TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE ZAMORA

Arquitectura de Servicios

U2: Tecnologías para el desarrollo de servicios
Actividad 2.-2do Avance del Proyecto

MIS. Roberto Suárez Zinzun
Especialidad: Desarrollador FullStack

Integrantes:

21010262 Barriga Garibay Leonardo

21010280 García Lira Carlos Humberto

21010292 Quesada López Carlos Emmanuel

21010310 Ramírez Rodríguez Manuel

ISC++
Ing. en sistemas computacionales

Zamora, Mich. Viernes, 14 de marzo de 2025

Índice

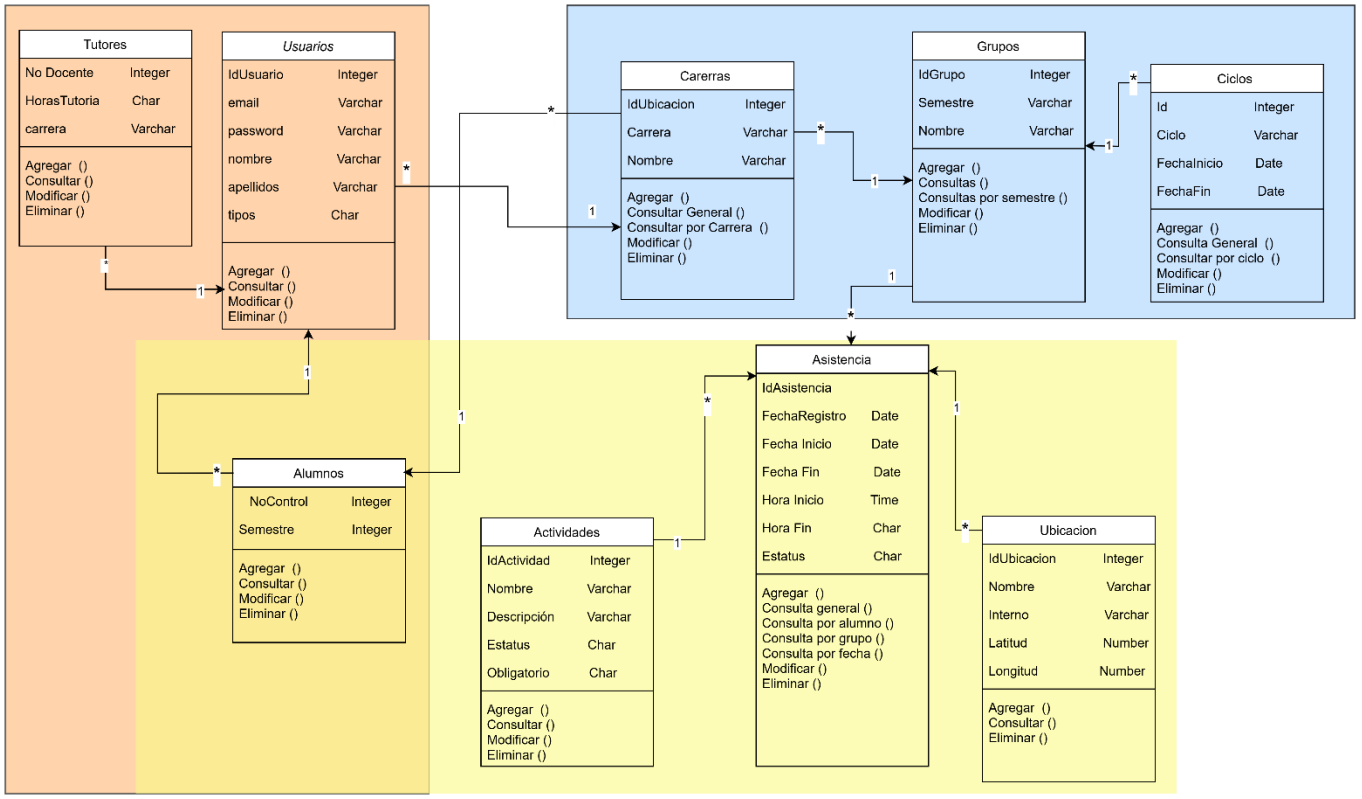
Introducción

El presente trabajo aborda la fase de diseño e implementación inicial de una base de datos para un proyecto de desarrollo de software, cuyo escenario fue previamente definido en la unidad anterior. El objetivo principal es transformar el diagrama de clases conceptual en un modelo de base de datos concreto y funcional. Para ello, se ha seleccionado el tipo de base de datos más adecuado (SQL o NoSQL) según las características del proyecto.

Este documento detalla el proceso de diseño del modelo de datos, la creación del script de la base de datos (o la estructura de colecciones en caso de NoSQL), la carga inicial de datos de prueba y, fundamentalmente, la definición exhaustiva de los requerimientos de los servicios que interactuarán con la base de datos. La definición de servicios incluye no solo la identificación de las operaciones, sino también una descripción detallada de cada una, especificando actores, URLs, métodos HTTP, lógica de negocio, formatos de entrada y salida de datos. Esta especificación rigurosa es crucial para asegurar que los servicios cumplan con las necesidades del proyecto y se integren de manera efectiva con la base de datos. El uso de MongoDB por su flexibilidad como un motor NoSQL.

