

Lab7 - APIs REST y Silence

Pepe Calderón, Fernando Sola, Daniel Ayala, Inma Hernández, Margarita Cruz, Carlos Arévalo, David Ruiz



Escuela Técnica Superior de
Ingeniería Informática

Índice

1. Objetivo	1
2. Introducción	2
3. Preparación del entorno	3
3.1. Qué hacer si el comando <code>silence</code> no se reconoce	5
4. Creación de un nuevo proyecto	10
5. Configuración del proyecto	11
6. Definición de la BD del proyecto	13
7. Definición de los endpoints del proyecto	15
8. Ejecución del proyecto y uso de endpoints	18
8.1. Consultas GET	19
8.2. Registro	21
8.3. Login	24
9. Creación, actualización y borrado	26
10. Definición de endpoints personalizados	29
11. Pruebas de aceptación HTTP	32
11.1. Pruebas de aceptación de Grados	32
11.2. Pruebas de aceptación de Asignaturas	34
11.3. Pruebas de aceptación de Alumnos	40
11.4. Pruebas de aceptación adicionales	45

Objetivo

El objetivo de esta práctica es usar el framework de backend Silence para implementar una API RESTful con la que acceder y modificar los elementos existentes en una base de datos relacional. El alumno aprenderá a:

- Instalar el framework Silence
 - Crear y configurar un proyecto Silence
 - Crear los scripts de creación y poblado de la BD del proyecto
 - Crear los endpoints asociados a la base de datos
 - Realizar pruebas sobre la API RESTful creada.
-

Introducción

Silence es un framework con propósito educativo creado en la Universidad de Sevilla para facilitar la construcción de APIs RESTful y aplicaciones web a partir de una base de datos relacional . Silence cuenta con una metodología de trabajo basada en proyectos, en la que cada proyecto contiene una estructura de base de datos, aplicación web y configuración propia.

Silence es una herramienta de código abierto, y su código fuente está disponible en su repositorio en GitHub.

Preparación del entorno

Silence se encuentra publicado en el índice de paquetes de Python, por lo que puede instalarse haciendo uso del gestor de paquetes `pip`, que debería estar disponible si se siguieron las instrucciones de instalación del primer laboratorio.

Para instalar Silence, abriremos una consola **con permisos de administrador*** y ejecutaremos el comando:

```
pip install silence
```

* Los permisos de administrador son necesarios para instalar Silence a nivel de sistema y poder usar el comando `silence` en cualquier lugar (el equivalente en Linux es usar `sudo pip install silence`). Se puede realizar una instalación sin permisos especiales usando un entorno virtual de Python, pero esta operación no está cubierta por este boletín.

```
Administrador: Símbolo del sistema
C:\Windows\system32>pip install silence
Collecting silence
  Downloading Silence-1.0.1-py3-none-any.whl (30 kB)
Collecting PyMySQL==0.9.3
  Downloading PyMySQL-0.9.3-py2.py3-none-any.whl (47 kB)
    |████████████████████| 47 kB 928 kB/s
Collecting dload==0.6
  Downloading dload-0.6-py3-none-any.whl (8.1 kB)
Collecting Flask==1.1.2
  Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
    |████████████████████| 94 kB 625 kB/s
Collecting Flask-Cors==3.0.8
  Downloading Flask_Cors-3.0.8-py2.py3-none-any.whl (14 kB)
Collecting colorama==0.4.3
  Downloading colorama-0.4.3-py2.py3-none-any.whl (15 kB)
Collecting pypika==0.37.7
  Downloading PyPika-0.37.7.tar.gz (53 kB)
    |████████████████████| 53 kB 358 kB/s
Collecting Werkzeug==1.0.1
  Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
    |████████████████████| 298 kB 6.4 MB/s
```

Esto instalará Silence y todas sus dependencias. Si el proceso es correcto, se nos informará de que todo ha quedado instalado:

```
Successfully installed Flask-1.1.2 Flask-Cors-3.0.8 Jinja2-2.11.2 MarkupSafe-1.1.1
PyMySQL-0.9.3 Six-1.15.0 Werkzeug-1.0.1 certifi-2020.6.20 chardet-3.0.4 click-7.1.2
colorama-0.4.3 dload-0.6 idna-2.10 itsdangerous-1.1.0 pypika-0.37.7 requests-2.24.
0 silence-1.0.1 urllib3-1.25.10
```

Podemos comprobar que tenemos acceso al comando `silence` usándolo en la consola, por ejemplo, comprobando la versión que se ha instalado con el comando:

```
silence --version
```

```
PS C:\Users\Agu\Desktop> silence --version
Silence v2.1.0
```

Si se desea actualizar el framework Silence para estar al día con una actualización publicada basta con ejecutar el comando de instalación, pero introduciendo `--upgrade` antes del paquete a actualizar:

```
pip install --upgrade silence
```

i Info

Si ya cuenta con una versión de Silence instalada del curso pasado, recomendamos encarecidamente que la actualice para asegurarse de que cuenta con la última versión, usando el comando anterior.

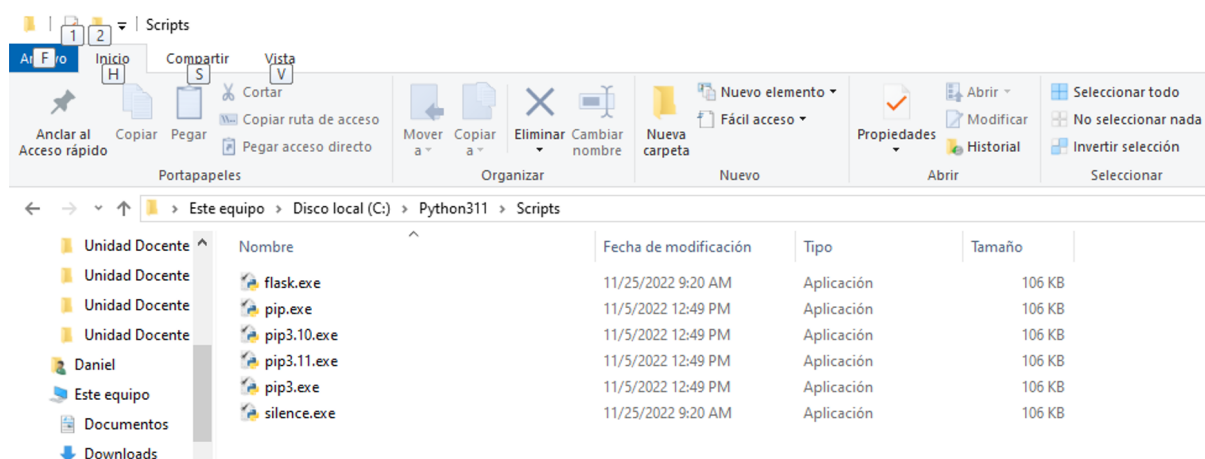
3.1. Qué hacer si el comando `silence` no se reconoce

Tras instalar Silence exitosamente, obtiene un error indicando que el comando `silence` no se reconoce como el siguiente:

```
'silence' no se reconoce como un comando interno o externo,  
programa o archivo por lotes ejecutable.
```

En Windows, se reconoce una aplicación como comando cuando su archivo ejecutable se encuentra en una de las rutas incluidas en la variable de entorno `PATH`. Si el comando `silence` no se reconoce, es posible que el directorio donde se ha instalado Silence no esté en la variable `PATH`.

