TCT-시스템&솔루션개발
실기형 문제지

[2023년 3차]

사번		성명	
유의 사항	1. 공정한 평가를 위해 동료를 도와주는 행위, 보여주는 행위를 금지하고 있습니다. 2. 부정행위 적발 시, 응시한 평가는 0점 처리됩니다. 3. 본 시험지는 응시장 외부로 유출할 수 없으며, 시험 종료 후 감독관에게 제출해야 합니다.		

LG CNS

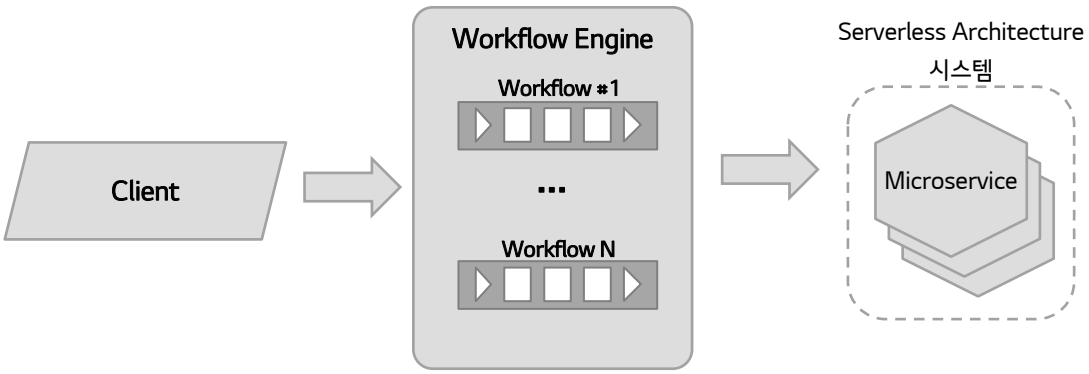
Workflow Engine

개요

해당 시스템 구현은 동해 요구사항 분석, 데이터 구조화, HTTP Server/Client 구현 등의 기술역량 및 프로그램 구현 역량을 측정하기 위한 문제입니다.

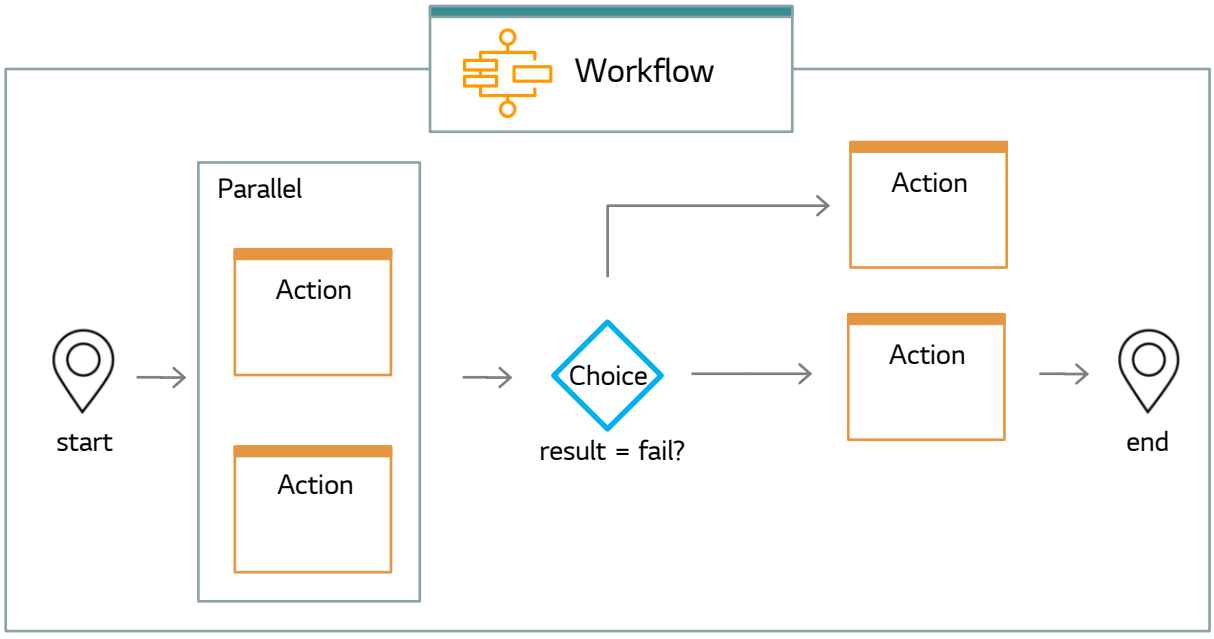
설명

본 프로그램은 Microservice를 활용한 Serverless Architecture로 구성된 시스템을 대상으로 Client의 요청에 따라 정해진 Workflow를 수행하는 Workflow Engine이다. 시스템은 Action을 수행하는 여러 개의 Microservice로 구성된다. Workflow는 State별 처리할 내용과 전이할 State가 정의된 State Machine 구조이다. State 유형이 Action이면 제공되는 Microservice를 호출하여 처리한다.



[기능 요약]

- Workflow(업무흐름)은 State와 State transition으로 표현된다
  - ◆ Action state는 실제 업무를 수행하는 Microservice의 호출 URL, 파라미터 등을 정의한다
  - ◆ Choice state는 업무 진행 흐름을 제어하기 위한 조건을 정의한다
  - ◆ Parallel state는 동시에 여러 업무 진행을 병렬로 수행하는 흐름을 정의한다
- Workflow Engine 솔루션은 Client의 업무 요청에 따라 정의된 State와 State transition을 수행한다.