1. Diseño del modelo de la base de datos

Nuestra propuesta de base de datos será en un motor NoSQL, específicamente en MongoDB, esta elección es debido a las ventajas de flexibilidad que ofrece SQL y que el escenario de nuestro proyecto no requiere de operaciones como lo son las transacciones, además de manejar un gran volumen de datos.
El modelo estará basado en el diagrama de clases



Posteriormente se identificó objetos que pueden ser embebidos en otras colecciones. A continuación, los modelos de los documentos propuestos:

Actividades

```
{
  "_id": "ObjectId()",
  "nombre": "3er Congreso de las Ciencias de la Computacion",
  "descripcion": "Congreso anual de las ciencias de la computacion",
  "estatus": "Por realizar",
  "obligatoria": true
}
```

Asistencias

```
{
  "_id": "ObjectId()",
  "actividad": "ref(actividades.idActividad)",
  "fechaRegistro": "2022-01-01 10:00:00.123Z",
  "fechaInicio": "2022-01-01 10:00:00.123Z",
  "fechaFin": "2022-01-01 10:00:00.123Z",
  "horaInicio": "test",
  "horaFin": "test",
  "estatus": "test",
  "ubicacion": "ref(ubicaciones.idUbicacion)",
  "grupo": "ref(grupos.idGrupo)",
  "listaAsistencia": ["ref(alumnos.noControl)"]
}
```

Carreras

```
{
  "_id": "ObjectId()",
  "carrera": "String",
  "nombre": "String"
}
```

Ciclos

```
{
  "_id": "ObjectId()",
  "ciclo": "ENE-JUN 2025",
  "fechaInicio": "04-02-2025",
  "fechaFin": "13-06-2025"
}
```

Grupos

```
{
  "_id": "ObjectId()",
  "nombre": "String",
  "semestre": "Integer",
  "ciclo": "ref(ciclos.idCiclo)",
  "carrera": "ref(carreras.idCarrera)",
  "tutor": "ref(usuarios.docente.noDocente)",
  "alumnos": ["ref(usuarios.alumno.noControl)"]
}
```

Ubicaciones

```
{
    "_id": "ObjectId()",
    "nombre": "string",
    "interno": "boolean",
    "latitud": "number",
    "longitud": "number"
}
```

Usuarios

```
{
  "_id": "ObjectId()",
  "email": "test@example.com",
  "password": "Hola1234",
  "nombre": "test",
  "apellidos": "test",
  "tipo": "alumno",
  "alumno": {
    "noControl": "21000000",
    "semestre": 1,
    "carrera": "ref(carreras._id)"
  },
  "tutor": {
    "noDocente": "123456",
    "horasTutoria": 10,
    "carrera": "ref(carreras._id)"
  }
}
```

2. Script de la BD

Revisar los archivos .json adjuntos

3. Carga inicial de datos

Revisar los archivos .txt con los comandos para la inserción desde MongoDB Shell

4. Diseño del modelo de la base de datos

Asistencias

1. **Tipo de Servicio:** *Entidad* (Gestiona la información de las asistencias), este debido a que se realizarán operaciones CRUD sobre la entidad.

2. Responsable de la entidad: Carlos Humberto García Lira y Leonardo Barriga

Elemento	*Valor*
Actor(es)	Coordinador y Tutor.
URL	/asistencias
Método HTTP	Post
Lógica de negocio	1. El sistema recibe la solicitud con los datos de la asistencia. 2. Valida que los datos de entrada sean correctos (fechas, IDs, etc.). 3. Verifica si ya existe una asistencia registrada para la misma actividad, fecha y grupo (opcional). 4. Crea un nuevo documento de asistencia en la base de datos. 5. Retorna confirmación de la asistencia.

Entrada	{
	"actividad": "ObjectId()",
	"fechaInicio": "YYYY-MM-DDTHH:mm:ss.sssZ",
	"fechaFin": "YYYY-MM-DDTHH:mm:ss.sssZ",
	"horaInicio": "HH:mm",
	"horaFin": "HH:mm",
	"estatus": "Pendiente/Realizada",
	"ubicacion": "ObjectId()",
	"grupo": "ObjectId()",
	"listaAsistencia": ["numero_control1", "numero_control2", ...]
	}

Salida	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001","21000003"]
	},
	
	O, en caso de error:
	{
	"error": "No se pudo registrar la asistencia."
	}

Garibay

Registrar asistencia

Consulta general de asistencia

Elemento	*Valor*
Actor(es)	Coordinador, Tutor y Alumno
URL	/asistencias
Método HTTP	Get
Lógica de negocio	1. El sistema recibe una solicitud. 2. Obtiene todas las asistencias registradas de la base de datos, se muestran en un formato de lista. 3. Regresa las asistencias.
Entrada	Ninguna, no se necesitan parámetros para obtener la consulta general.

Salida	[
	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001","21000003"]
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "No se encontró ningún dato registrado."
	}

Consulta por alumno

Elemento	***Valor***
Actor(es)	Coordinador, Tutor y Alumno
URL	/asistencias/{noControl}
Método HTTP	Get
Lógica de negocio	1. El sistema recibe la solicitud con el número de control. 2. Busca en la base de datos las asistencias donde el número de control aparezca en listaAsistencia. 3. Si no encuentra coincidencias, retorna un mensaje de error o una lista vacía. 4. Retorna una lista de asistencias donde el alumno asistió.
Entrada	Ninguna, (el número de control se pasa en la URL).