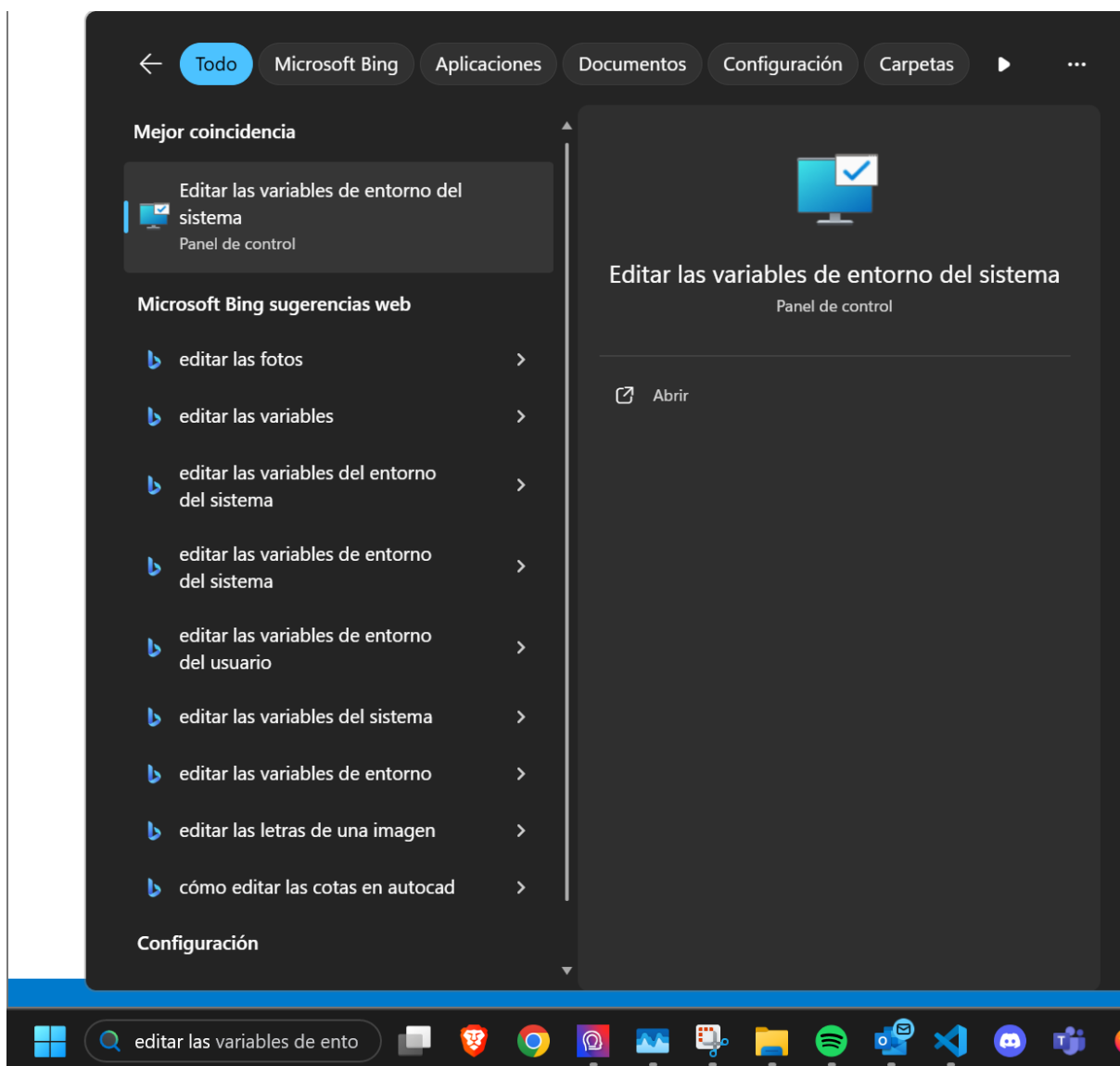
`Silence.exe` se crea al instalar Silence en la carpeta `<instalación de Python>\Scripts`. Puede encontrar esta ruta fácilmente si se fija en los mensajes de la consola durante la instalación de Silence.