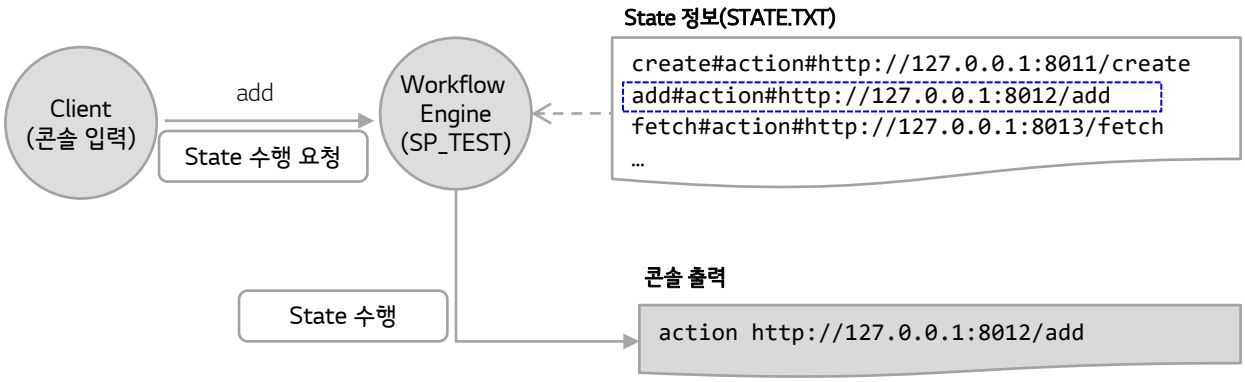


문제

아래 제시된 문항은 문항번호가 증가할수록 점진적 개선을 요구하는 방식으로 구성되어 있으며, 제시된 문항번호 별로 각각 **구현된 소스와 컴파일 된 실행파일을 제출**하시오.
cf) 1번 구현 → 1번 소스복사 → 2번 구현 → 2번 소스복사 → ...

1. 콘솔 입력을 통해서 'State Name'을 입력하면 해당 "action" 타입 State를 수행(Type과 Url 출력)하는 Workflow Engine을 구현하시오. (20점)

상세설명



※ "action" 타입 State 수행 ('※ 콘솔 입/출력' 참조)
- 콘솔을 통해 'State Name'을 입력 받음
- "action"타입 State 정보 파일의 State Name에 해당하는 State 수행 정보(Type과 Url)를 출력 ('※ "action"타입 State 정보 파일 형식정보' 참조)

형식정보

※ "action"타입 State 정보 파일 형식정보
- 파일명 : STATE.TXT (각 소문항 홈 아래)
- 파일 형식 : <State Name> + '#' + <State Type> + '#' + <Url>

```
create#action#http://127.0.0.1:8011/create
add#action#http://127.0.0.1:8012/add
fetch#action#http://127.0.0.1:8013/fetch
```

- <State Name> : State 명
- <State Type> : State의 종류, 특정 Microservice를 호출하는 State는 "action"
- <Url> : 호출할 Microservice의 URL 주소

※ 콘솔 입/출력
- 입력 포맷 : <State Name>
- 출력 포맷 : <State Type> + ' ' + <Url>

```
create<엔터키>                    ← 콘솔 입력
action http://127.0.0.1:8011/create  ← 콘솔 출력
add<엔터키>                       ← 콘솔 입력
action http://127.0.0.1:8012/add     ← 콘솔 출력
fetch<엔터키>                     ← 콘솔 입력
action http://127.0.0.1:8013/fetch   ← 콘솔 출력
```

평가대상

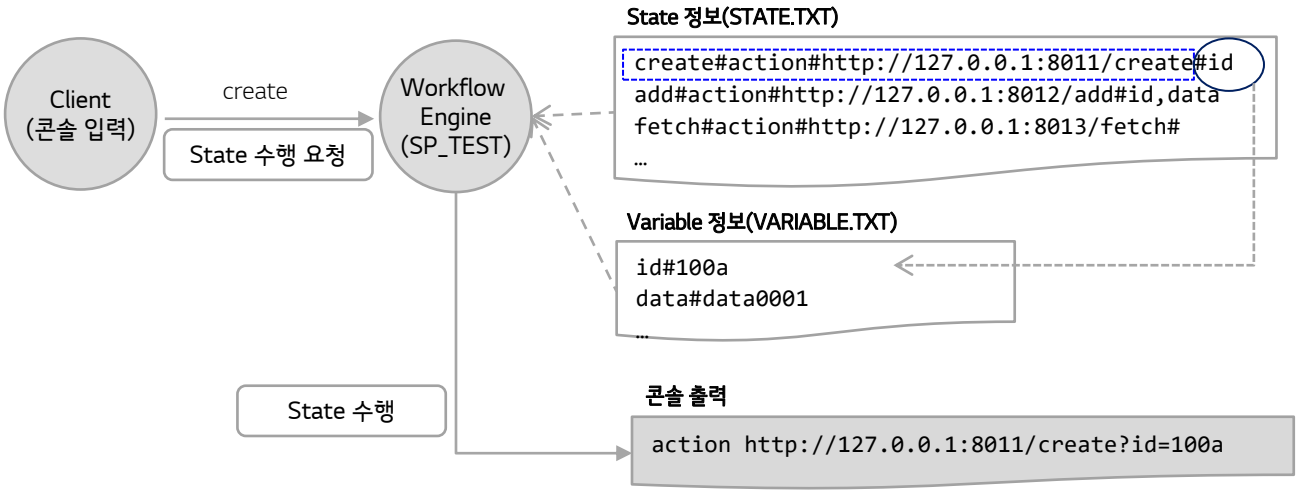
프로그램 정상 실행, 콘솔 입/출력 결과

문제

2. 위 1번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 Workflow Engine을 구현하시오. (15점)

- Workflow Engine에서 변수를 관리할 수 있도록 Variable 정보 추가
- Microservice 호출 시 Parameter를 전달할 수 있으며 전달할 변수 목록이 "action" 타입 State에 추가
- 콘솔 입력은 통해서 'State Name'을 입력하면 해당 State의 Type과 전달할 Parameter를 포함한 Url 정보를 출력

