



# **BENEMERITA UNIVERSIDAD AUTÓNOMA DE PUEBLA**

---

## **Facultad de Ciencias de la Computación**

### **Modelos de desarrollo web**

#### **Actividad 2**

Alumna: Liseth Danae Mecatl Toxcoyoa

Matricula: 202233510

Fecha:19-jun-2025

# Endpoints PATH

## GET por Carrera y Género

```
@app.get("/alumnos/carrera/{carrera}/genero/{genero}")
```

```
async def get_alumnos_por_carrera_y_genero(carrera: str, genero: str):
```

```
    alumnos_filtrados = filter(
```

```
        lambda alumno: (carrera.lower() in alumno.Carrera.lower()) and
```

```
        (alumno.Genero.lower() == genero.lower()),
```

```
        alumnos
```

```
    )
```

```
    return list(alumnos_filtrados)
```

The screenshot displays a web interface for testing API endpoints. At the top, there are two input fields: 'carrera' (required, string, path) with the value 'medicina' and 'genero' (required, string, path) with the value 'masculino'. Below these fields are 'Execute' and 'Clear' buttons. The 'Responses' section shows the results of a test request. It includes a 'Curl' command, the 'Request URL' (http://127.0.0.1:8000/alumnos/carrera/medicina/genero/masculino), and the 'Server response' (200). The 'Response body' is a JSON object containing student information.

```
curl -X 'GET' \
'http://127.0.0.1:8000/alumnos/carrera/medicina/genero/masculino' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/alumnos/carrera/medicina/genero/masculino
```

Server response

Code	Details
200	<pre>{   "Nombre": "Pablo Eduardo Rojas Martinez",   "Matricula": "202156865",   "Edad": 22,   "Carrera": "Medicina",   "Genero": "Masculino",   "Deporte": "volleyball",   "Pelicula": "Intensamente",   "Hobby": "Jugar videojuegos",   "Color": "verde",   "Musica": "pop en ingles",   "Facultad": "Facultad de Medicina" }</pre>

## GET por Facultad y Deporte

```
@app.get("/alumnos/facultad/{facultad}/deporte/{deporte}")
```

```
async def get_alumnos_por_facultad_y_deporte(facultad: str, deporte: str):
```

```
alumnos_filtrados = filter(
    lambda alumno: (facultad.lower() in alumno.Facultad.lower()) and
    (deporte.lower() in alumno.Deporte.lower()),
    alumnos
)
return list(alumnos_filtrados)
```

### GET por Matrícula

```
@app.get("/alumnos/matricula/{matricula}")
async def get_alumno_por_matricula(matricula: str):
    alumnos_filtrados = filter(
        lambda alumno: alumno.Matricula == matricula, alumnos)
    try:
        return list(alumnos_filtrados)[0]
    except:
        return {"error": "No se ha encontrado el alumno con esa matrícula"}
```

### GET por Nombre (búsqueda exacta)

```
@app.get("/alumnos/nombre/{nombre}")
async def get_alumno_por_nombre(nombre: str):
    alumnos_filtrados = filter(
        lambda alumno: alumno.Nombre.lower() == nombre.lower(), alumnos)
    try:
        return list(alumnos_filtrados)[0]
    except:
        return {"error": "No se ha encontrado el alumno con ese nombre exacto"}
```

## GET por Carrera

```
@app.get("/alumnos/carrera/{carrera}")

async def get_alumnos_por_carrera(carrera: str):

    alumnos_filtrados = filter(

        lambda alumno: carrera.lower() in alumno.Carrera.lower(), alumnos)

    return list(alumnos_filtrados)
```

## GET por Edad

```
@app.get("/alumnos/edad/{edad}")

async def get_alumnos_por_edad(edad: int):

    alumnos_filtrados = filter(lambda alumno: alumno.Edad == edad, alumnos)

    return list(alumnos_filtrados)
```

