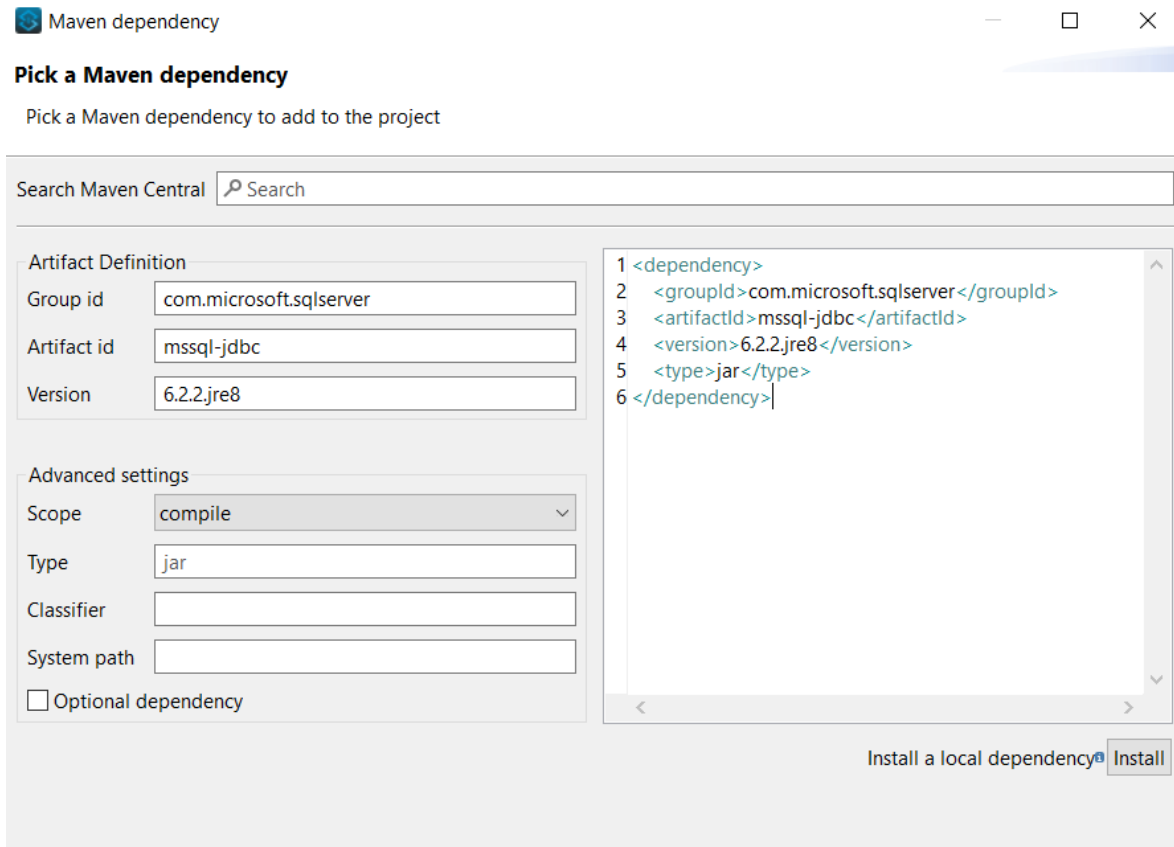


Figura 4. 5 Dependencia de controlador para conectar la base de datos

La configuración **DataBase_Config** definida globalmente (véase figura 4.4) se utiliza en cada una de las operaciones que ejecuta la API en la base de datos, variando solamente el lenguaje estructurado de petición **SQL** (Structure Query Language).

En este proyecto se implementan solamente las siguientes operaciones **SQL**:




- **SELECT**
- **INSERT**
- **UPDATE**
- **DELETE**



Un ejemplo de operación **INSERT** a la base de datos utilizado código **SQL** empleado en este proyecto se muestra en la figura 4.6. Esta operación pertenece al recurso que asocia un alumno a una materia por medio de la matrícula y la clave del curso respectivamente, en donde los datos que se insertan son obtenidos de la petición que hace el usuario cliente y que en este paso ya están validados.