상세설명



※ Variable 정보 ('※ Variable 정보 파일 형식정보' 참조)
- Workflow Engine이 **전역으로 관리**할 변수 정보
- Microservice 호출 시에 Parameter 목록의 변수 값은 설정할 수 있음

※ "action" 타입 State 수행 ('※ 콘솔 입/출력' 참조)
- 콘솔을 통해 'State Name'을 입력 받음
- 'State 정보 파일'의 State Name에 해당하는 State 수행 정보(Type과 **Parameter 포함된 Url**)를 출력
  ('※ "action"타입 State 정보 파일 형식정보' 참조)
- Parameter 포함된 Url : Parameter 목록의 변수와 해당 변수 값으로 생성한 Query String이 포함된 Url

형식정보

※ Variable 정보 파일 형식정보
- 파일명 : VARIABLE.TXT (각 소문항 홈 아래)
- 파일 형식 : <Variable Name> + '#' + <Variable Value>

```
id#100a
data#data0001
```

- <Variable Name> : 변수 명
- <Variable Value> : 변수가 가진 값(문자열)

형식정보 (계속)

※ 'State 정보 파일' 형식정보

- 파일명 : STATE.TXT (각 소문항 홈 아래)
- 파일 형식 : <State Name> + '#' + <State Type> + '#' + <Url> + '#' + <Variable Name> + ',' + ... + ',' + <Variable Name>

```
create#action#http://127.0.0.1:8011/create#id
add#action#http://127.0.0.1:8012/add#id,data
fetch#action#http://127.0.0.1:8013/fetch#          ← Parameter가 없을 수 있음
```

- <State Name> : State 명
- <State Type> : State의 종류, Microservice를 호출하는 State는 "action"
- <Url> : 호출할 Microservice의 URL
- Parameter 목록 : "action" 타입 State가 URL을 호출할 때 GET방식으로 전달할 변수 목록

※ 전달할 parameter가 없는 경우도 있음

※ 콘솔 입/출력

- 입력 포맷 : <State Name>
- 출력 포맷 : <State Type> + ' ' + <Url> + Query String 형식(Parameter 존재하는 경우)

```
create<엔터키>                                          ← 콘솔 입력
action http://127.0.0.1:8011/create?id=100a            ← 콘솔 출력
add<엔터키>                                             ← 콘솔 입력
action http://127.0.0.1:8012/add?id=100a&data=data0001 ← 콘솔 출력
fetch<엔터키>                                           ← 콘솔 입력
action http://127.0.0.1:8013/fetch                     ← 콘솔 출력
```

Parameter가 없는 경우

※ Query String이란?

- 입력데이터를 넘기기 위해 Url 주소에 Parameter 정보를 추가하는 방식
- Url 뒤에 '?' 를 붙이고 <Variable Name> + '=' + <Variable Value> 형식으로 구성
- 여러 개의 Parameter가 전달되는 경우 '&'를 사용

평가대상

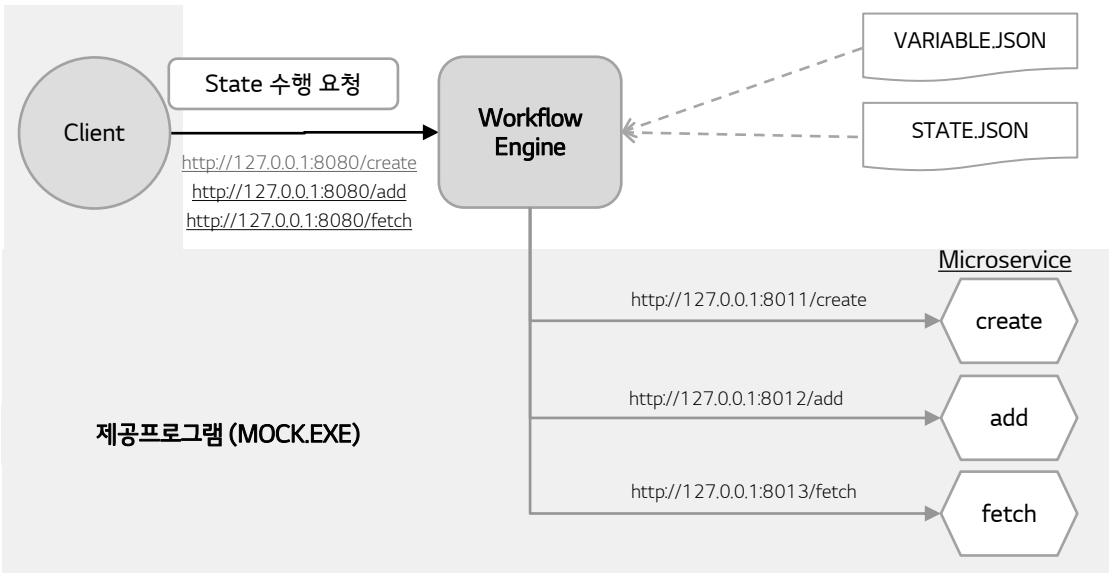
프로그램 정상 실행, 콘솔 입/출력 결과

문제

3. 위 2번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 Workflow Engine을 구현하시오. (15점)

- State 정보 및 Variable 정보가 Json 포맷으로 제공됨
- "action"타입 State 수행을 위한 Microservice가 제공됨
- HTTP 통신으로 State 수행 요청을 수신하면 HTTP 통신으로 Microservice를 호출하고 응답에 따라 **변수의 값을 업데이트 (다른 State 수행 요청에 업데이트된 값을 사용해야 함)**

상세설명



※ "action" 타입 State 수행 (변경)