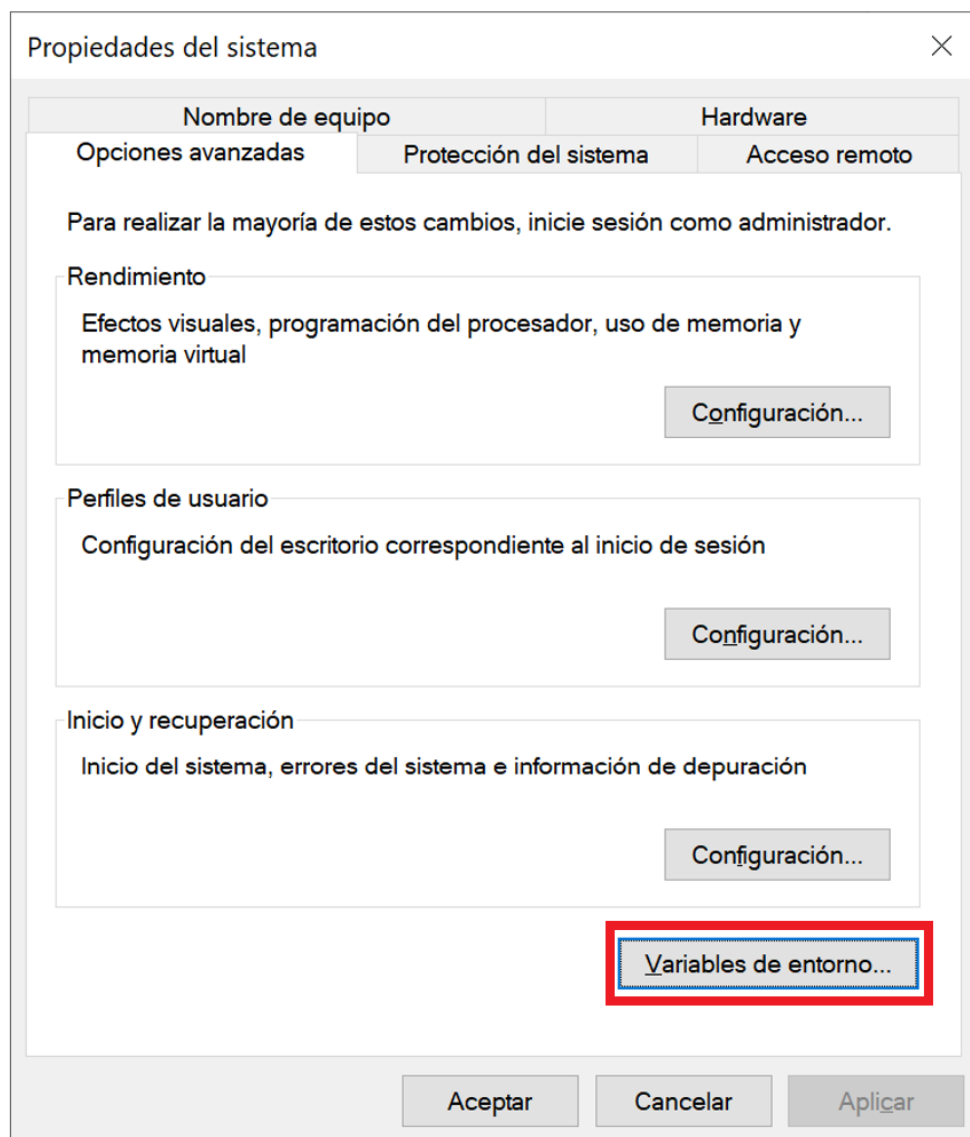


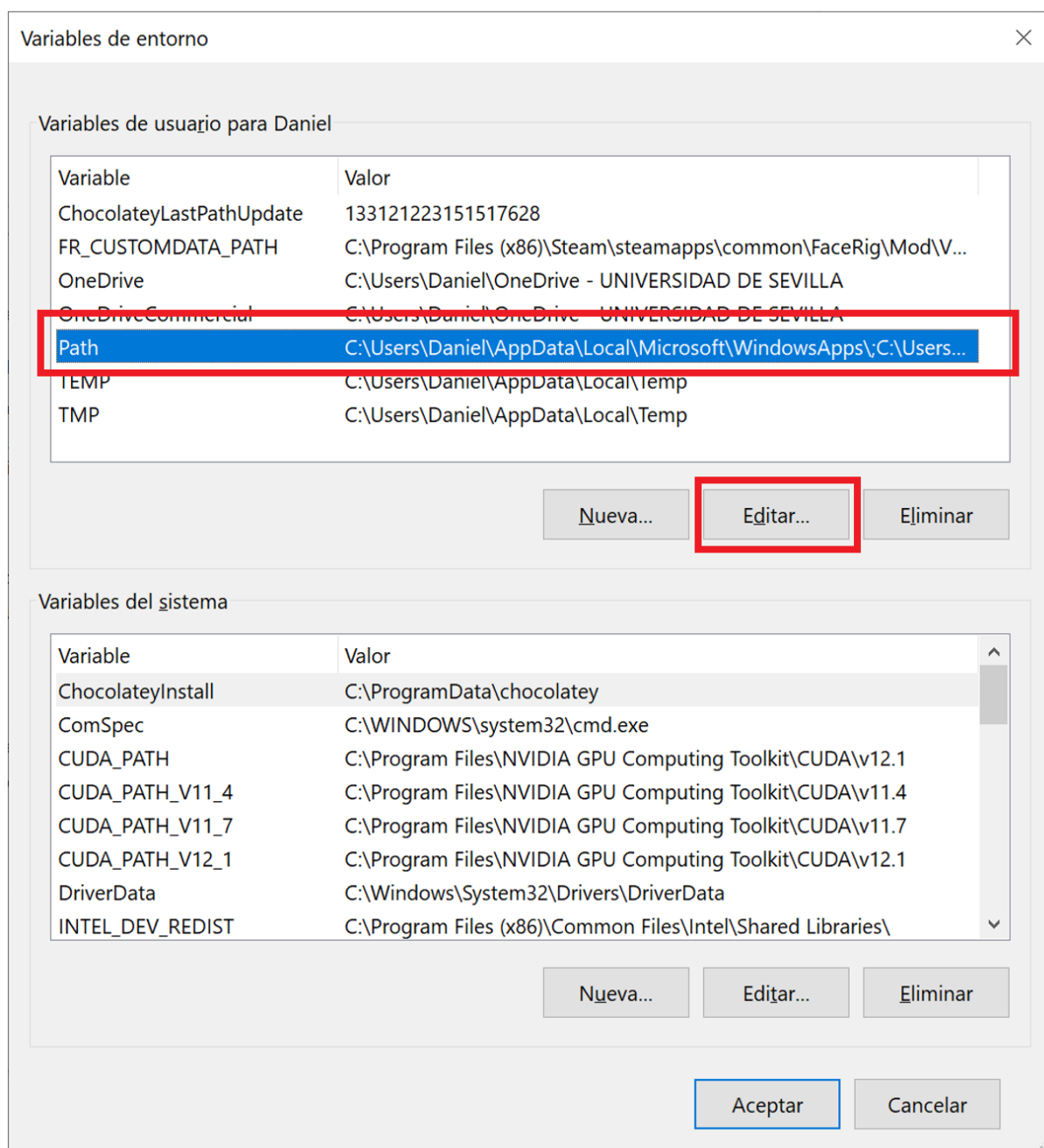
Siga estos pasos para añadir la ruta al directorio donde se encuentra `silence.exe` a la variable de entorno `PATH`:

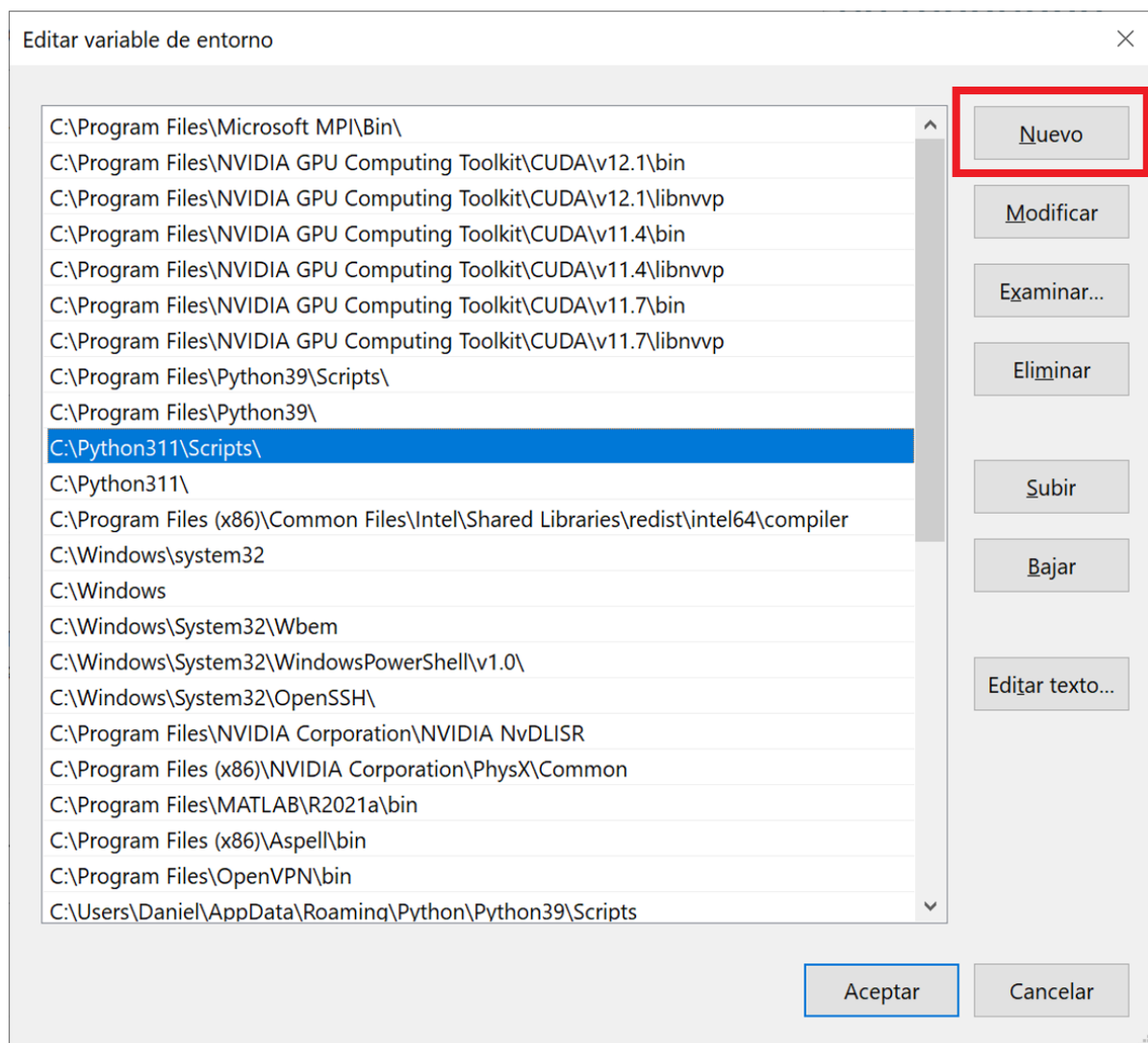
1. Abra el menú de inicio (pulse tecla Windows) y busque “Editar las variables de entorno del sistema”.
2. En la ventana que se abre, haga clic en el botón “Variables de entorno”.
3. En la sección “Variables del sistema”, busque la variable `PATH`, selecciónela y pulse el botón “Editar”.

4. Haga click en “Nuevo” y añada la ruta al directorio donde se encuentra `Silence.exe`.
5. Acepte los cambios y cierre las ventanas.
6. Reinicie la consola y pruebe a ejecutar el comando `silence --version` de nuevo. Si sigue obteniendo el error, reinicie el equipo y pruebe de nuevo.









Creación de un nuevo proyecto

i Info

Desde este punto en adelante, no es necesario tener permisos de administración en la consola

Para crear un proyecto Silence, navegaremos con la consola a la ubicación donde deseemos crear el proyecto y ejecutaremos el comando:

```
silence new <nombre> --template blank
```

Donde `<nombre>` es el nombre que le daremos a este proyecto. La opción `--template blank` indica que se creará un proyecto con una estructura de base de datos vacía, por lo que podemos usar la base de datos en la que hemos trabajado en los anteriores laboratorios:

```
PS C:\Users\Agu\Desktop> silence new proyecto_lab --template blank
The Silence project "proyecto_lab" has been created using the template 'blank'.
```

Configuración del proyecto

Cada proyecto contiene un archivo `settings.py` que almacena la configuración específica del proyecto. A continuación mostraremos las principales opciones a configurar en este archivo:

El parámetro `DEBUG_ENABLED` controla si se muestran o no mensajes de depuración. Si se activan, aparecerán mensajes que permiten conocer qué está haciendo internamente el framework en cada momento. Los mensajes de depuración aparecen en gris en la consola.

```
DEBUG_ENABLED = False
```

El parámetro `DB_CONN` configura los datos de acceso a la BD y la base de datos a usar para el proyecto:

```
DB_CONN = {  
    "host": "127.0.0.1",  
    "port": 3306,  
    "username": "iissi_user",  
    "password": "iissi$user",  
    "database": "grados",  
}
```

El parámetro `SQL_SCRIPTS` controla qué scripts SQL se ejecutarán, y en qué orden, cuando se le indique a Silence que debe crear la base de datos del proyecto. Los scripts SQL deben colocarse en la carpeta `sql/` del proyecto.