Salida	Mismo formato que la consulta general, pero filtrado por alumno.
	[
	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001","21000003"]
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "Asistencias no encontradas para el alumno"
	}

Consulta por grupo

Elemento	***Valor***
Actor(es)	Coordinador y Tutor
URL	/asistencias/{idGrupo}
Método HTTP	Get
Lógica de negocio	1. El sistema recibe el id del grupo. 2. Busca en la base de datos las asistencias que coincidan con el idGrupo. 3. Retorna la información.
Entrada	*Ninguna (el ID del grupo se pasa en la URL)*

Salida	Mismo formato que la consulta general, pero filtrado por grupo.
	[
	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001","21000003"]
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "Asistencias no encontradas para el grupo"
	}

Consulta por fecha

Elemento	***Valor***
Actor(es)	Coordinador y Tutor
URL	/asistencias/{fecha}
Método HTTP	Get
Lógica de negocio	1. El sistema recibe una solicitud. 2. Recibe la fecha en formato YYYY-MM-DD. 3. Busca asistencias cuya fechaInicio o fechaFin coincidan con la fecha dada. 4. Muestra coincidencias.
Entrada	*Ninguna (la fecha se pasa en la URL)*

Salida	Mismo formato que la consulta general, pero filtrado por fecha.
	[
	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001", "21000003"]
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "Asistencias no encontradas por la fecha"
	}

Modificar asistencia

Elemento	***Valor***
Actor(es)	Coordinador
URL	/asistencias/{idAsistencia}
Método HTTP	Put
Lógica de negocio	1. El sistema recibe la solicitud. 2. Recibe el idAsistencia y los datos a modificar en el cuerpo de la solicitud. 3. Valida la existencia de la asistencia. 4. Actualiza el documento de asistencia en la base de datos.

Entrada	Un objeto JSON con los campos a modificar. No se debe poder modificar id.
	{
	"actividad": "ObjectId()",
	"fechaInicio": "YYYY-MM-DDTHH:mm:ss.sssZ",
	"fechaFin": "YYYY-MM-DDTHH:mm:ss.sssZ",
	"horaInicio": "HH:mm",
	"horaFin": "HH:mm",
	"estatus": "Pendiente/Realizada",
	"ubicacion": "ObjectId()",
	"grupo": "ObjectId()",
	"listaAsistencia": ["numero_control1", "numero_control2", ...]
	}

Salida	[
	{
	"id": "ObjectId()",
	"actividad": "ObjectId()",
	"fechaRegistro": "2025-03-15T10:00:00.123Z",
	"fechaInicio": "2025-03-15T14:00:00.000Z",
	"fechaFin": "2025-03-15T16:00:00.000Z",
	"horaInicio": "14:00",
	"horaFin": "16:00",
	"estatus": "Realizada",
	"ubicacion": "ObjectId(ubicacion2)",
	"grupo": "ObjectId(grupo1)",
	"listaAsistencia": ["21000001","21000003"]
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "Asistencias no encontradas por la fecha"
	}

Eliminar asistencia

Elemento	*Valor*
Actor(es)	Coordinador
URL	/asistencias/{idAsistencia}
Método HTTP	Delete
Lógica de negocio	1. Recibe el idAsistencia en la URL. 2. Verifica que la asistencia exista y que el usuario tenga permisos para eliminarla. 3. Elimina de la base de datos.
Entrada	Ninguna (el ID de la asistencia se pasa en la URL)

Salida	{
	"mensaje": "Asistencia eliminada exitosamente",
	"idAsistencia": "ObjectId()"
	}
	
	O, en caso de error:
	{
	"error": "La asistencia no se pudo eliminar"
	}

Ubicación

1. **Tipo de Servicio:** *Entidad* (Gestiona la información de las ubicaciones), este debido a que se realizarán operaciones CRUD sobre la entidad.
2. **Responsable de la entidad:** Leonardo Barriga Garibay

Registrar ubicación

Elemento	*Valor*
Actor(es)	Coordinador y Tutor.
URL	/ubicaciones
Método HTTP	Post
Lógica de negocio	1. Recibe los datos de la nueva ubicación (nombre, interno, latitud, longitud). 2. Valida que los datos sean correctos (tipos, rangos de latitud/longitud, etc.). 3. Crea una nueva ubicación. (Código 201).

Entrada	{
	"nombre": "string",
	"interno": "boolean",
	"latitud": number,
	"longitud": number
	}

Salida	{
	"id": "ObjectId()",
	"nombre": "string",
	"interno": "boolean",
	"latitud": number,
	"longitud": number
	}
	
	O, en caso de error:
	{
	"error": "No se pudo registrar la ubicación"
	}

Consultar ubicación

Elemento	*Valor*
Actor(es)	Coordinador y Tutor.
URL	/ubicaciones
Método HTTP	Get
Lógica de negocio	1. Recibe la solicitud. 2. Obtiene la ubicación de la base de datos. 3. Retorna información.
Entrada	Ninguna, no se necesitan parámetros para obtener la consulta general.