- HTTP 통신으로 URL 호출 (Get 방식)
  ex) Url이 "http://127.0.0.1:8011/create"
  Parameter가 "id"이고, id 변수의 값이 "100a"인 경우
  * HTTP Method : GET
  * 호출 URL : http://127.0.0.1:8011/create?id=100a
- HTTP 응답에 따라 Workflow Engine이 관리하는 **변수의 값을 업데이트**
  ex) HTTP 응답이 "{"total":"1"}" 일 경우 total 변수의 값은 "1"로 업데이트
  HTTP 응답이 "{"key":"1a2b3c4d"}" 일 경우 key 변수의 값은 "1a2b3c4d"로 업데이트
- 업데이트된 변수의 값은 다른 State 수행시 사용

형식정보

※ 'Variable 정보 파일' 형식정보 (변경)

- 파일명 : VARIABLE.JSON (각 소문항 폼 아래)
- Workflow Engine의 변수의 값을 업데이트하더라도 본 파일의 내용은 업데이트하지 않음
- 데이터 포맷 : 다음과 같은 Json 형식.
  { <Variable Name>:<Variable Value>, ... , <Variable Name>:<Variable Value> }

Variable 정보 파일 예시

```
{
  "id": "100a",
  "key": "none",
  "data": "data0001",
  "total": "0"
}
```

형식정보

※ "action" 타입 State 형식정보(변경)

- 아래의 Json 형식

```
{
  “type”:“action”,
  “url”:<Url>,
  “parameters”: [<Variable Name>,…,<Variable Name>],
}
```

- type : <State Type>, Microservice를 호출하는 State는 "action"
- url : <Url> - 호출할 Microservice의 URL
- parameters : <Variable Name>배열 – Microservice를 호출할 때 GET방식으로 전달할 parameter 목록
  전달할 매개변수가 없는 경우, 빈 array
- 아래의 예시는 `"http://127.0.0.1:8011/create"` URL에 파라미터로 `"id"`를 추가하여 호출하는 State임. ※ **응답에 따라 Workflow Engine이 관리하는 변수의 값을 업데이트**

"action" 타입 State 예시

```
{
  "type": "action",
  "url": "http://127.0.0.1:8011/create",
  "parameters": [ "id" ]
}
```

※ 'State 정보 파일' 형식정보

- 파일명 : STATE.JSON (각 소문항 홈 아래)
- 데이터 포맷 : 다음과 같은 Json 형식
  { "state": { <State Name>:<State 정보>, … , <State Name>:<State 정보> } }

State 정보 파일 예시

```
{
  "state":
  {
    "create": {
      "type": "action",
      "url": "http://127.0.0.1:8011/create",
      "parameters": [ "id" ]
    },
    "add": {
      "type": "action",
      "url": "http://127.0.0.1:8012/add",
      "parameters": [ "id", "key", "data" ]
    },
    "fetch": {
      "type": "action",
      "url": "http://127.0.0.1:8013/fetch",
      "parameters": []
    }
  }
}
```

형식정보

※ State 수행 요청

- URI : GET http://127.0.0.1:8080/<State Name>
- 요청 Body : 없음
- 동작 : 'State 정보'에 정의된 Microservice에 Parameter와 함께 URL을 호출한 후
  응답에 따라 변수의 값을 업데이트, 다른 State 수행 요청에 업데이트된 값을 사용해야 함
- 응답 Status Code : 200 OK
- 응답 Body : 없음

※ Microservice 호출 ("action" 타입 State 수행 요청)

- URI : GET <Url> + ? + <Query String>
  ex) GET http://127.0.0.1:8011/create?id=100a
  ex) GET http://127.0.0.1:8012/add?id=100a&key=eH7bDVXX&data=data0001
  ex) GET http://127.0.0.1:8013/fetch

업데이트된 변수 값이 사용됨

- 요청 Body : 없음
- 동작 : "action" 타입 State를 수행 후 결과 응답
- 응답 Body : Variable의 값을 JSON 형식으로 응답, **응답에 따라 변수를 업데이트해야 함**
  ex) { "key" : "eH7bDVXX" } → key 변수의 값을 "eH7bDVXX"로 업데이트해야 함
  ex) { "total" : "1" } → total 변수의 값을 "1"로 업데이트해야 함

※ 제공프로그램(MOCK.EXE)

- MOCK.EXE를 콘솔에서 실행하면 Microservice가 기동 되며, 테스트 시나리오에 따라 Workflow Engine의 기능을 순차적으로 테스트함
- MOCK.EXE는 자가 검수의 기능도 수행하며, 모든 테스트 시나리오 성공 시 다음의 문구가 콘솔에 출력

```
C:\>MOCK.EXE<엔터키>          ← 제공프로그램 실행
...                           ← 테스트 시나리오에 따른 테스트 실행
테스트에 성공했습니다!        ← 소문항의 모든 테스트시나리오 성공
```

평가대상

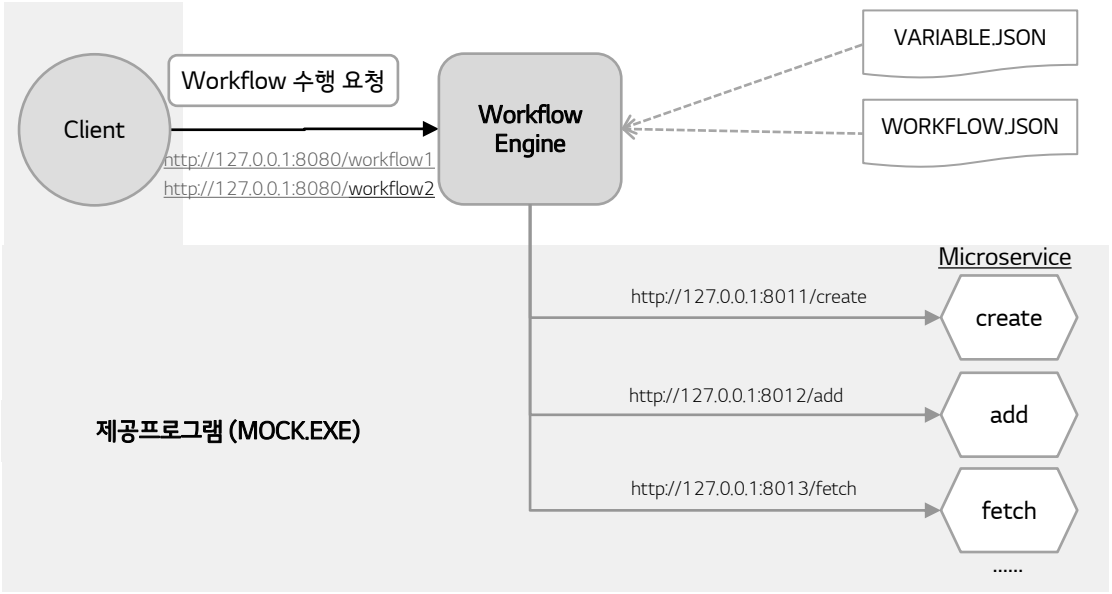
프로그램 정상 실행, Microservice 요청(HTTP 요청), HTTP 응답 결과

