Introduction to Web Programming with Flask (Contd.)

November 4, 2023

1 Introduction to Web Programming with Flask (Contd.)

1.1 Exception Handling

It is always important to cover the edge cases in all programs.

[1]:
```
from flask import Flask
from flask import request, make_response

app = Flask(__name__)

class Employee:
    name = ''
    id = ''
    address = ''

    def __init__(self, name, id, address):
        self.name = name
        self.id = id
        self.address = address

    def __str__(self):
        return str({
            "id": self.id,
            "name": self.name,
            "address": self.address
        })

    def validate(self):
        val_name = len(self.name) > 0 and len(self.name) < 256
        val_addr = len(self.address) > 0 and len(self.address) < 1024
        return val_name and val_addr

employees = {
    1: Employee("Prashanth", 1, "Bengaluru"),
    2: Employee("Shiva", 2, "Bengaluru"),
    3: Employee("Phaneendra", 3, "Mysore"),
    4: Employee("Pranav", 4, "Mysore"),
```

```python
}

count = 4

# Exception Handling Classes

class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})

class ValidationError(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "Invalid Input Parameter")


class EmployeeNotPresentError(BaseException):
    def __init__(self) -> None:
        super().__init__(404, "Employee not Present")


@app.route("/api/employee/", methods=["POST"])
def create_employee():
    global count
    employee = request.json
    count += 1
    employee['id'] = count
    emp = Employee(employee['name'], count, employee['address'])

    # Validation error
    if not emp.validate():
        err = ValidationError()
        return str(err), err.status

    employees[count] =  emp
    return employee

@app.route("/api/employee/<int:employee_id>", methods=["PUT"])
def alter_employee_data(employee_id):
    employee = request.json
    employees[employee_id].name = employee.get('name', employees[employee_id].
↪name)
    employees[employee_id].address = employee.get('address',␣
↪employees[employee_id].address)
    emp = Employee(employee['name'], count, employee['address'])

    # Validation error
    if not emp.validate():
        err = ValidationError()
        return str(err), err.status

    return str(employees[employee_id])

@app.route("/api/employee/<int:employee_id>", methods=["GET"])
def get_employee_information(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    return str(emp)

@app.route("/api/employee/<int:employee_id>", methods=["DELETE"])
def remove_employee_data(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    del employees[employee_id]
    return make_response(""), 200
```

1.2 Logging

Helps clearly identify errors in the production system when lot of users access the system

[3]:
```
from flask import Flask
from flask import request, make_response

app = Flask(__name__)

class Employee:
    name = ''
```

```python
    id = ''
    address = ''

    def __init__(self, name, id, address):
        self.name = name
        self.id = id
        self.address = address

    def __str__(self):
        return str({
            "id": self.id,
            "name": self.name,
            "address": self.address
        })

    def validate(self):
        val_name = len(self.name) > 0 and len(self.name) < 256
        val_addr = len(self.address) > 0 and len(self.address) < 1024
        return val_name and val_addr

employees = {
    1: Employee("Prashanth", 1, "Bengaluru"),
    2: Employee("Shiva", 2, "Bengaluru"),
    3: Employee("Phaneendra", 3, "Mysore"),
    4: Employee("Pranav", 4, "Mysore"),
}

count = 4

# Exception Handling Classes

class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})

class ValidationError(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "Invalid Input Parameter")
```

```python
class EmployeeNotPresentError(BaseException):
    def __init__(self) -> None:
        super().__init__(404, "Employee not Present")


# Managing Logging

def setup_logger(called):
    def f(*args, **kwargs):
        request.logger = app.logger
        return called(*args, **kwargs)

    f.__name__ = called.__name__
    return f

@app.route("/api/employee/", methods=["POST"])
@setup_logger
def create_employee():
    global count
    employee = request.json
    count += 1
    employee['id'] = count
    emp = Employee(employee['name'], count, employee['address'])
    log_message = {'operation': 'create employee', 'status': 'processing'}

    request.logger.info(str(log_message))

    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))

        err = ValidationError()
        return str(err), err.status

    employees[count] =  emp

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return employee

@app.route("/api/employee/<int:employee_id>", methods=["PUT"])
@setup_logger
def alter_employee_data(employee_id):
    employee = request.json
    employees[employee_id].name = employee.get('name', employees[employee_id].
 ↪name)
    employees[employee_id].address = employee.get('address',␣
 ↪employees[employee_id].address)
    emp = Employee(employee['name'], count, employee['address'])
    log_message = {'operation': 'alter employee', 'status': 'processing'}

    request.logger.info(str(log_message))
    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))

        err = ValidationError()
        return str(err), err.status

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return str(employees[employee_id])

@app.route("/api/employee/<int:employee_id>", methods=["GET"])
@setup_logger
def get_employee_information(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    return str(emp)

@app.route("/api/employee/<int:employee_id>", methods=["DELETE"])
@setup_logger
def remove_employee_data(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    del employees[employee_id]
    return make_response(""), 200
```