Salida	[
	{
	"id": "ObjectId()",
	"nombre": "string",
	"interno": "boolean",
	"latitud": number,
	"longitud": number
	},
	...
	]
	
	O, en caso de error:
	{
	"error": "No se encontró ningún dato registrado."
	}

Eliminar ubicación

Elemento	*Valor*
Actor(es)	Coordinador.
URL	/ubicaciones/{idUbicacion}
Método HTTP	Delete
Lógica de negocio	1. Recibe la solicitud y el id de la ubicación a eliminar en la URL. 2. Verifica si existe. 3. Elimina.
Entrada	Ninguna (el ID de la ubicación se pasa en la URL)

Salida	{
	mensaje": "Ubicación eliminada exitosamente"
	}
	
	O, en caso de error:
	{
	"error": "La ubicación no se pudo eliminar"
	}

Grupos

1. **Tipo de Servicio:** *Entidad* (Gestiona la información de los grupos), este debido a que se realizarán operaciones CRUD sobre la entidad.
2. **Responsable de la entidad:** Carlos Humberto García Lira

Registrar grupo

Elemento	***Valor***
Actor(es)	Coordinador
URL	/grupos
Método HTTP	Post
Lógica de negocio	1. Recibe los datos de un nuevo grupo (nombre, semestre, ciclo escolar al que pertenece, tutor asignado y al menos un alumno). 2. Valida que los datos sean correctos (tipos, ciclo, carrera, tutor, etc.). 3. Crea un nuevo grupo. (Código 201).

Entrada	{
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId(ciclos)",
	"carrera": " ObjectId(carreras)",
	"tutor": " ObjectId (usuarios)",
	"alumnos": [" ObjectId (usuarios)"
	}

Salida	{
	"id": "ObjectId()",
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId(ciclos.idCiclo)",
	"carrera": " ObjectId(carreras.idCarrera)",
	"tutor": " ObjectId(usuarios.idUsuario)",
	"alumnos": [" ObjectId(usuarios.idUsuario)"]
	}
	O, en caso de error:
	{
	"error": "No se pudo registrar el grupo"
	}

Consulta general de grupos

Elemento	*Valor*
Actor(es)	Coordinador, Tutor
URL	/grupos
Método HTTP	Get
Lógica de negocio	1. El sistema recibe una solicitud. 2. Obtiene todos los grupos registrados de la base de datos, se muestran en un formato de lista. 3. Regresa los grupos.
Entrada	Ninguna, no se necesitan parámetros para obtener la consulta general.

Salida	[
	{
	"id": "ObjectId()",
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId (ciclos.idCiclo)",
	"carrera": " ObjectId (carreras.idCarrera)",
	"tutor": " ObjectId (usuarios.idUsuario)",
	"alumnos": [" ObjectId (usuarios.idUsuario)"]
	}
	...
	]
	
	O, en caso de error:
	{
	"error": "No se encontró ningún dato registrado."
	}

Consultar grupo por semestre

Elemento	*Valor*
Actor(es)	Coordinador y Tutor
URL	/grupos/{semestre}
Método HTTP	Get
Lógica de negocio	1. El sistema recibe el semestre. 2. Busca en la base de datos las asistencias que coincidan con el semestre. 3. Retorna la información.
Entrada	*NA (el semestre del grupo se pasa en la URL)*

Salida	Mismo formato que la consulta general, pero filtrado por semestre.
	[
	{
	"id": "ObjectId()",
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId (ciclos.idCiclo)",
	"carrera": " ObjectId (carreras.idCarrera)",
	"tutor": " ObjectId (usuarios.idUsuario)",
	"alumnos": [" ObjectId (usuarios.idUsuario)"]
	}
	...
	]
	
	O, en caso de error:
	{
	"error": "Asistencias no encontradas para el grupo"
	}

Modificar grupo

Elemento	*Valor*
Actor(es)	Coordinador
URL	/grupo/{idGrupo}
Método HTTP	Put
Lógica de negocio	1. El sistema recibe la solicitud. 2. Recibe el idGrupo y los datos a modificar en el cuerpo de la solicitud. 3. Valida la existencia del grupo. 4. Actualiza el documento de grupo en la base de datos.

Entrada	Un objeto JSON con los campos a modificar. No se debe poder modificar id.
	{
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId(ciclos)",
	"carrera": " ObjectId(carreras)",
	"tutor": " ObjectId (usuarios)",
	"alumnos": [" ObjectId (usuarios)"
	}

Salida	{
	"id": "ObjectId()",
	"nombre": "String",
	"semestre": "Integer",
	"ciclo": " ObjectId(ciclos.idCiclo)",
	"carrera": " ObjectId(carreras.idCarrera)",
	"tutor": " ObjectId(usuarios.idUsuario)",
	"alumnos": [" ObjectId(usuarios.idUsuario)"]
	}
	
	O, en caso de error:
	{
	"error": "Grupo con id no encontrado"
	}

Eliminar grupo

Elemento	*Valor*
Actor(es)	Coordinador
URL	/grupo/{idGrupo}
Método HTTP	Delete
Lógica de negocio	1. Recibe el idGrupo en la URL. 2. Verifica que el grupo exista y que el usuario tenga permisos para eliminarla. 3. Elimina el grupo de la base de datos.
Entrada	Ninguna (el ID de la asistencia se pasa en la URL)

Salida	{
	"mensaje": "Grupo eliminado exitosamente",
	"idGrupo": "ObjectId()"
	}
	
	O, en caso de error:
	{
	"error": "El grupo no se pudo eliminar"
	}

Actividades

1. **Tipo de Servicio:** Entidad (Gestiona la información de las actividades).
2. **Responsable de la entidad**: Manuel Ramírez Rodríguez

Crear Actividad.