Display Name:


Basic Settings

Connector configuration:   

Query

SQL Query Text:

```
INSERT INTO CURSO_ALUMNO (MATRICULA_ALUMNO,CLAVE_CURSO) VALUES (:matriculaAlumno,:claveCurso);
```

Input Parameters: 

```
1={
2=  matriculaAlumno: vars.matriculaAlumno,
3   claveCurso: vars.claveCurso
4 }
```

Figura 4.6 Petición SQL INSERT a base de datos



4.2 DESARROLLO DE LA BASE DE DATOS

En esta sección se describen un par de sintaxis que se emplean para generar una de las secciones de la base de datos y que también se utiliza para generar todas las secciones y tablas restantes.

Para crear las tablas se usa el lenguaje de definición de datos **DDL** (Data Definition Language) que se encarga también de la modificación de la estructura de los objetos de la base de datos.

El siguiente fragmento **DDL** define la tabla llamada **ALUMNOS** con las columnas **MATRICULA**, **NOMBRE**, **APELLIDO_PATERNO**, **APELLIDO_MATERNO**, teniendo como llave primaria a **MATRICULA** que identifica a cada renglón y alumno. Cada columna tiene una longitud definida de tipo carácter y no pueden tener un valor nulo.

```
CREATE TABLE ALUMNOS(  
    MATRICULA VARCHAR(20) PRIMARY KEY NOT NULL,  
    NOMBRE VARCHAR(60) NOT NULL,  
    APELLIDO_PATERNO VARCHAR(30) NOT NULL,  
    APLLIDO_MATERNO VARCHAR(30) NOT NULL  
);
```

El siguiente fragmento **DDL** define la tabla llamada **CONTACTOS_ALUMNOS**, y se observa cómo generar una llave foránea. La llave foránea esta en la columna **MATRICULA_ALUMNO** de **CONTACTOS_ALUMNOS** que está asociada con la columna de llave primaria **MATRICULA** de **ALUMNOS**.

```
CREATE TABLE CONTACTOS_ALUMNOS(  
    ID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,  
    MATRICULA_ALUMNO VARCHAR(20) NOT NULL,  
    TELEFONO VARCHAR(11) NULL,  
    E_MAIL VARCHAR(30) NULL,  
    CONSTRAINT fk_contacto_alumno FOREIGN KEY (MATRICULA_ALUMNO)  
    REFERENCES ALUMNOS(MATRICULA)  
);
```

Los fragmentos anteriores generan la estructura de datos mostrada en la figura 4.7.



Figura 4. 7 Relación entre tablas con MATRICULA_ALUMNO de CONTACTOS_ALUMNOS como llave foránea de MATRICULA en ALUMNOS



4.3 DESARROLLO DE PORTAL WEB

Como se estudio anteriormente, la herramienta o framework con el que se desarrolla el portal web de este proyecto es NodeJs y el módulo express para configurar y programar el servidor.

En la figura (4) se tiene la configuración mínima para construir una vista en la ruta /profesores. El módulo express primero se requiere, se inicializa y finalmente configuramos una vista con un verbo GET (por defecto un navegador web hace operaciones GET al momento de visitar los sitios web), esta configuración la encontramos en la línea que empieza con app.get(). Como parámetros se escribe el nombre de la ruta y una función con dos parámetros (req,res) que se traducen como *petición* y *respuesta*, adentro de esta función ocupamos una propiedad de res (render) para renderizar un archivo HTML llamado 'profesores' y el navegador web lo represente.

```
const express = require('express')
const app = express()

app.get('/profesores', (req,res) => {
  res.render('profesores', {
    title: 'Profesores',
    name: 'César Rodríguez'
  })
})
```

Después de crear las rutas y renderizar las vistas HTML necesarias se configura el puerto en donde el servidor estará esperando peticiones con otra funcionalidad de express llamada 'listen'.

```
app.listen(3000, () => {
  console.log('Servidor en puerto 3000.')
})
```



MODULO 'REQUEST'