문제

4. 위 3번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 Workflow Engine를 구현하시오. (20점)

- 여러 개의 State를 업무 흐름에 따라 수행할 수 있는 Workflow 수행 기능 추가
- State 정보가 **Workflow 정보에 통합**되며, State에 'next' 속성이 추가
  (※ 3번 문항의 **State 정보파일 제거**되고 Workflow 정보에 통합됨)
- HTTP 통신으로 Workflow 수행 요청을 수신하면 Workflow를 수행하고 결과를 응답
  (※ 3번 문항의 **State 수행 기능은 제거**되고 Workflow 수행에 통합됨)
- 병렬로 동시에 여러 업무를 수행할 수 있는 "parallel" type State 추가

상세설명



※ Workflow 수행

- Workflow : 여러 개의 State를 정의된 transition에 따라 수행하는 업무 흐름
- Client 요청 시 "startFrom" 속성에 지정된 State 부터 수행
- State 수행 후 해당 State의 "next" 속성의 State를 순차적으로 수행
- "next" 속성이 "end"일 경우 Workflow가 종료됨
- 완료 후 Client에 "responses" 속성 내 변수의 값을 응답

※ "parallel" 타입 State 수행

- "parallel" 타입 State : 동시에 여러 개의 Workflow흐름을 진행하는 State
- "branches" 속성에 포함된 여러 개의 Workflow를 동시에 병렬로 작업을 시작하여 가장 느린 Workflow가 종료될 때까지 대기
- "branches"에 포함된 모든 Workflow가 종료되면 "next" 속성의 State로 진행

형식정보

※ "action" 타입 State 형식정보(변경, "next" 추가됨)

- 아래의 Json 형식

```
{
  “type”:“action”,
  “url”:<Url>,
  “parameters”: [<Variable Name>,…,<Variable Name>],
  “next”:<다음 State의 ‘State Name’>
}
```

- type, url, parameters : [3번 문항과 동일]
- next : 다음에 수행할 State의 <State Name>, "end"일 경우 Workflow 종료
- 아래의 예시는 **"http://127.0.0.1:8011/create"** URL에 파라미터로 **"id"**를 추가하여 호출하여 응답에 따라 **변수의 값을 업데이트하고 "add" State로 진행**

"action" 타입 State 예시

```
{
  "type": "action",
  "url": "http://127.0.0.1:8011/create",
  "parameters": [ "id" ],
  "next": "add"
}
```

※ Workflow 형식정보

- 아래의 Json 형식

```
{
  “startFrom”:<시작 State의 ‘State Name’>,
  “state”: { <State Name>:<State 정보>, … ,<State Name>:<State 정보> },
  “responses”:[ <Variable Name>, … ,<Variable Name> ]
}
```

- startFrom : Workflow 시작시 수행할 State의 <State Name>
- state : Workflow에 포함된 State 정보
  <State Name> : <State 정보> 의 반복으로 배열이 아님
- responses : <Variable Name> 배열 – Workflow가 Client에 응답할 변수 목록
  응답할 변수가 없는 경우, 빈 array
- Workflow 예시는 ※ Workflow 정보 예시(workflow1), ※ Workflow 정보 예시(workflow2) 참조

※ Workflow 수행 요청

- URI : GET http://127.0.0.1:8080/<Workflow Name>
- 요청 Body : 없음
- 동작 : 'Workflow 정보'에 정의된 Workflow대로 순차적으로 State를 실행 후 결과 응답
- 응답 Body : responses에 정의된 Variable의 값을 JSON 형식으로 응답
  ex) { "id":"100a", "total":"1" }

형식정보

※ "parallel" 타입 State 형식 정보(신규)

- 아래의 Json 형식

```
{
  “type”:“parallel”,
  “branches”: [ <Workflow 정보>,…,<Workflow 정보> ],
  “next”:<다음 State의 ‘State Name’>
}
```

- type : <State Type>, 병렬로 2개 이상의 workflow를 수행하는 State는 "parallel"
- branches : <Workflow 정보> 배열 – 병렬로 동시에 수행할 Workflow 목록
  ※ branches에 포함된 Workflow는 Client에 응답하지 않으므로, responses가 비어 있음
- next : 다음에 수행할 State의 <State Name>, "end"일 경우 Workflow 종료
- 아래의 예시는 **병렬로 수행**할 2개의 branch로 구성.
  - 첫번째 branch는 getdata와 adddata를 수행하는 workflow.
  - 두번째 branch는 addmedia를 수행하는 workflow
  - 2개의 branch의 Workflow가 **모두 종료된 후에 report로 진행**

"parallel" 타입 State 예시

```
{
  "type": "parallel",
  "branches": [
    {
      "startFrom": "getdata",
      "state": {
        "getdata": {
          "type": "action",
          "url": "http://127.0.0.1:9011/getdata",
          "parameters": [ "id" ],
          "next": "adddata"
        },
        "adddata": {
          "type": "action",
          "url": "http://127.0.0.1:9012/adddata",
          "parameters": [ "queueid", "data" ],
          "next": "end"
        }
      },
      "responses": []
    },
    {
      "startFrom": "addmedia",
      "state": {
        "addmedia": {
          "type": "action",
          "url": "http://127.0.0.1:9013/addmedia",
          "parameters": [ "queueid", "mediaid" ],
          "next": "end"
        }
      },
      "responses": []
    }
  ],
  "next": “report"
}
```

첫번째 workflow

두번째 workflow

형식정보

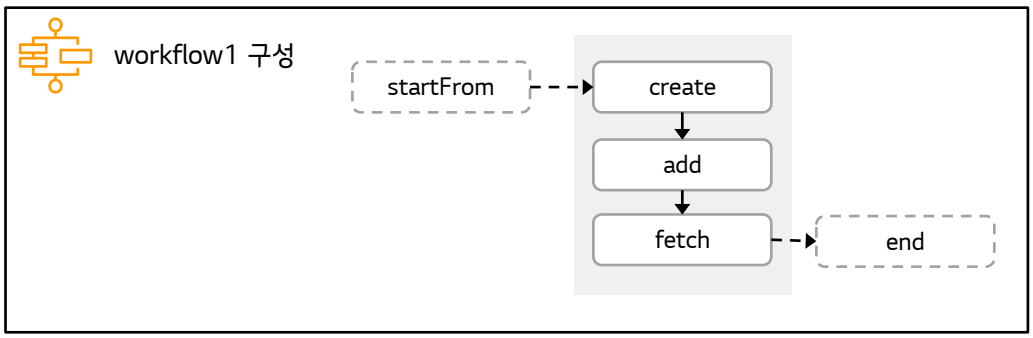
※ 'Workflow 정보 파일' 형식정보

- 파일명 : WORKFLOW.JSON (각 소문항 홈 아래)
- 데이터 포맷 : 다음과 같은 Json 형식.

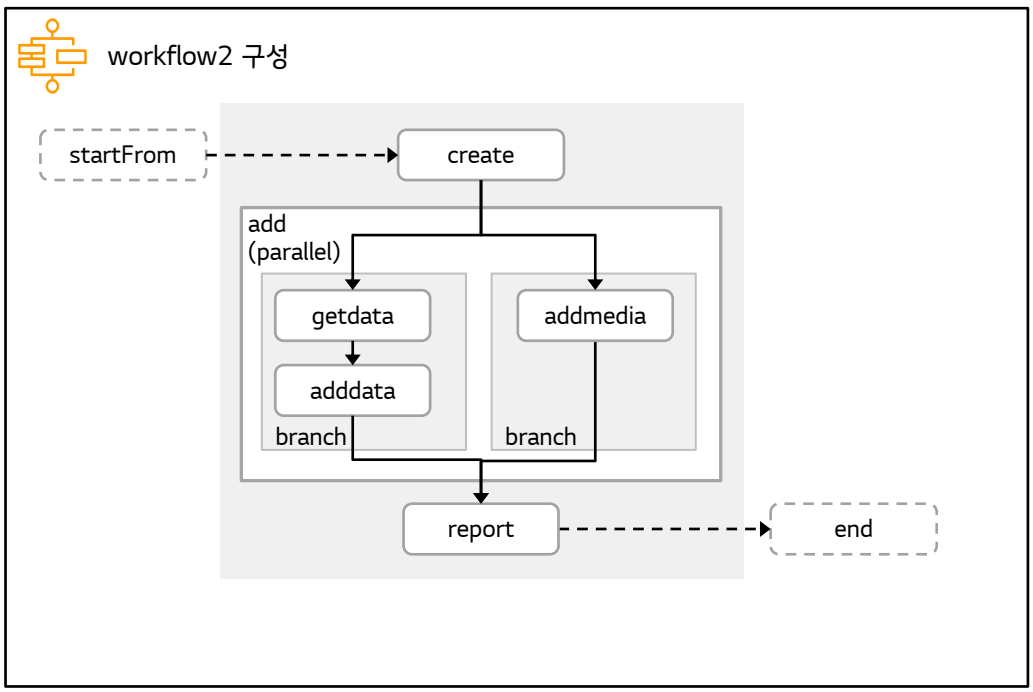
{ "workflow": { <Workflow Name>:<Workflow 정보>, ... , <Workflow Name>:<Workflow 정보> } }

```
{
 "workflow":
  {
    "workflow1": <WORKFLOW.JSON의 workflow1 참조>,
    "workflow2": <WORKFLOW.JSON의 workflow2 참조>
  }
}
```

※ Workflow 정보 예시 (workflow1)



※ Workflow 정보 예시 (workflow2)



평가대상

프로그램 정상 실행, Microservice 요청(HTTP 요청), HTTP 응답 결과

문제

5. 위 4번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 Serverless Workflow를 구현하시오. (10점)

- 조건식에 따라 Workflow를 제어할 수 있는 "choice" type State 추가

상세설명

※ "choice" 타입 State 수행

- "choice" 타입 State : **조건에 따라 Workflow 흐름을 분기**할 수 있는 State
- "choices" 속성에 포함된 여러 개의 조건을 **순서대로 비교**하고, 만족하는 조건의 "next" 속성의 State로 진행
  ※ 순서대로 비교하여 첫번째로 만족하는 조건의 "next"속성으로 진행함
  (if (...) else if (...) else if (...) else (...) 구조와 동일)
- 만족하는 조건이 없을 경우에는 "next" 속성의 State로 진행