Elemento	*Valor*
Actor(es)	Coordinador.
URL	/actividades
Método HTTP	POST
Lógica de negocio	1. El sistema recibe la solicitud POST a /actividades. 2. Verifica que el usuario autenticado sea un Coordinador. Si no, retorna 403 Forbidden. 3. Valida que el cuerpo de la solicitud contenga los campos obligatorios: nombre, descripcion, estatus, obligatoria, ubicacion. 4. Valida que estatus sea uno de los valores permitidos ("Por realizar", "Realizada", "Cancelada"). 5. Valida que obligatoria sea un valor booleano. 6. Verifica que la ubicacion (ID) exista en la colección de ubicaciones. Si no, retorna 404 Not Found. 7. Crea un nuevo documento en la colección actividades con los datos proporcionados. El campo tutor se inicializa como null. 8. Retorna el objeto de la actividad creada, incluyendo el ID generado, con un código de estado 201 Created.

Entrada	{
	"nombre": "Nombre de la Actividad",
	"descripcion": "Descripción de la actividad",
	"estatus": "Por realizar",
	"obligatoria": true
	}

Salida	Éxito (201 Created): {
	"id": "ObjectId(...)",
	"nombre": "Nombre de la Actividad",
	"descripcion": "Descripción de la actividad",
	"estatus": "Por realizar",
	"obligatoria": true
	}
	Error (403 Forbidden): json
	{
	"error": "Acceso no autorizado"
	} Error (400 Bad Request): json {
	"error": "Mensaje detallado del error de validación"
	}

Consultar Actividad.

Elemento	*Valor*
Actor(es)	Coordinador, Tutor y Alumno.
URL	actividades/{idActividad}
Método HTTP	GET
Lógica de negocio	1. El sistema recibe la solicitud GET a /actividades/{idActividad}. 2. Verifica que el id proporcionado sea un ObjectId válido. Si no, retorna 400 Bad Request. 3. Busca la actividad con el id dado en la colección "actividades". 4. Si la encuentra, retorna la actividad (200 OK). 5. Si no la encuentra, retorna 404 Not Found.
Entrada	Ninguna (el ID va en la URL)

Salida	Éxito (200 OK):
	{
	"id": "ObjectId(...)",
	"nombre": "...",
	"descripcion": "...",
	"estatus": "...",
	"obligatoria": ...
	}
	Error (404 Not Found): json {
	"error": "Actividad no encontrada"
	}
	Error (400 Bad Request): json {
	"error": "Id Invalido"
	}

Actualizar actividad.

Elemento	***Valor***
Actor(es)	Coordinador.
URL	/asistencias/{idActividad}
Método HTTP	PUT
Lógica de negocio	1. Recibe PUT a /actividades/{idActividad}. 2. Verifica rol (403). 3. Verifica id. 4. Valida datos. 5. Busca actividad (404). 6. Actualiza. 7. Retorna actualizada (200).

Entrada	{
	"nombre": "Nuevo Nombre",
	"descripcion": "Nueva Descripción",
	"estatus": "En curso",
	"obligatoria": false
	}

Salida	Éxito (200):
	{
	"id": "ObjectId(...)",
	"nombre": "Nuevo Nombre",
	"descripcion": "Nueva Descripción",
	"estatus": "En curso",
	"obligatoria": false
	}
	Error (403/404/400):
	{
	"error": "Mensaje de error"
	}

Eliminar Actividad.

Elemento	***Valor***
Actor(es)	Coordinador.
URL	/actividades/{idActividad}
Método HTTP	DELETE
Lógica de negocio	1. Recibe DELETE a /actividades/{idActividad} 2. Verifica rol (403). 3. Verifica id. 4. Busca (404). 5. Verifica asistencias (409). 6. Elimina. 7. Retorna éxito (200).
Entrada	Ninguna, no se requiere ninguna especie de parámetros.

Salida	Éxito (200):
	{
	"mensaje": "Actividad eliminada"
	}
	Error (403/404/409/400):
	{
	"error": "Mensaje de error"
	}

Asignar Tutor por Actividad.

Elemento	***Valor***
Actor(es)	Coordinador.
URL	actividades/{idActividad}
Método HTTP	PUT
Lógica de negocio	1. Recibe PUT. 2. Verifica rol (403). 3. Verifica id y idTutor. 4. Verifica existencia (404). 5. Actualiza actividad. 6. Retorna actualizada (200).
Entrada	{“idTutor": "ObjetcId()" }

Salida	Éxito (200):
	{
	"id": "ObjectId(...)",
	"nombre": "Nombre Actividad",
	"descripcion": "Descripción",
	"estatus": "Algun Estado",
	"obligatoria": true,
	"idTutor": "ObjectId(...)"
	}
	Error (404/400/403):
	{
	"error": "Mensaje de error"
	}

Usuarios

1. **Tipo de Servicio:** Entidad (Gestiona la información de los usuarios en un contexto general).
2. **Responsable de la entidad:** Manuel Ramírez Rodríguez

Registrar Usuario.