El módulo request es una manera simple de hacer llamadas HTTP con código JavaScript. Este módulo se utiliza para crear las llamadas a los recursos que expone el API, parametrizarlas y configurarlas. Cuando se usa este modulo el portal web se convierte en el cliente de la API. En la figura (2) se muestra un ejemplo del uso de request para llamar al recurso `/cursos` que expone la API. La funcionalidad se engloba en la función `obtenerCursos` y se exporta para poder usarlo fuera de este archivo.

```
const request = require('request')

const obtenerCursos = (curso, callback) => {
  console.log(curso)
  var url = 'http://localhost:8081/api/cursos'// + curso
  if( curso !== '') {
    var url = `http://localhost:8081/api/cursos/${curso}`
  }
  console.log(url)
  request({ url: url, json: true}, (error, {body}) => {
    console.log(body)
    callback(body)
  })
}

module.exports = obtenerCursos
```

En el archivo en donde se configuran las vistas se hace la petición de los módulos que hacen la llamada a la API con la siguiente línea:

```
const obtenerCursos = require('./utils/obtenerCursos.js')
```

Después se crea un recurso que hace la petición a la API con los parámetros necesarios por medio de la función `obtenerCursos`:

```
app.get('/api/cursos', (req,res) => {
  if(req.query.codigoCurso == '')
    return res.send ({
      error: 'Debes proveer un filtro de busqueda'
    })
  // console.log(req)
  obtenerCursos(req.query.codigoCurso, (cursos) => {
    res.send(cursos)
  })
})
```



Finalmente se hacen las peticiones a los recursos dinámicos del servidor por medio de otro modulo llamado *fetch*, se puede observar que *desinscripcionAlumno* es asignado a un botón de las vistas HTML con ayuda de *#Desinscribir*, de igual manera los elementos clave, matricula y message-1, que en este caso son formas HTML. Cuando el usuario presiona el botón relacionado al elemento Desinscribir se dispara la función ***addEventListener*** que hace finalmente la petición al recurso del servidor con la información que se recibe de la vista HTML.

```
const desinscripcionAlumno = document.querySelector('#Desinscribir')
const clave = document.querySelector('#clave')
const matricula = document.querySelector('#matricula')
const mensajeUno = document.querySelector('#message-1')

desinscripcionAlumno.addEventListener ( 'submit', (e) => {
  mensajeUno.textContent = 'Cargando...'
  e.preventDefault()
  const valores = {clave: clave.value, matricula: matricula.value}
  console.log(valores)
  fetch('http://localhost:3000/api/alumno/baja?claveCurso=' + valores.clave + '&' + 'matricula=' + valores.matricula ,{
    method: 'DELETE',
    body: JSON.stringify({ "matricula": valores.matricula }),
    headers: { 'Content-Type': 'application/json' }}
  ).then( (response) => {
    console.log(response),
    mensajeUno.textContent = "Baja exitosa"
  })
})
```




VISTAS HTML

En el siguiente fragmento de código se observa el HTML de la vista para desinscribir a un alumno de un curso. En la línea con la etiqueta script se direcciona al código visto anteriormente en donde se utiliza el módulo *fetch*. Este fragmento HTML es también el que renderiza el módulo express.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Alumnos</title>
  <link rel="icon" href="img/alumno.jpg">
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <div class="main-content">
    {{>header}}
    {{>alumnosHeader}}
    