형식정보

※ "choice" 타입 State 형식 정보

- 아래의 Json 형식

```
{
  "type":"choice",
  "next":<일치하는 조건이 없을 경우 다음 State의 'State Name'>
  "choices":[
    {
      "variable": <Variable Name>,
      "equal": <Variable Value>,
      "next": <조건 만족시 다음 State의 'State Name'>
    },
    … (<Choice 정보> 반복)
    {
      "variable": <Variable Name>,
      "equal": <Variable Value>,
      "next": <조건 만족시 다음 State의 'State Name'>
    }
  ]
}
```

<Choice 정보>

<Choice 정보>

- type : <State Type>, 조건에 따라 진행 흐름을 제어하는 State는 "choice"
- next : choices 속성에 일치하는 조건이 없을 경우 진행할 다음 State의 <State Name>
  "end"일 경우 Workflow 종료
- choices : <Choice 정보> 배열 – 순서대로 만족하는 조건에 따라 Workflow 진행
- variable(<Choice 정보> 내) : 비교할 변수의 변수 명 (<Variable Name>)
- equal(<Choice 정보> 내) : 비교할 변수의 값
- next(<Choice 정보> 내) : 비교할 변수의 값이 일치(조건 만족)할 때 진행할 State의 <State Name>
  "end"일 경우 Workflow 종료

형식정보

※ 다음 예시는 "result" 변수가 "fail"일 경우 "error" State로 진행하고,
"queuetotal" 변수가 "3"일 경우에 종료하고, 두 조건 모두 만족하지 않을 경우 "add" State로 진행

"choice" 타입 State 예시

```
{
  "type": "choice",
  "choices": [
    {
      "variable": "result",
      "equal": "fail",
      "next": "error"
    },
    {
      "variable": "queuetotal",
      "equal": "3",
      "next": "end"
    }
  ],
  "next": "add"
}
```

<Choice 정보>
result 변수가 fail일 경우 error도 진행

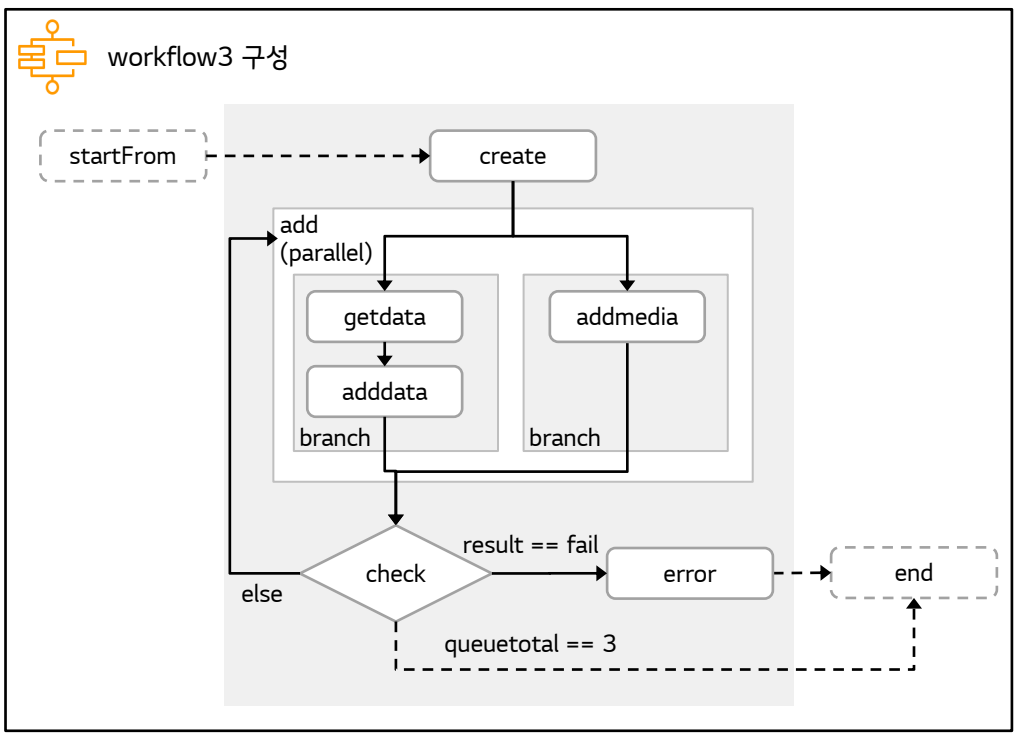
<Choice 정보>
queuetotal 변수가 3일 경우 종료

→ 그 외의 경우 add도 진행

Pseudo code

```
if result == fail :
    next := error
elif queuetotal == 3 :
    next := end
else
    next := add
```

※ Workflow 정보 예시 (workflow3)



평가대상

프로그램 정상 실행, Microservice 요청(HTTP 요청), HTTP 응답 결과

폴더 정보

※ 프로그램 및 파일 위치 정보 (실행위치 기반 상대경로 사용 필수)

- 구현할 프로그램 위치 및 실행 위치 : 각 소문항 홈 (SUB1/SUB2/SUB3/SUB4/SUB5)
- 자가 검수용 참고 파일명 : COMPARE 폴더 내 CMP_CONSOLE.TXT (SUB1/SUB2)
- 제공되는 MOCK 프로그램 파일명 : 각 소문항 홈 아래 MOCK.EXE (SUB3/SUB4/SUB5)

*** 제공되는 파일들은 문항에 따라 다를 수 있음**

실행 방식

※ 구현할 프로그램 형식

- 프로그램 형태 : 콘솔(Console) 프로그램
- 프로그램 파일명 : SP_TEST
- 실행 방식(문항1~2) : 콘솔 실행 → 콘솔 입/출력처리 반복 (종료 없음)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
...
```

- 실행방식(문항3~5) : 콘솔 실행 → 실시간 HTTP 요청 수신 (종료 없음)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
...
```

제공 및 제출

- ✓ 각 언어별 제공파일 압축 해제 후 자동 생성된 폴더 사용 필수
- ✓ 제공되는 주요 내용
  - 샘플 파일
  - 제공 프로그램 실행파일 (MOCK.EXE)
  - 제출시 사용할 문항별 폴더 구조
- ✓ 제출 파일 및 폴더 상세 내용 (각 언어별 실기 가이드 참고)