Elemento	*Valor*
Actor(es)	Coordinador, Tutor, Alumno.
URL	/usuarios
Método HTTP	POST
Lógica de negocio	1. Recibe solicitud POST a /usuarios. 2. Verifica que el usuario autenticado tenga permisos para crear usuarios (Coordinador). Si no, retorna 403 *Forbidden (prohibido)*. 3. Valida el cuerpo de la solicitud: - Correo: único y con formato válido. - Contraseña: criterios de seguridad (longitud, etc.). *La contraseña se debe hashear antes de guardarla*. - Tipo: "alumno" o "tutor". - Si tipo es "alumno", valida: noControl, semestre, carrera (que exista). - Si tipo es "tutor", valida: noDocente, horasTutoria, carrera. 4. Crea el documento en la colección "usuarios". 5. Retorna el usuario creado (con ID), código 201.

Entrada	Ejemplo para un alumno:
	{
	"email": "nuevo@example.com",
	"password": "PasswordSegura123",
	"nombre": "Nuevo",
	"apellidos": "Usuario",
	"tipo": "alumno",
	"alumno": {
	"noControl": "21000004",
	"semestre": 2,
	"carrera": "ObjectId(...)"
	}
	}

Salida	Éxito (201):
	(Similar a la entrada, pero con el id generado)
	{
	"id": "ObjectId(...)",
	"email": "nuevo@example.com",
	"password": "<HASHED_PASSWORD>",
	"nombre": "Nuevo",
	"apellidos": "Usuario",
	"tipo": "alumno",
	"alumno": {
	"noControl": "21000004",
	"semestre": 2,
	"carrera": "ObjectId(...)"
	}
	} Error (403/400):
	{
	"error": "Mensaje de error"
	}

Consultar Usuario.

Elemento	***Valor***
Actor(es)	Coordinador, Tutor, Alumno.
URL	/usuarios /{idUsuario}
Método HTTP	GET
Lógica de negocio	1. Recibe GET a /usuarios/{idUsuario}. 2. Verifica id (ObjectId válido). 3. Busca usuario por id. 4. Si existe, retorna usuario (200). 5. Si no existe, retorna 404. 6. *Nota:* Se debe verificar permisos. Un usuario normal solo debe poder ver *su propia* información.
Entrada	Ninguna, sin parámetros extraordinarios.

Salida	Éxito (200): {
	"id": "ObjectId(...)",
	"email": "...",
	"nombre": "...", ...
	}
	Error (404/400): {
	"error": "Mensaje de error"
	}

Consultar General.

Elemento	***Valor***
Actor(es)	Coordinador, Tutor.
URL	/usuarios
Método HTTP	GET
Lógica de negocio	1. Recibe GET a /usuarios. 2. Verificar permisos (solo coordinador/tutor) (403). 3. Retorna lista de todos los usuarios.
Entrada	Ninguna, sin parámetros extraordinarios.

Salida	Éxito (200): [
	{
	"id": "...",
	"email": "...", ...
	},
	{
	"id": "...",
	"email": "...", ...
	}, ...
	]
	Error (403): {
	"error": "Acceso denegado"
	}

Actualizar Usuario.

Elemento	*Valor*
Actor(es)	Coordinador, Tutor, Alumno.
URL	/usuarios /{idUsuario}
Método HTTP	PUT
Lógica de negocio	1. Recibe PUT a /usuarios/{idUsuario}. 2. Verifica id (ObjectId válido). 3. Verifica permisos: - Coordinador puede modificar cualquier usuario. - Usuario normal solo puede modificar *su propia* información. (403 si no tiene permisos). 4. Valida datos del cuerpo (JSON). - *Cuidado con la contraseña*: Si se actualiza, *hashearla*. - No se debe permitir cambiar el tipo de usuario directamente. 5. Busca usuario. Si no existe, 404. 6. Actualiza documento. 7. Retorna usuario actualizado (200).

Entrada	Ejemplo: {
	"nombre": "Nuevo Nombre",
	"apellidos": "Nuevos Apellidos",
	"email": "nuevo_correo@example.com"
	}
	Nota: Los campos a actualizar pueden variar.

Salida	Éxito (200): {
	"id": "ObjectId(...)",
	"email": "nuevo_correo@example.com",
	"nombre": "Nuevo Nombre", ...
	}
	Error (403/404/400): {
	"error": "Mensaje de error"
	}

Eliminar Usuario.

