UNIVERSIDAD MESOAMERICANA
TÉCNICO EN DESARROLLO DE SOFTWARE
ESTRUCTURAS LÓGICAS II

Simulador de Autómatas Finitos

Descripción General

Desarrollar una aplicación en Python que procese archivos JSON conteniendo definiciones de autómatas finitos, genere diagramas de transición de estados y valide cadenas de entrada.

Objetivos de Aprendizaje

- Aplicar principios de Programación Orientada a Objetos (POO)
- Implementar algoritmos recursivos para procesamiento de autómatas
- Manejar excepciones y validación de datos
- Crear APIs REST con Flask
- Generar visualizaciones con Graphviz
- Aplicar mejores prácticas de programación

Tecnologías Requeridas

- **Python**
- **Flask** para la API REST
- **Graphviz** para generación de diagramas
- **JSON** para estructura de datos
- **Unittest** para pruebas

Estructura del JSON de Entrada

El archivo JSON debe contener una lista de autómatas con la siguiente estructura:


```
[
  {
    "id": "automata_1",
    "name": "Reconocedor de números pares",
    "initial_state": "q0",
    "acceptance_states": ["q0"],
    "alphabet": ["0", "1"],
    "states": ["q0", "q1"],
    "transitions": [
      { "from_state": "q0", "symbol": "0", "to_state": "q0" },
      { "from_state": "q0", "symbol": "1", "to_state": "q1" },
      { "from_state": "q1", "symbol": "0", "to_state": "q0" },
      { "from_state": "q1", "symbol": "1", "to_state": "q1" }
    ],
    "test_strings": ["0", "10", "101", "1010", ""]
  },
  {
    "id": "automata_2",
    "name": "Reconocedor de números pares",
    "initial_state": "q0",
    "acceptance_states": ["q0"],
    "alphabet": ["0", "1"],
    "states": ["q0", "q1"],
    "transitions": [
      { "from_state": "q0", "symbol": "0", "to_state": "q0" },
      { "from_state": "q0", "symbol": "1", "to_state": "q1" },
      { "from_state": "q1", "symbol": "0", "to_state": "q0" },
      { "from_state": "q1", "symbol": "1", "to_state": "q1" }
    ],
    "test_strings": ["0", "10", "101", "1010", ""]
  }
]
```

Requerimientos Funcionales

1. Validaciones Requeridas

- **Estados válidos**: Todos los estados referenciados deben existir
- **Alfabeto consistente**: Símbolos en transiciones deben estar en el alfabeto
- **Estado inicial válido**: Debe existir en la lista de estados
- **Estados de aceptación válidos**: Deben existir en la lista de estados
- **Transiciones completas**: Verificar que no haya estados sin transiciones definidas
- **Valores nulos**: Validar campos obligatorios
- **Tipos de datos**: Verificar tipos correctos (strings, arrays)

2. Procesamiento Recursivo

3. API REST

Endpoint único: POST /process-automata

- Recibe archivo JSON con definiciones de autómatas
- Procesa cada autómata
- Genera diagramas de transición
- Retorna resultados de validación y procesamiento

4. Generación de Diagramas

- Usar Graphviz para crear diagramas de transición
- Guardar imágenes en carpeta generated_diagrams/
- Formato: automata_{id}_{timestamp}.png
- Estados de aceptación con doble círculo
- Estado inicial con flecha entrante

5. Manejo de excepciones

- Los errores y excepciones que puedan generarse deber ser específicos, ejemplos:
    - No hay estados de aceptación en el autómata
    - El carácter x o y no esta definido en el alfabeto del autómata
    - Los estados del autómata no están definidos
    - No hay estado inicial definido para el autómata

6. Respuesta de procesamiento

- Al finalizar el procesamiento se deberá devolver como respuesta la siguiente estructura:

```
[
  {
    "id": "automata_1",
    "success": false,
```

```
    "error_description": "No hay estado inicial definido para el autómata"
  },
  {
    "id": "automata_2",
    "success": true,
    "inputs_validation": [
      {
        "input": "00",
        "result": false
      },
      {
        "input": "001",
        "result": true
      }
    ]
  }
]
```

Entregables

1. **README.md** con instrucciones de instalación y uso
2. **Pruebas unitarias** para las clases principales
3. **Documentación de API** (puede ser en README)
4. **Archivo JSON de ejemplo** con al menos 2 autómatas diferentes
5. **Repositorio de git con todo el código fuente**
    - Considere que el repositorio debe tener al menos 7 días de trabajo; esto se refiere número de días donde al menos debe existir un commit, caso contrario se penalizará con 5 puntos menos por día faltante
    - Los mensajes de cada commit debe ser claros y acordes al cambio que se ha realizado, ejemplos:
        1. Commit inicial de la creación del repositorio
        2. Se define la clase con el endpoint solicitado
        3. Se realiza la integración de graphviz
        4. Se realiza la definición de la clase x que se utilizará para…
        5. etc

Fecha de Entrega: 13/08/2025 - 23:59 hrs

Notas importantes

- El repositorio deberá nombrarse: **finite-automata-simulator_firstName_surname**
- El proyecto debe ejecutarse sin errores
- Incluir instrucciones claras de instalación
- El código debe estar en **inglés** consistentemente
- Validar el proyecto con los JSON de ejemplo antes de entregar
- **Si se detecta que la solución total o parcial de la práctica ha sido generado con IA, la misma será anulada sin excepción.**