1.3 Add the loggers to the other two methods

1.4 Tracing the APIs

[5]:
```python
import time
from flask import Flask
from flask import request, make_response

app = Flask(__name__)

class Employee:
    name = ''
    id = ''
    address = ''

    def __init__(self, name, id, address):
        self.name = name
        self.id = id
        self.address = address

    def __str__(self):
        return str({
            "id": self.id,
            "name": self.name,
            "address": self.address
        })

    def validate(self):
        val_name = len(self.name) > 0 and len(self.name) < 256
        val_addr = len(self.address) > 0 and len(self.address) < 1024
        return val_name and val_addr

employees = {
    1: Employee("Prashanth", 1, "Bengaluru"),
    2: Employee("Shiva", 2, "Bengaluru"),
    3: Employee("Phaneendra", 3, "Mysore"),
    4: Employee("Pranav", 4, "Mysore"),
}

count = 4

# Exception Handling Classes

class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})

class ValidationError(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "Invalid Input Parameter")


class EmployeeNotPresentError(BaseException):
    def __init__(self) -> None:
        super().__init__(404, "Employee not Present")

# Managing Logging

def setup_logger(called):
    def f(*args, **kwargs):
        request.logger = app.logger
        return called(*args, **kwargs)

    f.__name__ = called.__name__
    return f

def time_request(called):
    def f(*args, **kwargs):
        request.start_time = time.time() * 1000
        res = called(*args, **kwargs)
        request.end_time = time.time() * 1000
        request.time = request.end_time - request.start_time
        app.logger.info('request time: {}'.format(request.time))
        return res

    f.__name__ = called.__name__
    return f

def setup_tracing(called):
    def f(*args, **kwargs):
        request.req_id = 'req_{}'.format(time.time() * 1000)
        res, status = called(*args, **kwargs)
        res_send = make_response(res)
        res_send.headers['X-Request-Id'] = request.req_id
        return res_send, status

    f.__name__ = called.__name__
    return f


@app.route("/api/employee/", methods=["POST"])
@setup_logger
@time_request
@setup_tracing
def create_employee():
    global count
    employee = request.json
    count += 1
    employee['id'] = count
    emp = Employee(employee['name'], count, employee['address'])
    log_message = {'tracking_id': request.req_id,
                   'operation': 'create employee',
                   'status': 'processing'}

    request.logger.info(str(log_message))

    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))

        err = ValidationError()
        return str(err), err.status

    employees[count] =  emp

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return employee, 201

@app.route("/api/employee/<int:employee_id>", methods=["PUT"])
@setup_logger
@time_request
@setup_tracing
def alter_employee_data(employee_id):
    employee = request.json
    employees[employee_id].name = employee.get('name', employees[employee_id].
 ↪name)
    employees[employee_id].address = employee.get('address',␣
 ↪employees[employee_id].address)
    emp = Employee(employee['name'], count, employee['address'])
    log_message = { 'tracking_id': request.req_id,
                    'operation': 'alter employee',
                    'status': 'processing'}
    request.logger.info(str(log_message))
    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))

        err = ValidationError()
        return str(err), err.status

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return str(employees[employee_id]), 200

@app.route("/api/employee/<int:employee_id>", methods=["GET"])
@setup_logger
def get_employee_information(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    return str(emp), 200

@app.route("/api/employee/<int:employee_id>", methods=["DELETE"])
@setup_logger
def remove_employee_data(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    del employees[employee_id]
    return make_response(""), 200
```

Add tracing to the other two APIs and check the response.

