PLMS Assignment 5

PLMS - Building Microservices using NodeJS Assignments

Task:

Setup an API Gateway.

Task. Setup an API Gateway

In this task, you will be working on creating an API Gateway. There are two ways to do it.

1. Creating a new service called API Gateway. (if you cannot use Docker)
2. Use NGNIX. (Better Method but uses Docker)

Method 1 –

Creating a new service called API Gateway

Step 1 - Let us install the packages required. The packages required here are:

a) express-gateway
b) nodemon
c) path

Step 2 –

After installing these packages then let us create some model file. These files are just setup files for `express-gateway`.

i) applications.json

```
{
  "$id": "http://express-gateway.io/models/applications.json",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "isRequired": true
    },
    "name": {
      "type": "string",
      "isRequired": true
    },
    "redirectUri": {
      "type": "string"
    },
    "userId": {
      "type": "string",
      "isRequired": true
    }
  }
}
```

ii) credentials.json

```
{
  "$id": "http://express-gateway.io/models/credentials.json",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "isRequired": true
    },
    "secret": {
      "type": "string",
      "isRequired": true
    },
    "consumerId": {
      "type": "string",
      "isRequired": true
    }
  }
}
```

iii) user.json

```
{
  "$id": "http://express-gateway.io/models/users.json",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "isRequired": true
    },
    "name": {
      "type": "string",
      "isRequired": true
    },
    "email": {
      "type": "string",
      "isRequired": true
    },
    "password": {
      "type": "string",
      "isRequired": true
    },
    "role": {
      "type": "string",
      "isRequired": true,
      "enum": ["customer", "manager"]
    }
  }
}
```

Note: These files would be under the `path/models` *file.*

Step 3 – We will now work on the system config file. This file is used to set up the database for this service.

For now, you can use in-memory database but we you can use better storage using `redis`.

system.config.yml

```yaml
db:
  redis:
    emulate: true
    namespace: EG

crypto:
  cipherKey: "changeMe"

session:
  secret: "keyboard cat"
  resave: false
  saveUninitialized: false
```

Here the emulated tag is used to emulate that redis is working but you can also use your own redis server port if you have it installed.

Step 4 – Now, we actually work on the API gateway configuration. Create a file in the base the directory

```yaml
http:
  port: 8080

apiEndpoints:
  banks:
    host: localhost
    paths: "/api/bank/*"
  user:
    host: localhost
    paths: "/api/users/*"
  loan:
    host: localhost
    paths: "/api/loans/*"
  review:
    host: localhost
    paths: "/api/review/*"
  notification:
    host: localhost
    paths: "/api/notifications/*"

serviceEndpoints:
  bank-service:
    url: "http://localhost:8002/api/banks"
  auth-service:
    url: "http://localhost:8001/api/users"
  loan-service:
    url: "http://localhost:8003/api/loans"
  review-service:
    url: "http://localhost:8004/api/review"
  notification-service:
    url: "http://localhost:8005/api/notifications"

policies:
  - rate-limit
  - proxy

pipelines:
  bank-pipeline:
    apiEndpoints:
      - banks
    policies:
```

```yaml
    - rate-limit:
        - action:
            max: 100
            windowMs: 60000
            rateLimitBy: "${req.ip}"
            skipSuccessfulRequests: false
            skipFailedRequests: false
            message: "Too many requests from this IP for bank service"
    - proxy:
        - action:
            serviceEndpoint: bank-service
            changeOrigin: true
            stripPath: true

auth-pipeline:
  apiEndpoints:
    - user
  policies:
    - rate-limit:
        - action:
            max: 100
            windowMs: 60000
            rateLimitBy: "${req.ip}"
            skipSuccessfulRequests: false
            skipFailedRequests: false
            message: "Too many requests from this IP for auth service"
    - proxy:
        - action:
            serviceEndpoint: auth-service
            changeOrigin: true
            stripPath: true

loan-pipeline:
  apiEndpoints:
    - loan
  policies:
    - rate-limit:
        - action:
            max: 100
            windowMs: 60000
            rateLimitBy: "${req.ip}"
            skipSuccessfulRequests: false
            skipFailedRequests: false
            message: "Too many requests from this IP for loan service"
```

```yaml
    - proxy:
        - action:
            serviceEndpoint: loan-service
            changeOrigin: true
            stripPath: true

review-pipeline:
  apiEndpoints:
    - review
  policies:
    - rate-limit:
        - action:
            max: 100
            windowMs: 60000
            rateLimitBy: "${req.ip}"
            skipSuccessfulRequests: false
            skipFailedRequests: false
            message: "Too many requests from this IP for review service"
    - proxy:
        - action:
            serviceEndpoint: review-service
            changeOrigin: true
            stripPath: true

notification-pipeline:
  apiEndpoints:
    - notification
  policies:
    - rate-limit:
        - action:
            max: 50
            windowMs: 60000
            rateLimitBy: "${req.ip}"
            skipSuccessfulRequests: false
            skipFailedRequests: false
            message: "Too many requests from this IP for notification service"
    - proxy:
        - action:
            serviceEndpoint: notification-service
            changeOrigin: true
            stripPath: true
```

Note: This is a basic config file example but you can choose the configuration for your own project.

Step 5 – Now we create a server.js file in the base directory of our service.

```
const gateway = require("express-gateway");
const path = require("path");

process.env.EG_CONFIG_DIR = path.join(__dirname);

gateway().run();

console.log("API Gateway service listening at port defined in config!");
```

And according to your configuration file you can evaluate out your API Gateway from `localhost:<PORT>` where PORT is 8080 according to the example above.

Note: You need to run all the service before using the API gateway otherwise it will just give you errors.

Method 2 –
Use NGNIX.

This Method is easier to setup and requires minimal setup configuration.

Step 1 – Create a config file in your base directory called default.conf

```
worker_processes 1;
events { worker_connections 1024; }

http {
    # Rate limit: 10 requests/sec per client IP with burst up to 5
    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=2r/s;

    server {
        listen 80;

        # Proxy settings for all services
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # These ensure all headers (including custom ones) are preserved
        proxy_pass_request_headers on;

        location /api/users {
            limit_req zone=mylimit;
            limit_req_status 429;
            proxy_pass http://host.docker.internal:8001/api/users;
        }

        location /api/banks {
            limit_req zone=mylimit;
            limit_req_status 429;
            proxy_pass http://host.docker.internal:8002/api/banks;
        }

        location /api/loans {
            limit_req zone=mylimit;
            limit_req_status 429;
            proxy_pass http://host.docker.internal:8003/api/loans;
        }
```

```
        location /api/review {
            limit_req zone=mylimit;
            limit_req_status 429;
            proxy_pass http://host.docker.internal:8004/api/review;
        }

        location /api/api/notifications {
            limit_req zone=mylimit;
            limit_req_status 429;
            proxy_pass http://host.docker.internal:8005/api/notifications;
        }
    }
}
```

Step 2 – After the setup is complete we can simply start with creating the docker container.

```
FROM nginx:latest
COPY default.conf /etc/nginx/nginx.conf
```

First, move to the project directory then we will run the following command to run the API gateway.

```
docker build -t nginx-gateway .
```

```
docker run -d --name nginx-gateway -p 8080:80 nginx-gateway
```

Congratulation, you have successfully set up your API Gateway with rate limiter.

Now, you can test your API Gateway at `localhost:8080` using Postman or API tester of your choice.

Congrats! You have successfully completed this 5-day intensive React training!

We really appreciate your support and hardworking throughout the week. Hope you have enjoyed React!

Have a good weekend and see you next time!