Para que nuestro proyecto funcione correctamente, debemos crear dos scripts SQL:

- `tables.sql` : Contiene el código SQL necesario para crear las tablas de la base de datos.

- `populate.sql` : Contiene el código SQL necesario para poblar las tablas con datos iniciales.

El código de estos scripts puede encontrarlos en los correspondientes anexos `pCreateDB` y `pPopulateDB`.

Y también podemos añadir cualquier otro script que sea necesario para crear nuestra BD.

A continuación, configuraremos el parámetro `SQL_SCRIPTS` para que se ejecuten en el orden correcto:

```
SQL_SCRIPTS = [  
    "tables.sql",  
    "populate.sql",  
]
```

Es importante tener en cuenta que los scripts SQL deben estar en el orden adecuado, ya que si no, podríamos tener problemas al crear la base de datos e insertar los datos correspondientes.

Los parámetros `HTTP_PORT` y `API_PREFIX` permiten configurar el puerto a usar para el servidor HTTP y el prefijo que tendrán todas las rutas de la API, respectivamente. Mantendremos los valores por defecto:

```
HTTP_PORT = 8080  
API_PREFIX = "/api/v1"
```

El parámetro `USER_AUTH_DATA` permite indicar qué tabla es la que se usará para identificar usuarios, y dentro de esta tabla, qué columnas corresponden al identificador del usuario y a su contraseña. Estos datos son necesarios para que Silence pueda proveer registro y login de usuarios, así como protección de endpoints para usuarios registrados y control de roles.

En los scripts SQL proporcionados para este boletín, se ha añadido un campo `password` a la tabla de estudiantes, para permitir hacer login y registro con ellos.

```
USER_AUTH_DATA = {  
    "table": "Students",  
    "identifier": "email",  
    "password": "password",  
}
```

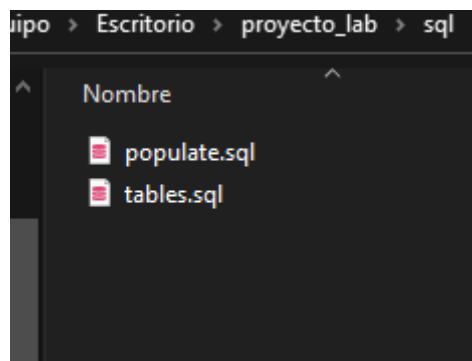
Finalmente, el parámetro `SECRET_KEY` se utiliza para elementos internos del framework que requieren una cadena secreta aleatoria criptográficamente segura. Este parámetro se genera de manera automática cuando se crea un nuevo proyecto usando el comando `silence new`.

Existen muchos otros parámetros de configuración que pueden encontrarse en la Wiki de Silence en GitHub.

Definición de la BD del proyecto

Cada proyecto Silence tiene asociada una estructura de base de datos y una carga inicial de datos, de manera que sea sencillo desplegar éstas siempre que la configuración de acceso al SGBD sea correcta.

Una vez hemos configurado en la sección anterior los datos de acceso a la BD, añadiremos a nuestro proyecto los scripts de creación de la BD y de la carga inicial de datos, en la carpeta `sql/`. Usaremos para ello los scripts proporcionados en conjunto con este boletín:



Una vez estos scripts se encuentran en la carpeta adecuada, y su orden de ejecución ha sido definido en el parámetro `SQL_SCRIPTS` del archivo `settings.py`, podemos inicializar automáticamente la base de datos usando el comando:

```
silence createdb
```

Si el proceso es correcto, se mostrará todo el código SQL que se está ejecutando:

```
Windows PowerShell  Debian
(2, 3, 1),
(3, 7, 1),
(4, 8, 1),
(5, 10, 1),
(6, 12, 1),
(7, 2, 2),
(8, 3, 2),
(9, 10, 2),
(10, 12, 2),
(11, 18, 21),
(12, 21, 21);

INSERT INTO Grades (gradeId, value, gradeCall, withHonours, studentId, groupId) VALUES
(1, 4.50, 1, 0, 1, 1),
(2, 3.25, 2, 0, 1, 1),
(3, 9.95, 1, 0, 1, 7),
(4, 7.5, 1, 0, 1, 10),
(5, 2.50, 1, 0, 2, 2),
(6, 5.00, 2, 0, 2, 2),
(7, 10.00, 1, 1, 2, 10),
(8, 0.00, 1, 0, 21, 18),
(9, 1.25, 2, 0, 21, 18),
(10, 0.5, 3, 0, 21, 18);

Done!
(venv) PS C:\Users\Agu\Desktop\proyecto_lab>
```

Tener la estructura de la BD y sus datos almacenados en el proyecto favorecen un despliegue de las mismas más sencillo, y permite devolverla a un estado controlado en cualquier momento simplemente ejecutando el comando `silence createdb`.

Definición de los endpoints del proyecto

Un endpoint representa una operación que puede realizarse sobre un recurso, que se ejecuta cuando se recibe una petición HTTP determinada a una ruta concreta. En Silence, los endpoints del proyecto se definen usando la notación JSON, donde para cada endpoint se define obligatoriamente:

- La ruta del endpoint
- El método HTTP asociado
- La consulta SQL que debe ejecutarse

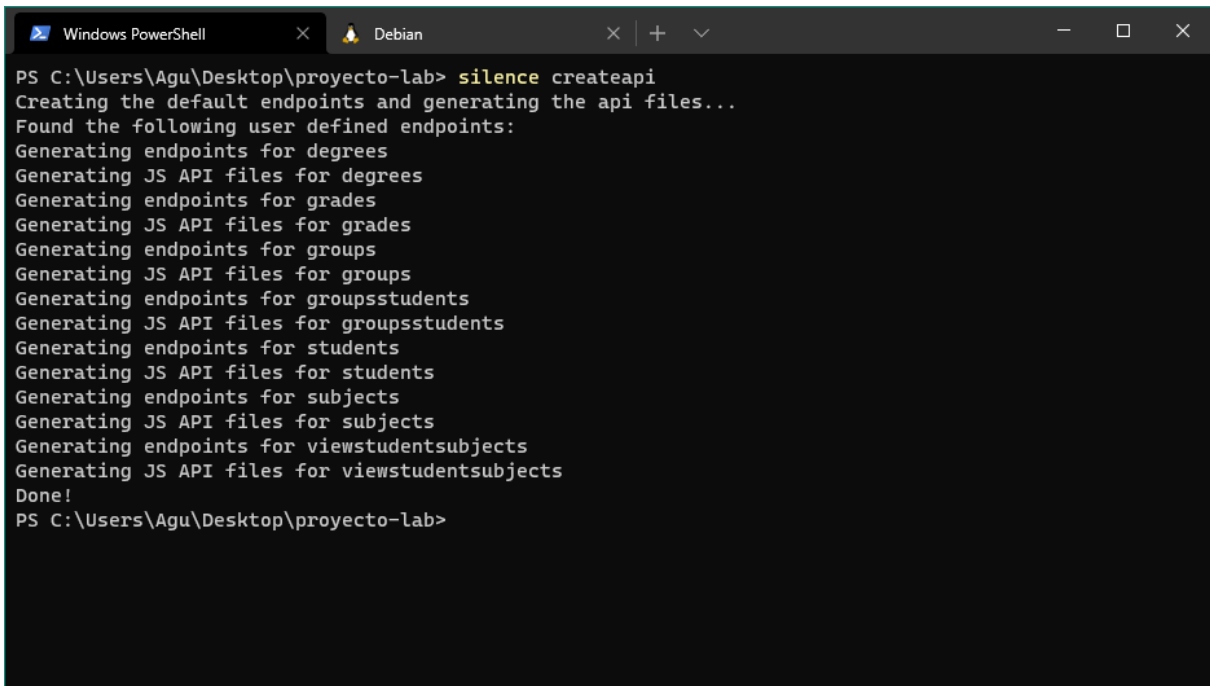
Opcionalmente, también puede especificarse:

- La descripción del endpoint
- Si puede ser usado por todos los usuarios o sólo por los usuarios autenticados
- Si puede ser usado por cualquier usuario autenticado, o sólo por aquellos que tengan un determinado rol

Generalmente, se desean definir las operaciones CRUD (crear, leer, actualizar y borrar) para cada recurso, representados como tablas en nuestra BD, y operaciones de lectura para las vistas. La definición de estos endpoints se realizan en archivos JSON que deben estar en la carpeta `endpoints/`.