Elemento	***Valor***
Actor(es)	Coordinador.
URL	/usuarios /{idUsuario}
Método HTTP	DELETE
Lógica de negocio	1. Recibe DELETE a /usuarios/{idUsuario}. 2. Verifica rol Coordinador (403 si no). 3. Verifica id (ObjectId válido). 4. Busca usuario. Si no existe, 404. 5. Importante: Verifica dependencias. Ej: ¿Tiene asistencias registradas? ¿Es tutor de un grupo? Si tiene dependencias, se podría: - Retornar error 409 (Conflict). - Eliminar en cascada (con mucho cuidado). - "Desactivar" el usuario en lugar de eliminarlo (recomendado). 6. Elimina/Desactiva usuario. 7. Retorna mensaje de éxito (200).
Entrada	Ninguna, sin parámetros extraordinarios.

Salida	Éxito (200): {
	"mensaje": "Usuario eliminado/desactivado"
	} Error (403/404/409/400): {
	"error": "Mensaje de error"
	}

Iniciar Sesión.

Elemento	***Valor***
Actor(es)	Coordinador, Tutor, Alumno.
URL	actividades/login
Método HTTP	POST
Lógica de negocio	1. Recibe POST a /usuarios/login. 2. Recibe email y password (texto plano) en el cuerpo. 3. Busca usuario por email. Si no existe, 401 (Unauthorized). 4. Si existe, *compara la contraseña proporcionada (hasheada) con la contraseña hasheada almacenada*. Si no coinciden, 401. 5. Si coinciden, autentica al usuario (genera token JWT, por ejemplo). 6. Retorna token (200).

Entrada	{
	"email": "usuario@example.com",
	"password": "Password123"
	}

Salida	Éxito (200): {
	"token": "eyJhbGciOiJIUz..."
	} Error (401): {
	"error": "Credenciales inválidas"
	}

Ciclos

1. **Tipo de Servicio:** Entidad (Gestiona la información de los ciclos escolares), este debido a que se realizarán operaciones CRUD sobre la entidad.
2. **Responsable de la entidad**:Lopez Quesada Carlos Emmanuel

Registrar Ciclos

Elemento	*Valor*
Actor(es)	Coordinador
URL	/ciclos
Método HTTP	Post
Lógica de negocio	1. El sistema recibe la solicitud con los datos del nuevo ciclo escolar. 2. Valida que los datos de entrada sean correctos: 3. Crea un nuevo documento de ciclo en la base de datos. 4. Retorna una confirmación con el ID del ciclo creado

Entrada	{
	"ciclo": "ENE-JUN 2025",
	"fechaInicio": "2025-01-06",
	"fechaFin": "2025-06-13"
	}

Salida	{
	"id": "ObjectId()
	"ciclo": "ENE-JUN 2025",
	"fechaInicio": "2025-01-06",
	"fechaFin": "2025-06-13"
	}
	
	O en caso de error :
	{
	"error": “No se pudo registrar el ciclo “
	}

Consulta general Ciclos

Elemento	*Valor*
Actor(es)	Coordinador, Tutor y Alumno
URL	/ciclos
Método HTTP	Get
Lógica de negocio	1. Recibir solicitud. 2. Obtener todos los ciclos. 3. Retornar 200 con la lista de ciclos.
Entrada	Ninguna, no se necesitan parámetros para obtener la consulta general.

Salida	[
	{
	"id": "ObjectId() ",
	"ciclo": "ENE-JUN 2025",
	"fechaInicio": "2025-01-06",
	"fechaFin": "2025-06-13"
	},
	{
	"id": "ObjectId() ",
	"ciclo": "AGO-DIC 2024",
	"fechaInicio": "2024-08-05",
	"fechaFin": "2024-12-13"
	}
	]
	O, en caso de error:
	{
	"error": "No se encontraron ciclos registrados ."
	}

Consulta por Ciclo

Elemento	*Valor*
Actor(es)	Coordinador, Tutor y Alumno
URL	/ciclos/{id}
Método HTTP	Get
Lógica de negocio	1. Recibir solicitud con id. 2. Buscar ciclo por id. 3. Si existe, retornar 200 y el ciclo; si no, 404 .
Entrada	Ninguna, (el id se pasa en la URL).

Salida	Mismo formato que la consulta general, pero filtrado por ciclo.
	
	{
	"id": "ObjectId() ",
	"ciclo": "ENE-JUN 2025",
	"fechaInicio": "2025-01-06",
	"fechaFin": "2025-06-13"
	}
	
	O, en caso de error:
	{
	"error": "Ciclo escolar no encontrado”
	}

Modificar Ciclo

Elemento	***Valor***
Actor(es)	Coordinador
URL	/ciclos/{id}
Método HTTP	Put
Lógica de negocio	1. Recibir solicitud con id y datos. 2. Validar id y datos. 3. Verifica que no existan ciclos que se superpongan con las fechas ingresadas 4. Actualizar ciclo. 5. Retornar 200 o 204 .

Entrada	Un objeto JSON con los campos a modificar. No se debe poder modificar id.
	{
	"ciclo": "ENE-JUN 2025 ",
	"fechaInicio": "2025-01-07",
	"fechaFin": "2025-06-14"
	}

Salida	[
	
	{
	"id": "ObjectId()",
	"ciclo": "ENE-JUN 2025 ",
	"fechaInicio": "2025-01-07",
	"fechaFin": "2025-06-14"
	}
	]
	O, en caso de error:
	{
	"error": " Ciclo escolar no encontrado "
	}

Eliminar Ciclos

Elemento	*Valor*
Actor(es)	Coordinador
URL	/ciclos/{id}
Método HTTP	Delete
Lógica de negocio	1. Recibir solicitud con id. 2. Validar id. 3. Verificar dependencias. 4. Eliminar ciclo. 5. Retornar 200 más un mensaje , o 204 .
Entrada	Ninguna (el ID de ciclo se pasa en la URL)

Salida	{
	"mensaje": "Ciclo escolar eliminado exitosamente."
	"id": "ObjectId()"
	}
	
	O, en caso de error:
	{
	"error": ""No se puede eliminar el ciclo porque tiene grupos asociados”
	}

Carreras

1. **Tipo de Servicio**: Entidad (Gestiona la información de las carreras), este debido a que se realizarán operaciones CRUD sobre la entidad.
2. **Responsable de la entidad**:Lopez Quesada Carlos Emmanuel

Registrar carreras

Elemento	*Valor*
Actor(es)	Coordinador
URL	/carreras
Método HTTP	Post
Lógica de negocio	1. Recibir datos de la carrera. 2. Validar la carrera 3. Crear la carrera. 4. Retornar 201 con la cabecera Location y el nuevo recurso

Entrada	{
	"carrera": "ISC",
	"nombre": "Ingeniería en Sistemas Computacionales"
	}

Salida	{
	"id": "ObjectId(.)
	"carrera": "ISC",
	"nombre": "Ingeniería en Sistemas Computacionales"
	}
	O, en caso de error:
	{
	"error": "No se pudo registrar la carrera "
	}

Consultar General de las carreras

Elemento	*Valor*
Actor(es)	Coordinador , Tutor
URL	/carreras
Método HTTP	Get
Lógica de negocio	1. Recibir solicitud. 2. Obtener todas las carreras. 3. Retornar 200 .
Entrada	Ninguna, no se necesitan parámetros para obtener la consulta general.

Salida	[
	{
	"id": "ObjectId()",
	"carrera": "ISC",
	"nombre": "Ingeniería en Sistemas Computacionales"
	},
	{
	"id": "ObjectId(...) o string",
	"carrera": "LA",
	"nombre": "Licenciatura en Administración"
	}
	]
	O, en caso de error:
	{
	"error": "No se encontraron carreras “
	}

Consultar Carreras por ID

Elemento	*Valor*
Actor(es)	Coordinador , Tutor
URL	/carreras/{id}
Método HTTP	Get
Lógica de negocio	1. Recibir id. 2. Buscar por id. 3. Si existe, retornar 200 y la carrera; si no, 404.
Entrada	Ninguna, el id va en la UR

Salida	[
	{
	"id": "ObjectId()",
	"carrera": "ISC",
	"nombre": "Ingeniería en Sistemas Computacionales"
	}
	]
	O, en caso de error:
	{
	"error": "No se encontró la carrera “
	}

Modificar Carrera

Elemento	*Valor*
Actor(es)	Coordinador
URL	/carreras/{id}
Método HTTP	Put
Lógica de negocio	1. Recibir id y datos. 2. Validar. 3. Verificar que el nuevo valor de carrera no entre en conflicto. 4. Actualizar. 5. Retornar 200 o 204.

Entrada	Un objeto JSON con los campos a modificar. No se debe poder modificar id.
	{
	"carrera": "ISIC",
	"nombre": "Ing. en Sistemas Computacionales"
	}

Salida	[
	{
	"carrera": "ISIC",
	"nombre": "Ing. en Sistemas Computacionales"
	}
	]
	O, en caso de error:
	{
	"error": " Ciclo escolar no encontrado"
	}

Eliminar Carrera

Elemento	*Valor*
Actor(es)	Coordinador.
URL	/carreras/{id}
Método HTTP	Delete
Lógica de negocio	1. Recibir id. 2. Validar id. 3. Verificar dependencias (usuarios, grupos). 4. Eliminar. 5. Retornar 200 más un mensaje, o 204
Entrada	Ninguna (el ID de la carrera se pasa en la URL)

Salida	{
	
	"mensaje": "Carrera eliminada exitosamente."
	
	}
	
	O, en caso de error:
	{
	"error": "La carrera no se pudo eliminar"
	}

Conclusiones

La realización de este trabajo ha permitido establecer una base sólida para el desarrollo del proyecto. La elección del tipo de base de datos y la creación del modelo correspondiente (relacional o documental) garantizan que la estructura de datos se ajusta a las necesidades identificadas. La carga inicial de datos proporciona un entorno de pruebas realista para validar el modelo y las operaciones de los servicios.

La definición detallada de los requerimientos de los servicios, incluyendo la descripción de cada operación, es un paso esencial para asegurar la correcta implementación de la lógica de negocio y la interacción con la base de datos. Este enfoque sistemático no solo facilita el desarrollo, sino que también contribuye a la mantenibilidad y escalabilidad del proyecto a largo plazo. En resumen, este trabajo sienta las bases para una implementación exitosa, minimizando riesgos y maximizando la eficiencia en las siguientes etapas del desarrollo. Se espera que, con esta base, el equipo pueda avanzar con confianza hacia la construcción de un sistema robusto y funcional, ademas de ser escalable y con buen rendimiento.