# Endpoints query

## GET con filtros combinados por query parameters

```
@app.get("/alumnos/")

async def filtrar_alumnos(

    edad: Optional[int] = None,

    carrera: Optional[str] = None,

    genero: Optional[str] = None,

    facultad: Optional[str] = None

):

    alumnos_filtrados = alumnos
```

```
if edad:
    alumnos_filtrados = filter(
        lambda alumno: alumno.Edad == edad, alumnos_filtrados)
if carrera:
    alumnos_filtrados = filter(lambda alumno: carrera.lower(
        ) in alumno.Carrera.lower(), alumnos_filtrados)
if genero:
    alumnos_filtrados = filter(
        lambda alumno: alumno.Genero.lower() == genero.lower(), alumnos_filtrados)
if facultad:
    alumnos_filtrados = filter(lambda alumno: facultad.lower(
        ) in alumno.Facultad.lower(), alumnos_filtrados)

return list(alumnos_filtrados)
```

```
← → ↻ ⓘ 127.0.0.1:8000/alumnos/?edad=21
🗎 | AGA Elipse: definición, ec... 🖼 Subir imágenes y da... 📄 Listas CSS - C
Pretty-print ☒
[
  {
    "Nombre": "Maximiliano Aranda León",
    "Matricula": "202213246",
    "Edad": 21,
    "Carrera": "Ingeniería en tecnologías de la información",
    "Genero": "Masculino",
    "Deporte": "Tennis",
    "Película": "Mad Max: Fury Road",
    "Hobby": "Dibujar",
    "Color": "Blanco",
    "Musica": "Rock",
    "Facultad": "Facultad de Ciencias de la Computación"
  },
  {
    "Nombre": "Miguel Angel Castillo Corona",
    "Matricula": "202221447",
    "Edad": 21,
    "Carrera": "Ingeniería en Tecnologías de la Información",
    "Genero": "Masculino",
    "Deporte": "Fútbol",
    "Película": "Atrapame si puedes",
    "Hobby": "Escuchar música",
    "Color": "Rosa",
    "Musica": "Baladas en español",
    "Facultad": "Facultad de Ciencias de la Computación"
  },
  {
    "Nombre": "Mario Alberto Navarro Soto",
    "Matricula": "202264602",
    "Edad": 21,
    "Carrera": "Ingeniería en Ciencias de la Computación",
    "Genero": "Masculino",
    "Deporte": "Boxeo",
    "Película": "Chronicle",
    "Hobby": "Videojuegos",
    "Color": "Verde",
    "Musica": "Clásica",
    "Facultad": "Facultad de Ciencias de la Computación"
  },
  {
    "Nombre": "Javier Mendieta Pérez",
    "Matricula": "202263837",
    "Edad": 21,
    "Carrera": "Ingeniería en Ciencias de la Computación",
    "Genero": "Masculino",
    "Deporte": "Atletismo",
    "Película": "Pacific Rim",
    "Hobby": "Videojuegos",
    "Color": "Verde",
    "Musica": "Rock/Pop",
    "Facultad": "Facultad de Ciencias de la Computación"
  },
  {
    "Nombre": "Elías Romero Morales",
    "Matricula": "202157519",
    "Edad": 21,
```

GET por deporte y música

```
@app.get("/alumnos/intereses/")
```

```
async def alumnos_por_intereses(
```

```
    deporte: Optional[str] = None,
```

```
    musica: Optional[str] = None
```

```

):

alumnos_filtrados = alumnos

if deporte:
    alumnos_filtrados = filter(lambda alumno: deporte.lower(
        ) in alumno.Deporte.lower(), alumnos_filtrados)
if musica:
    alumnos_filtrados = filter(lambda alumno: musica.lower(
        ) in alumno.Musica.lower(), alumnos_filtrados)

result = list(alumnos_filtrados)
return result if result else {"message": "No se encontraron alumnos con esos criterios"}

```

## GET por rango de edades

```

@app.get("/alumnos/rango-edad/")
async def alumnos_por_rango_edad(
    edad_min: int,
    edad_max: int
):
    alumnos_filtrados = filter(
        lambda alumno: edad_min <= alumno.Edad <= edad_max,
        alumnos
    )
    return list(alumnos_filtrados)

```

# GET por hobby y color favorito

```

@app.get("/alumnos/hobbies/")
async def alumnos_por_hobby_color(

```

```

hobby: Optional[str] = None,
color: Optional[str] = None
):
    alumnos_filtrados = alumnos

    if hobby:
        alumnos_filtrados = filter(
            lambda alumno: hobby.lower() in alumno.Hobby.lower(), alumnos_filtrados)

    if color:
        alumnos_filtrados = filter(
            lambda alumno: color.lower() in alumno.Color.lower(), alumnos_filtrados)

    return list(alumnos_filtrados)

```

## GET por película favorita y facultad

```

@app.get("/alumnos/cine/")
async def alumnos_por_pelicula_facultad(
    pelicula: str,
    facultad: Optional[str] = None
):
    alumnos_filtrados = filter(
        lambda alumno: pelicula.lower() in alumno.Pelicula.lower(),
        alumnos
    )

    if facultad:
        alumnos_filtrados = filter(
            lambda alumno: facultad.lower() in alumno.Facultad.lower(),
            alumnos_filtrados

```



)

result = list(alumnos\_filtrados)

return result if result else {"message": "No se encontraron coincidencias"}