Silence es capaz de analizar la estructura de una base de datos, y generar automáticamente los endpoints que implementan las operaciones descritas anteriormente para todas las entidades encontradas en ella. Para ello, basta con usar el comando:

```
silence createapi
```



```
PS C:\Users\Agu\Desktop\proyecto-lab> silence createapi
Creating the default endpoints and generating the api files...
Found the following user defined endpoints:
Generating endpoints for degrees
Generating JS API files for degrees
Generating endpoints for grades
Generating JS API files for grades
Generating endpoints for groups
Generating JS API files for groups
Generating endpoints for groupsstudents
Generating JS API files for groupsstudents
Generating endpoints for students
Generating JS API files for students
Generating endpoints for subjects
Generating JS API files for subjects
Generating endpoints for viewstudentssubjects
Generating JS API files for viewstudentssubjects
Done!
PS C:\Users\Agu\Desktop\proyecto-lab>
```

Si el proceso finaliza correctamente, se muestra una lista de las entidades para las que se han generado endpoints. Es importante recordar que este proceso se realiza en base a **la estructura de la BD**, por lo que es necesario tener definida en ella las tablas antes de ejecutar el comando.

Los endpoints autogenerados se encuentran en la carpeta `endpoints/auto/`, agrupados en un archivo JSON por cada entidad encontrada en la BD. A modo de ejemplo, podemos analizar el endpoint de consulta para la tabla Degrees:

```
"getAll": {
  "route": "/degrees",
  "method": "GET",
  "sql": "SELECT * FROM degrees",
  "description": "Gets all degrees",
  "auth_required": false,
  "allowed_roles": ["*"]
}
```

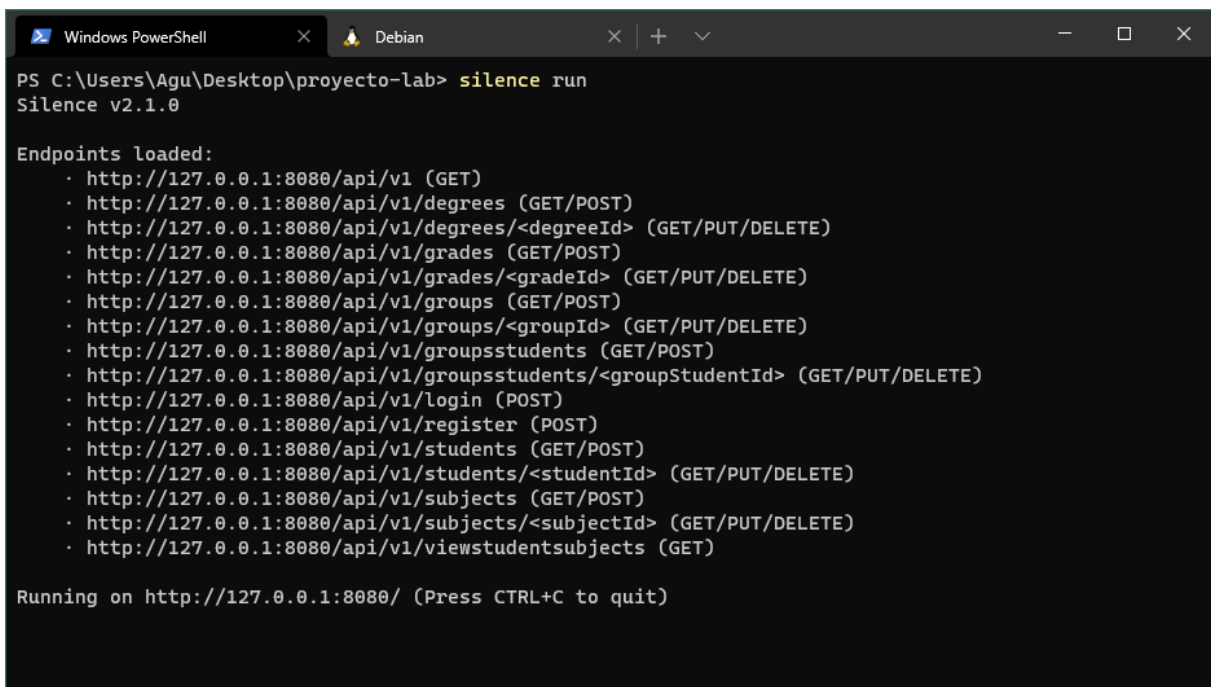
Como se observa, este endpoint asocia la consulta SQL `SELECT * From degrees` a la ruta `/degrees` y el método `GET`. Además, se especifica que no es necesario estar autenticado para acceder al endpoint, que puede ser usado por usuarios con cualquier rol (`*`), y se provee una descripción del mismo.

Por defecto, los endpoints de consulta pueden ser usados por usuarios no autenticados, mientras que los de modificación (creado, actualizado y borrado) sólo pueden ser usados por usuarios autenticados. Es posible definir endpoints personalizados mediante archivos JSON en la carpeta `endpoints/`, usando la sintaxis mostrada.

Ejecución del proyecto y uso de endpoints

Una vez está la base de datos inicializada y los endpoints definidos, podemos lanzar nuestro proyecto usando el comando:

```
silence run
```



```
PS C:\Users\Agu\Desktop\proyecto-lab> silence run
Silence v2.1.0

Endpoints loaded:
  · http://127.0.0.1:8080/api/v1 (GET)
  · http://127.0.0.1:8080/api/v1/degrees (GET/POST)
  · http://127.0.0.1:8080/api/v1/degrees/<degreeId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/grades (GET/POST)
  · http://127.0.0.1:8080/api/v1/grades/<gradeId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/groups (GET/POST)
  · http://127.0.0.1:8080/api/v1/groups/<groupId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/groupsstudents (GET/POST)
  · http://127.0.0.1:8080/api/v1/groupsstudents/<groupStudentId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/login (POST)
  · http://127.0.0.1:8080/api/v1/register (POST)
  · http://127.0.0.1:8080/api/v1/students (GET/POST)
  · http://127.0.0.1:8080/api/v1/students/<studentId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/subjects (GET/POST)
  · http://127.0.0.1:8080/api/v1/subjects/<subjectId> (GET/PUT/DELETE)
  · http://127.0.0.1:8080/api/v1/viewstudentsubjects (GET)

Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

Para hacer uso de los endpoints utilizaremos la extensión REST Client para VSCode, cuyo uso explicamos a continuación.

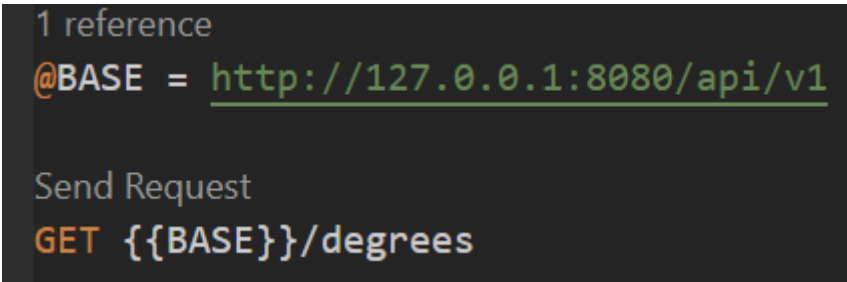
8.1. Consultas GET

REST Client utiliza archivos `.http` que pueden contener una o varias consultas. En nuestra asignatura, agruparemos estas consultas en varios archivos, según la entidad a la que estemos accediendo. A modo de ejemplo, haremos consultas GET a Degrees, para lo cual crearemos un archivo `degrees.http` **en una nueva carpeta, a la que llamaremos `requests`**.

En primer lugar, definiremos la URL base, a partir de la cual se construirán todas las consultas a nuestra API, para evitar tener que repetirla cada vez:

```
@BASE = http://127.0.0.1:8080/api/v1
```

A continuación, podemos usarla para realizar consultas GET al endpoint `/degrees`. Para hacer una petición HTTP basta con indicar el verbo y la URL, y pulsar en el botón “Send request” que aparece en la parte superior. Se pueden usar las variables definidas anteriormente usando dobles llaves:



```
1 reference
@BASE = http://127.0.0.1:8080/api/v1

Send Request
GET {{BASE}}/degrees
```

El servidor ejecutará la consulta SQL asociada a la ruta y método, y devolverá una respuesta en formato JSON:

```

✓[
✓ {
  "degreeId": 1,
  "name": "Ingenier\u00eda del Software",
  "years": 4
},
✓ {
  "degreeId": 2,
  "name": "Ingenier\u00eda del Computadores",
  "years": 4
},
✓ {
  "degreeId": 3,
  "name": "Tecnolog\u00edas Inform\u00e1ticas",
  "years": 4
}
]

```

Además de una consulta a todas las entradas de la tabla Degrees, los endpoints generados automáticamente permiten consultar una de ellas por su ID, mediante la ruta `/degrees/id`. Para ello, basta con indicar el ID en la URL, y pulsar en el botón “Send request”:

```

5   ###
6
   Send Request
7   GET {{BASE}}/degrees/2
8

```

A lo que el servidor responde:

```
{
  "degreeId": 2,
  "name": "Ingenier\u00eda del Computadores",
  "years": 4
}
```

Es importante destacar que, para tener **varias consultas en el mismo archivo**, éstas deben **separarse mediante una línea con tres almohadillas (###)**.

Intentemos ahora, por ejemplo, borrar el grado con ID 2 mediante el endpoint asociado al verbo DELETE:

```
Send Request
DELETE {{BASE}}/degrees/2
```

```
{
  "code": 401,
  "message": "Unauthorized"
}
```

En este caso, el servidor responde con un código de error HTTP 401 (no autorizado). Esto se debe a que el endpoint correspondiente está protegido, y sólo los usuarios autenticados pueden acceder a él. En la siguiente sección registraremos un nuevo usuario, y aprenderemos a hacer login con uno de los usuarios ya existentes para tener acceso a los endpoints protegidos.

8.2. Registro

Silence provee un endpoint por defecto, `/register`, para realizar el registro de un nuevo usuario. Para que funcione correctamente, el parámetro `USER_AUTH_DATA` debe estar definido correctamente en `settings.py`, indicando el nombre de la tabla que almacena los usuarios y los campos (columnas) que se usan para el login. En nuestro caso, serán la tabla “Students” y las columnas “email” y “password” respectivamente.

Una vez configurado, se le pueden enviar al servidor los datos de un nuevo usuario en formato JSON, incluyendo tantos campos como sean necesarios en la tabla Students. La contraseña enviada será almacenada de manera segura en forma de hash¹, y no será visible en la respuesta.

Para registrar un nuevo usuario, enviaremos una petición HTTP POST a la ruta `/register`, con los datos del nuevo usuario en formato JSON. Cuando se envían datos en una petición POST o PUT en el cuerpo de la petición, es importante indicarle al servidor el formato de los mismos, que en este caso es “application/json”:

```
POST http://127.0.0.1:8080/api/v1/register
Content-Type: application/json

{
  "accessMethod": "Selectividad",
  "dni": "99999999Z",
  "firstName": "Nuevo",
  "surname": "Usuario",
  "birthDate": "2000-01-01",
  "email": "nuevo@alum.us.es",
  "password": "password123"
}
```

```
Send Request
POST {{BASE}}/register
Content-Type: application/json

{
  "firstName": "Elvis",
  "surname": "Presley",
  "dni": "98421245J",
  "birthDate": "1935-01-08",
  "accessMethod": "Titulado Extranjero",
  "email": "elvis@alum.us.es",
  "password": "elvis1234"
}
```

Si el registro es exitoso, el servidor devolverá los datos del usuario creado, y un token de sesión. Este token es necesario para acceder a los endpoints protegidos, ya que contiene la información del usuario que ha iniciado sesión:

¹ Más información sobre los conceptos básicos de este proceso: <https://security.stackexchange.com/a/31846>

```

{
  "sessionToken": ".eJwtjstqwzAQRX9FiCxDbQp1rzqoqE0
kNKff2BsjbASW071aBtK_72R6TCrM4e594fCMGBKZ8zjYm1La0dz
mcAu5PidI4QLxoVuCe19zOMzZFydg1vCND1BINzsFWGsXZe8nLtq
2-CrZ7QUXEh1qgxn8F01OH369ARTmXc17TDVo_Mx5TeY1_fHK1R6
g5S-1rj2uvVX60SbRhDq0HLFhrNhf1Ef92u3cY0Z0B1mDQotXSM0
cnCcH_aSaa01HoyzstHKCjZwKV1fFSGgefTXTta01IvFkF9rnBAV
1Bj-K71HTBPe6e8fyJ5Yng.YYkreQ.dGI0jayT3dkCZv3PgABLJ0
Wqb7k",
  "user": {
    "accessMethod": "Titulado Extranjero",
    "birthDate": "Tue, 08 Jan 1935 00:00:00 GMT",
    "dni": "98421245J",
    "email": "elvis@alum.us.es",
    "firstName": "Elvis",
    "studentId": 22,
    "surname": "Presley"
  }
}

```

Por defecto, los tokens de sesión son válidos durante 24 horas. Pasado ese tiempo, caducan y no son considerados tokens válidos.

Si se intentara registrar de nuevo el mismo usuario, el servidor devolverá un error:

```

{
  "code": 400,
  "message": "There already exists another user with
that email"
}

```

8.3. Login

Como se ha mostrado en la sección anterior, una manera de obtener un token de sesión es registrar un usuario. Otra manera es iniciar sesión con un usuario ya existente. Para ello, se debe enviar una petición POST a la ruta `/login`, con los datos de inicio de sesión del usuario en formato JSON. El servidor emitirá una respuesta idéntica a la del registro, con los datos del usuario que ha iniciado sesión (excepto su contraseña) y un token de sesión:

```
POST {{BASE}}/login
Content-Type: application/json

{
  "email": "elvis@alum.us.es",
  "password": "elvis1234"
}
```

```
✓{
  "sessionToken": ".eJwtjstqwzAQRX9FiCxDbQp1rzqoqE0
kNKff2BsjbASW071aBtK_72R6TCrM4e594fCMGBKZ8zjYm1LaOdz
mcAu5PidI4QLxoVuCe19zOMzZFydg1vCND1BINzsFWGsXZe8nLtq
2-CrZ7QUXEh1qgxn8F0lOH369ARTmXc17TDVo_Mx5TeY1_fHK1R6
g5S-1rj2uvVX60SbRhDq0HLFhrNhflEf92u3cY0Z0B1mDQotXSM0
cnCcH_aSaa01HoyzstHKCjZwKV1fFSGgefTXTta01IvFkF9rnBAV
lBj-K71HTBPe6e8fyJ5Yng.YYkreQ.dGI0jayT3dkCZv3PgABLJ0
Wqb7k",
  "user": {
    "accessMethod": "Titulado Extranjero",
    "birthDate": "Tue, 08 Jan 1935 00:00:00 GMT",
    "dni": "98421245J",
    "email": "elvis@alum.us.es",
    "firstName": "Elvis",
    "studentId": 22,
    "surname": "Presley"
  }
}
```

Creación, actualización y borrado

Si queremos acceder a los endpoints protegidos para usuarios autenticados, debemos enviar el token de sesión como cabecera de la petición HTTP. En nuestro caso, el nombre de la cabecera será `Token`, y el valor será el token de sesión.

Dada la longitud de estos tokens, y el hecho de que expiran a las 24h de ser creados, no es una buena idea proveerlos directamente como cadenas en la cabecera de nuestras peticiones HTTP, ya que dificultaría la lectura del archivo y las peticiones asociadas dejarían de funcionar al día siguiente. En su lugar, podemos darle un nombre a la petición de login anterior y, usando los datos que nos ha devuelto el servidor en ella, referenciarlo en las siguientes peticiones.

Como ejemplo, en el siguiente fragmento de código creamos un nuevo grado (Degree) mediante una petición POST a la ruta `/degrees`. Para que el servidor nos permita realizar la operación, antes realizaremos un login, y referenciaremos el token de sesión obtenido:

```
# @name login
Send Request
POST {{BASE}}/login
Content-Type: application/json

{
  "email": "elvis@alum.us.es",
  "password": "elvis1234"
}

###

Send Request
POST {{BASE}}/degrees
Content-Type: application/json
Token: {{login.response.body.sessionToken}}

{
  "name": "Ingeniería de Sistemas",
  "years": 4
}
```

Ahora sí, el servidor aceptará la operación y nos devolverá el ID del nuevo recurso creado:

```
{
  "lastId": 6
}
```

Para modificar un recurso, usaremos el método HTTP PUT. Para ello, en la petición HTTP, indicaremos el ID del recurso que queremos modificar, y en el cuerpo de la petición, los nuevos atributos del recurso. Es importante destacar que, en una petición de tipo PUT, se deben enviar todos los atributos del recurso, y no sólo los que se desean modificar:

```

Send Request
PUT {{BASE}}/degrees/6
Content-Type: application/json
Token: {{login.response.body.sessionToken}}

{
  "name": "Ingeniería de la Salud",
  "years": 4
}

```

Podemos comprobar que los cambios se han efectuado en la base de datos:

🔑 degreeId	🔑 name	years
1	Ingeniería del Software	4
2	Ingeniería del Computadores	4
3	Tecnologías Informáticas	4
6	Ingeniería de la Salud	4

Finalmente, podemos eliminar un recurso mediante una petición DELETE. Al igual que en las peticiones anteriores, los endpoints generados automáticamente requieren el token de sesión para modificar el estado de la base de datos. Para eliminar el grado que acabamos de crear, en la petición HTTP, indicaremos el ID del recurso que queremos eliminar:

```

Send Request
DELETE {{BASE}}/degrees/6
Token: {{login.response.body.sessionToken}}

```

En el caso de las peticiones DELETE, no es necesario enviar contenido de ningún tipo en el cuerpo de la petición.

Definición de endpoints personalizados

Todos los endpoints que hemos usado hasta el momento han sido generados automáticamente por Silence, usando el comando `silence createapi`. Aunque son útiles, en ocasiones queremos definir nuestros propios endpoints, para ejecutar consultas más complejas.

Por ejemplo, consideremos que es necesario tener un endpoint que, a partir de un grado concreto, nos devuelva las asignaturas que se imparten en él. De acuerdo a la teoría, la ruta adecuada para tal endpoint sería:

```
GET /degrees/$degreeId/subjects
```

Asimismo, la consulta SQL que devuelve las asignaturas de un grado es:

```
SELECT * FROM Subjects WHERE degreeId = $degreeId
```

Donde `$degreeId` es la ID del grado en cuestión.

Para asociar esa consulta a la ruta deseada, debemos definir un nuevo endpoint. Para ello, crearemos un nuevo archivo JSON en la carpeta `endpoints` (NO en la carpeta `auto`, que contiene los endpoints autogenerados). Dado que la entidad que se devolverá es Degree, llamaremos al archivo `degrees.json`.

Este archivo contendrá una lista de endpoints, con el mismo formato que los generados por Silence. Por ejemplo, el endpoint deseado se definiría así:

```
{
  "getByDegree": {
    "route": "/degrees/$degreeId/subjects",
    "method": "GET",
    "sql": "SELECT * FROM Subjects WHERE degreeId = $degreeId",
    "auth_required": false,
    "allowed_roles": ["*"],
    "description": "Gets the subjects of a degree"
  }
}
```

Si ejecutamos de nuevo el servidor con `silence run`, podremos ver que el endpoint definido se ha cargado correctamente. Podemos probarlo haciendo una petición GET:

```
Send Request
GET {{BASE}}/degrees/2/subjects
```

A lo que el servidor responde con las asignaturas de dicho grado:

```
✓[
✓ {
  "acronym": "IMD",
  "credits": 6,
  "degreeId": 2,
  "name": "Introducci\u00f3n a la Matematica Discreta",
  "subjectId": 6,
  "type": "Formacion Basica",
  "year": 1
},
✓ {
  "acronym": "RC",
  "credits": 6,
  "degreeId": 2,
  "name": "Redes de Computadores",
  "subjectId": 7,
  "type": "Obligatoria",
  "year": 2
},
✓ {
  "acronym": "TG",
  "credits": 6,
  "degreeId": 2,
  "name": "Teor\u00eda de Grafos",
```

Pruebas de aceptación HTTP

A continuación, realizaremos las pruebas de aceptación del dominio del problema vistas en el laboratorio anterior en formato SQL, pero a través de peticiones HTTP. Considere ejecutar `silence createdb` antes de realizar las pruebas de cada tabla, ya que a diferencia de las pruebas SQL, estas no ejecutan el script de poblado cada vez.

11.1. Pruebas de aceptación de Grados

Cree un archivo `degrees.http` en la carpeta `requests` y añada las siguientes peticiones:

```
@BASE = http://127.0.0.1:8080/api/v1

### Login as a valid user (Daniel)
# @name login
POST {{BASE}}/login
Content-Type: application/json
{
    "email": "daniel@alum.us.es",
    "password": "daniel"
}
###
# [P] 1. Crear grado con datos correctos.
POST {{BASE}}/degrees
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
```

```

    "name": "Grado en Ingeniería de la Salud",
    "years": 4
}
###
# [N] 2. Crear grado con nombre vacío.
POST {{BASE}}/degrees
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": null,
    "years": 4
}
###
# [N] 3. Crear grado con el mismo nombre que otro ya existente.
POST {{BASE}}/degrees
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Grado en Ingeniería de la Salud",
    "years": 4
}
###
# [N] 4. Crear grado con years incorrectos.
POST {{BASE}}/degrees
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Grado con anyos incorrectos",
    "years": -1
}
###
# [P] 5. Actualizar grado con datos correctos.
PUT {{BASE}}/degrees/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Grado en Ingeniería de la Salud Actualizado",
    "years": 3
}
###
# [N] 6. Actualizar grado con nombre a NULL.
PUT {{BASE}}/degrees/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": null,
    "years": 4
}
###
# [N] 7. Actualizar grado con el mismo nombre que otro existente.

```

```

PUT {{BASE}}/degrees/2
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Grado en Ingeniería de la Salud Actualizado",
    "years": 4
}
###
# [N] 8. Actualizar grado con years incorrectos.
PUT {{BASE}}/degrees/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Grado en Ingeniería de la Salud Actualizado",
    "years": -1
}
###
# [P] 9. Eliminar grado.
DELETE {{BASE}}/degrees/4
Token: {{login.response.body.sessionToken}}
###
# [N] 10. Eliminar grado no existente (no lanza excepción).
DELETE {{BASE}}/degrees/9999
Token: {{login.response.body.sessionToken}}
###
# [N] 11. Eliminar grado con relaciones.
DELETE {{BASE}}/degrees/1
Token: {{login.response.body.sessionToken}}

```

11.2. Pruebas de aceptación de Asignaturas

Cree un archivo `subjects.http` en la carpeta `requests` y añada las siguientes peticiones:

```

@BASE = http://127.0.0.1:8080/api/v1

### Login as a valid user (Daniel)
# @name login
POST {{BASE}}/login
Content-Type: application/json
{
    "email": "daniel@alum.us.es",
    "password": "daniel"
}
###
# [P] 1. Crear asignatura con datos correctos.
POST {{BASE}}/subjects

```

```

Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Física Informática",
    "acronym": "FI",
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
}
###
# [N] 2. Crear asignatura con nombre vacío.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": null,
    "acronym": "FP",
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
}
###
# [N] 3. Crear asignatura con el mismo nombre que otra ya existente.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Fundamentos de Programación",
    "acronym": "FP2",
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
}
###
# [N] 4. Crear asignatura con acrónimo vacío.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "name": "Física Informática",
    "acronym": null,
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
}
###

```

```
# [N] 5. Crear asignatura con el mismo acrónimo que una ya existente.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Física Informática",
  "acronym": "FP",
  "credits": 6,
  "year": 1,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 6. Crear asignatura con créditos incorrectos.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Física Informática",
  "acronym": "FI",
  "credits": -1,
  "year": 1,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 7. Crear asignatura con curso incorrecto.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Física Informática",
  "acronym": "FI",
  "credits": 6,
  "year": 7,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 8. Crear asignatura con tipo incorrecto.
POST {{BASE}}/subjects
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Física Informática",
  "acronym": "FI",
  "credits": 6,
  "year": 1,
  "type": "Incorrecto",
  "degreeId": 1
}
```