주의사항

실행 결과로 평가하고 부분점수는 없으므로 아래사항을 필히 주의해야 함

- 구현된 프로그램은 실행 완결성 필수 (명확한 실행&종료 조건 처리, 통상의 실행 시간)
- 소 문항별 결과 검수 필수 (선행문항 오류 시, 후속문항 전체에 오류가 발생할 수 있음)
- 제시된 조건이 없는 한 선행요구사항 유지 필수
- 프로그램 실행 위치 및 실행결과출력 (폴더위치, 파일명, 데이터포맷 등) 정확히 일치
- 제시된 모든 위치는 상대경로 사용 필수 (프로그램 실행 위치 기준)
- 모든 문자는 대소문자 구분 필수

테스트 방법

※ 자가 검수를 위해 제공되는 샘플은 채점용 데이터와 다를 수 있음

검수를 위한 샘플 결과 파일은 각 문항 출력 폴더(COMPARE)에 사전 제공됨

[문항1]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:\>SP_TEST<엔터키>              ← 구현한 프로그램 실행 (Argument 없음)
create<엔터키>                                    ← 콘솔 입력
action http://127.0.0.1:8011/create               ← 콘솔 출력
add<엔터키>                                       ← 콘솔 입력
action http://127.0.0.1:8012/add                  ← 콘솔 출력
fetch<엔터키>                                     ← 콘솔 입력
action http://127.0.0.1:8013/fetch                ← 콘솔 출력
```

[문항2]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:\>SP_TEST<엔터키>              ← 구현한 프로그램 실행 (Argument 없음)
create<엔터키>                                    ← 콘솔 입력
action http://127.0.0.1:8011/create?id=100a       ← 콘솔 출력
add<엔터키>                                       ← 콘솔 입력
action http://127.0.0.1:8012/add?id=100a&data=data0001  ← 콘솔 출력
fetch<엔터키>                                     ← 콘솔 입력
action http://127.0.0.1:8013/fetch                ← 콘솔 출력
```

[문항3]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.EXE의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.EXE<엔터키>
테스트를  시작합니다.
1. State 수행 요청 : create
[http://127.0.0.1:8011/create?id=100a]
{
  "key": "eH7bDVXX"
}
요청 : create     응답 : 없음
------------------------------------------------------------------
2. State 수행 요청 : add
[http://127.0.0.1:8012/add?id=100a&key=eH7bDVXX&data=data0001]
{}
요청 : add        응답 : 없음
------------------------------------------------------------------
3. State 수행 요청 : fetch
[http://127.0.0.1:8013/fetch]
{
  "total": "1"
}
요청 : fetch      응답 : 없음
테스트에 성공했습니다!
```

테스트 방법 (계속)

[문항4]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.EXE의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.EXE<엔터키>
테스트를 시작합니다.
1. workflow1을 테스트합니다.
[http://127.0.0.1:8011/create?id=100a]
{
  "key": "YgdBIIU7"
}
[http://127.0.0.1:8012/add?id=100a&key=YgdBIIU7&data=data0001]
{}
[http://127.0.0.1:8013/fetch]
{
  "total": "1"
}
요청 : workflow1          응답 : {"id": "100a","total": "1"}
---------------------------------------------------------------
2. workflow2를 테스트합니다.
[http://127.0.0.1:9010/create?queuesize=10]
{
  "queueid": "q001",
  "key": "3Oq46zzS"
}
[http://127.0.0.1:9011/getdata?id=100a]
{
  "data": "data for parallel"
}
[http://127.0.0.1:9012/adddata?queueid=q001&data=data for parallel]
{
  "result": "fail",
  "errorcode": "err00a8"
}
[http://127.0.0.1:9013/addmedia?queueid=q001&mediaid=media001]
{
  "queuetotal": "1",
  "mediaid": "media002"
}
[http://127.0.0.1:9014/report?result=fail&errorcode=err00a8&queuetotal=1]
{
  "result": "reported"
}
요청 : workflow2          응답 : {"queueid": "q001","queuetotal":
"1","result": "reported"}, 응답 소요시간 : X 밀리초
테스트에 성공했습니다!
```

테스트 방법 (계속)

[문항5]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.EXE의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.EXE<엔터키>
테스트를 시작합니다.
1. workflow1을 테스트합니다.
… (3번 문항과 동일) …
요청 : workflow1          응답 : {"id": "100a","total": "1"}
-----------------------------------------------------------------
2. workflow3의 실패 케이스를 테스트합니다.
[http://127.0.0.1:9010/create?queuesize=10]
{
  "queueid": "q001",
  "key": "2v7BOzuq"
}
…
[http://127.0.0.1:9014/error?queueid=q001&queuetotal=1&errorcode=err00a8]
{
  "result": "error reported"
}
요청 : workflow3          응답 : {"queueid": "q001","queuetotal":
"1","result": "error reported"}, 응답 소요시간 : X  밀리초
-----------------------------------------------------------------
3. workflow3의 성공 케이스를 테스트합니다.
[http://127.0.0.1:9010/create?queuesize=10]
{
  "queueid": "q002",
  "key": "kXrjJpBn"
}
[http://127.0.0.1:9011/getdata?id=100a]
{
  "data": "data for parallel"
}
[http://127.0.0.1:9012/adddata?queueid=q002&data=data for parallel]
{
  "result": "ok",
  "errorcode": "none"
}
… (add parallel state가 3회 반복됨) …
[http://127.0.0.1:9013/addmedia?queueid=q002&mediaid=media004]
{
  "queuetotal": "3",
  "mediaid": "media005"
}
요청 : workflow3          응답 : {"queueid": "q002","queuetotal":
"3","result": "ok"}, 응답 소요시간 : X 밀리초
테스트에 성공했습니다!
```