    <form id="Desinscribir">
      <input id='matricula' name='matricula' placeholder="Matricula">
      <input id='clave' name='clave' placeholder="Código de Curso">
      <button>Dar de baja</button>
    </form>
    <p id="message-1"></p>
  </div>
  {{>footer}}
  <script src="/js/desinscribirAlumno.js"></script>
</body>
</html>
```

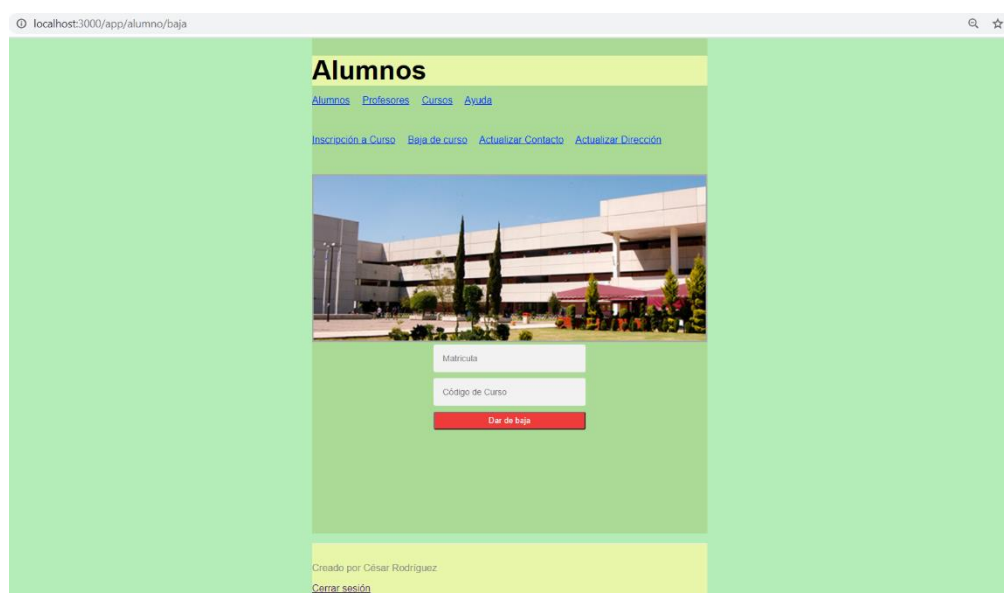


Figura 4. 8 Vista gráfica del fragmento HTML



CONCLUSIONES

El último suceso que ha tenido un impacto en prácticamente todos los sentidos de la vida de las personas se debe a la pandemia que comenzó en diciembre del 2019, COVID-19.

Es por ello por lo que encontré una fuerte motivación y oportunidad para desarrollar un trabajo que permitiera a un segmento de nuestra sociedad poder operar de manera virtual, como lo mencioné al principio, la demanda que las organizaciones están teniendo hacia la digitalización es a nivel global, y es el punto más alto en la historia de la humanidad, nunca se había tenido tanta necesidad de que nos tuviéramos que organizar de esta manera lo más posible en todos los ámbitos sociales.

Este trabajo busca dar un paso en este sentido con la implementación de una API que como vimos es reutilizable, adaptable, descubrible, y accesible. Estas características le dan la capacidad de trabar batalla contra las altas demandas de digitalización en la que vivimos.

Se logró aportar funcionalidad a los diferentes agentes de una institución académica para darse de alta, actualizar información, dar de alta o baja a alumnos de ciertas materias y a los profesores de impartir materias, consultar información de materias, seguridad de autenticación siendo una parte muy importante ya que se expone información sensible cómo direcciones y contactos de alumnos y profesores, todo esto desde una página web que utiliza una interfaz de programación de aplicaciones y una base de datos.

Este diseño, código y método lo he puesto accesible a cualquiera en mi repositorio Git, aquí se puede descargar de manera virtual también, el trabajo escrito, la presentación, el artefacto de la API, el código JavaScript y nodeJs, el diseño del portal web, el código para construir la estructura de la base de datos y en base a ello reutilizar, acoplar y adaptar este trabajo para complementar aplicaciones, para dar base a algún proyecto que apenas comienza o para dar una guía metodológica de implementación.



CIBERGRAFÍA Y BIBLIOGRAFIA

- [1] <https://blog.gft.com/es/2020/04/28/cuales-son-los-aspectos-clave-de-la-digitalizacion-en-tiempos-de-pandemia-y-post-pandemia/>
- [2] <https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/>
- [3] <https://abi.gitbook.io/net-core/3.-servicios-rest/3.1-servicios-rest>
- [4] <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>
- [5] <https://www.nts-solutions.com/blog/salesforce-mulesoft-que-es.html>
- [6] Viescas, J., y Hernandez, M. (2018). SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL, 4th Edition. Anatomy of a Relational Database. USA SQL Standards Committee
- [7] <https://openwebinars.net/blog/que-es-nodejs/>
- [8] <https://blogs.mulesoft.com/learn-apis/api-led-connectivity/what-is-api-led-connectivity/>
- [9] <https://www.tokioschool.com/noticias/detalles-escalabilidad-en-redes/>

Repositorio GitHub

<https://github.com/CesarFlick/shule-server-repositorio/tree/session-branch>

<https://github.com/CesarFlick/schule-api-repository>