```

}
###
# [P] 9. Actualizar asignatura con valores correctos.
PUT {{BASE}}/subjects/14
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Física Informática",
  "acronym": "FI",
  "credits": 6,
  "year": 1,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 10. Actualizar asignatura con el mismo nombre que otra.
PUT {{BASE}}/subjects/2
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Fundamentos de Programación",
  "acronym": "FP2",
  "credits": 6,
  "year": 1,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 11. Actualizar asignatura con nombre a None.
PUT {{BASE}}/subjects/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": null,
  "acronym": "FP",
  "credits": 6,
  "year": 1,
  "type": "Obligatoria",
  "degreeId": 1
}
###
# [N] 12. Actualizar asignatura con el mismo acrónimo que otra.
PUT {{BASE}}/subjects/2
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "name": "Fundamentos",
  "acronym": "FP",
  "credits": 6,
  "year": 1,

```

```

    "type": "Obligatoria",
    "degreeId": 1
  }
  ###
  # [N] 13. Actualizar asignatura con acrónimo a None.
  PUT {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Fundamentos de Programación",
    "acronym": null,
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
  }
  ###
  # [N] 14. Actualizar asignatura con créditos incorrectos.
  PUT {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Fundamentos de Programación",
    "acronym": "FP",
    "credits": -1,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": 1
  }
  ###
  # [N] 15. Actualizar asignatura con curso incorrecto.
  PUT {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Fundamentos de Programación",
    "acronym": "FP",
    "credits": 6,
    "year": 7,
    "type": "Obligatoria",
    "degreeId": 1
  }
  ###
  # [N] 16. Actualizar asignatura con tipo incorrecto.
  PUT {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Fundamentos de Programación",
    "acronym": "FP",

```

```

    "credits": 6,
    "year": 1,
    "type": "Incorrecto",
    "degreeId": 1
  }
  ###
  # [P] 17. Borrar asignatura.
  DELETE {{BASE}}/subjects/3
  Token: {{login.response.body.sessionToken}}
  ###
  # [N] 18. Borrar asignatura que ha sido borrada (no lanza excepción).
  DELETE {{BASE}}/subjects/3
  Token: {{login.response.body.sessionToken}}
  ###
  # [N] 19. Borrar asignatura con relaciones.
  DELETE {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  ###
  # [N] 20. Crear una asignación entre grado y asignatura, con grado a None.
  POST {{BASE}}/subjects
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Matemática Discreta",
    "acronym": "MD",
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": null
  }
  ###
  # [N] 21. Actualizar una asignación entre grado y asignatura, con grado a
  None.
  PUT {{BASE}}/subjects/1
  Token: {{login.response.body.sessionToken}}
  Content-Type: application/json
  {
    "name": "Fundamentos de Programación",
    "acronym": "FP",
    "credits": 6,
    "year": 1,
    "type": "Obligatoria",
    "degreeId": null
  }
}

```

11.3. Pruebas de aceptación de Alumnos

Cree un archivo `students.http` en la carpeta `requests` y añada las siguientes peticiones:

```
@BASE = http://127.0.0.1:8080/api/v1

### Login as a valid user (Daniel)
# @name login
POST {{BASE}}/login
Content-Type: application/json
{
  "email": "daniel@alum.us.es",
  "password": "daniel"
}

###
# [P] 1. Crear alumno con datos correctos.
POST {{BASE}}/register
Content-Type: application/json
{
  "accessMethod": "Selectividad",
  "dni": "12345678F",
  "firstName": "David",
  "surname": "Ruiz",
  "birthDate": "2000-01-01",
  "email": "david.ruiz@example.com",
  "password": "david"
}

###
# [N] 2. Crear alumno con el mismo DNI que otro.
POST {{BASE}}/register
Content-Type: application/json
{
  "accessMethod": "Selectividad",
  "dni": "12345678A",
  "firstName": "David",
  "surname": "Ruiz",
  "birthDate": "2000-01-01",
  "email": "david.ruiz@example2.com",
  "password": "david"
}

###
# [N] 3. Crear alumno con el mismo email que otro.
POST {{BASE}}/register
Content-Type: application/json
{
  "accessMethod": "Selectividad",
  "dni": "12345678F",
  "firstName": "Daniel",
  "surname": "Ayala",
  "birthDate": "2000-01-01",
```

```

    "email": "daniel@alum.us.es",
    "password": "daniel"
}
###
# [N] 4. Crear alumno con formato de fecha de nacimiento incorrecta.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-32",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [N] 5. Crear alumno con método de acceso incorrecto.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "InvalidMethod",
    "dni": "12345678F",
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [N] 6. Crear alumno con nombre vacío.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": null,
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "password"
}
###
# [N] 7. Crear alumno con apellidos vacío.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": "Daniel",

```

```

    "surname": null,
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [N] 8. Crear alumno con email vacío.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": null,
    "password": "daniel"
}
###
# [N] 9. Crear alumno con DNI vacío.
POST {{BASE}}/register
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": null,
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [P] 10. Actualizar alumno con datos correctos.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678Z",
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [N] 11. Actualizar alumno con el mismo DNI que otro.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json

```

```

{
  "accessMethod": "Selectividad",
  "dni": "12345678B",
  "firstName": "Daniel",
  "surname": "Ayala",
  "birthDate": "2000-01-01",
  "email": "daniel@alum.us.es",
  "password": "daniel"
}
###
# [N] 12. Actualizar alumno con el mismo email que otro.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "accessMethod": "Selectividad",
  "dni": "12345678W",
  "firstName": "Daniel",
  "surname": "Ayala",
  "birthDate": "2000-01-01",
  "email": "rafael@alum.us.es",
  "password": "daniel"
}
###
# [N] 13. Actualizar alumno con formato de fecha de nacimiento incorrecto.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "accessMethod": "Selectividad",
  "dni": "12345678F",
  "firstName": "Daniel",
  "surname": "Ayala",
  "birthDate": "2000-01-32",
  "email": "daniel.ayala@example.com",
  "password": "daniel"
}
###
# [N] 14. Actualizar alumno con método de acceso incorrecto.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
  "accessMethod": "InvalidMethod",
  "dni": "12345678F",
  "firstName": "Daniel",
  "surname": "Ayala",
  "birthDate": "2000-01-01",
  "email": "daniel.ayala@example.com",
  "password": "daniel"
}

```

```

}
###
# [N] 15. Actualizar alumno con nombre vacío.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": null,
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": ""
}
###
# [N] 16. Actualizar alumno con apellidos vacío.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": "Daniel",
    "surname": null,
    "birthDate": "2000-01-01",
    "email": "daniel.ayala@example.com",
    "password": "daniel"
}
###
# [N] 17. Actualizar alumno con email vacío.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "accessMethod": "Selectividad",
    "dni": "12345678F",
    "firstName": "Daniel",
    "surname": "Ayala",
    "birthDate": "2000-01-01",
    "email": null,
    "password": "daniel"
}
###
# [N] 18. Actualizar alumno con DNI vacío.
PUT {{BASE}}/students/1
Token: {{login.response.body.sessionToken}}
Content-Type: application/json
{
    "accessMethod": "Selectividad",

```

```
"dni": null,  
"firstName": "Daniel",  
"surname": "Ayala",  
"birthDate": "2000-01-01",  
"email": "daniel.ayala@example.com",  
"password": "daniel"  
}  
###  
# [P] 19. Borrar alumno.  
DELETE {{BASE}}/students/3  
Token: {{login.response.body.sessionToken}}  
###  
# [N] 20. Borrar alumno que no existe (no lanza excepción).  
DELETE {{BASE}}/students/999  
Token: {{login.response.body.sessionToken}}  
###  
# [N] 21. Borrar alumno que tiene relaciones.  
DELETE {{BASE}}/students/1  
Token: {{login.response.body.sessionToken}}
```

11.4. Pruebas de aceptación adicionales

Como ejercicio se propone diseñar e implementar las pruebas de aceptación de las tablas:

- Grupos (Groups).
- Grupos-alumnos (GroupsStudents).
- Notas (Grades).

Info

Versión PDF disponible