1.5 Timing APIs

[7]:
```
import time
from flask import Flask
from flask import request, make_response
```

```python
app = Flask(__name__)

class Employee:
    name = ''
    id = ''
    address = ''

    def __init__(self, name, id, address):
        self.name = name
        self.id = id
        self.address = address

    def __str__(self):
        return str({
            "id": self.id,
            "name": self.name,
            "address": self.address
        })

    def validate(self):
        val_name = len(self.name) > 0 and len(self.name) < 256
        val_addr = len(self.address) > 0 and len(self.address) < 1024
        return val_name and val_addr

employees = {
    1: Employee("Prashanth", 1, "Bengaluru"),
    2: Employee("Shiva", 2, "Bengaluru"),
    3: Employee("Phaneendra", 3, "Mysore"),
    4: Employee("Pranav", 4, "Mysore"),
}

count = 4

# Exception Handling Classes

class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})
```

```python
class ValidationError(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "Invalid Input Parameter")


class EmployeeNotPresentError(BaseException):
    def __init__(self) -> None:
        super().__init__(404, "Employee not Present")

# Managing Logging

def setup_logger(called):
    def f(*args, **kwargs):
        request.logger = app.logger
        return called(*args, **kwargs)

    f.__name__ = called.__name__
    return f

def time_request(called):
    def f(*args, **kwargs):
        request.start_time = time.time() * 1000
        res = called(*args, **kwargs)
        request.end_time = time.time() * 1000
        request.time = request.end_time - request.start_time
        app.logger.info('request time: {}'.format(request.time))
        return res

    f.__name__ = called.__name__
    return f

def setup_tracing(called):
    def f(*args, **kwargs):
        request.req_id = 'req_{}'.format(time.time() * 1000)
        res, status = called(*args, **kwargs)
        res_send = make_response(res)
        res_send.headers['X-Request-Id'] = request.req_id
        return res_send, status

    f.__name__ = called.__name__
    return f


@app.route("/api/employee/", methods=["POST"])
@setup_logger
@time_request
@setup_tracing
```

```
def create_employee():
    global count
    employee = request.json
    count += 1
    employee['id'] = count
    emp = Employee(employee['name'], count, employee['address'])
    log_message = {'tracking_id': request.req_id,
                   'operation': 'create employee',
                   'status': 'processing'}

    request.logger.info(str(log_message))

    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))

        err = ValidationError()
        return str(err), err.status

    employees[count] =  emp

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return employee, 201

@app.route("/api/employee/<int:employee_id>", methods=["PUT"])
@setup_logger
@time_request
@setup_tracing
def alter_employee_data(employee_id):
    employee = request.json
    employees[employee_id].name = employee.get('name', employees[employee_id].
 ↪name)
    employees[employee_id].address = employee.get('address',␣
 ↪employees[employee_id].address)
    emp = Employee(employee['name'], count, employee['address'])
    log_message = { 'tracking_id': request.req_id,
                    'operation': 'alter employee',
                    'status': 'processing'}

    request.logger.info(str(log_message))
    # Validation error
    if not emp.validate():
        log_message['status'] = 'unsuccessful'
        request.logger.error(str(log_message))
        err = ValidationError()
        return str(err), err.status

    log_message['status'] = 'successful'
    request.logger.info(str(log_message))
    return str(employees[employee_id]), 200

@app.route("/api/employee/<int:employee_id>", methods=["GET"])
@setup_logger
def get_employee_information(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    return str(emp), 200

@app.route("/api/employee/<int:employee_id>", methods=["DELETE"])
@setup_logger
def remove_employee_data(employee_id):

    try:
        emp = employees[employee_id]
    except KeyError as e:
        err = EmployeeNotPresentError()
        return str(err), err.status

    del employees[employee_id]
    return make_response(""), 200
```

[ ]: