The QuoteServices App Case Study: Gateway Microservice Structure & Functionality

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

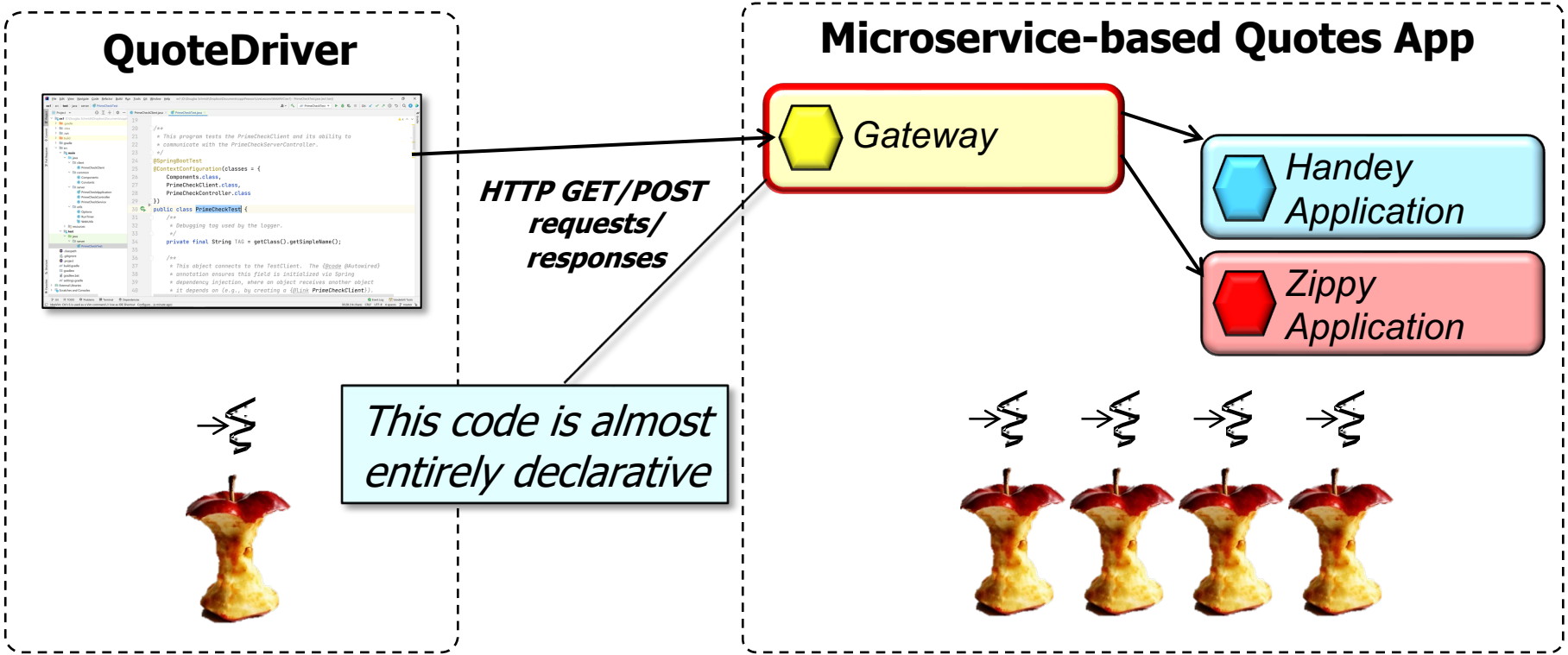
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of the Gateway microservice implementation in the QuotesServices app case study

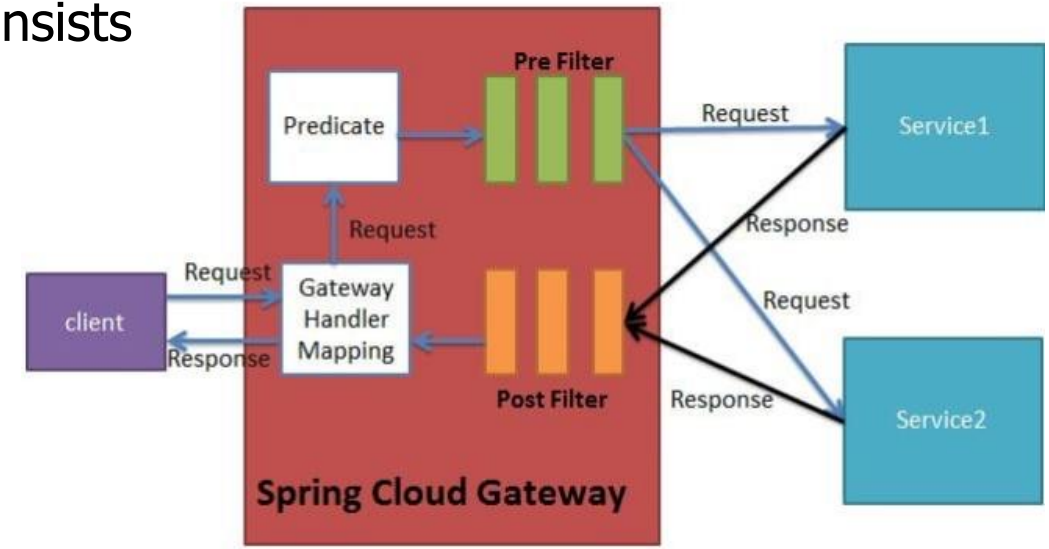


See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex4/gateway

Overview of the Spring Cloud API Gateway

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components



See blog.knoldus.com/spring-cloud-api-gateway

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - **Routes**
    - Consists of an ID, destination URI, collection of predicates, & a collection of filters

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
  - id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - **Routes**
    - Consists of an ID, destination URI, collection of predicates, & a collection of filters
      - Routes can be created either programmatically (using Java) or declaratively (using YAML)

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
  - id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - Routes
  - **URI**
    - The URI indicates where the microservice resides
      - This directive isn't necessary if a discover service is used

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
  - id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

This implementation hard-codes the URIs & does not use a discovery service

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - Routes
  - URI
  - **Predicates**
    - Can match HTTP requests
      - e.g., headers, URLs, cookies, or parameters

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
  - id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

Match "handey" or "zippy" route names in the path

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - Routes
  - URI
  - **Predicates**
    - Can match HTTP requests
    - A route is considered "matched" if the aggregate predicate is true

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
- id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

Overview of the Spring Cloud API Gateway

- The Spring Cloud API Gateway consists of several components
  - Routes
  - URI
  - Predicates
  - **Filters**
    - Can modify the request or response as per requirements

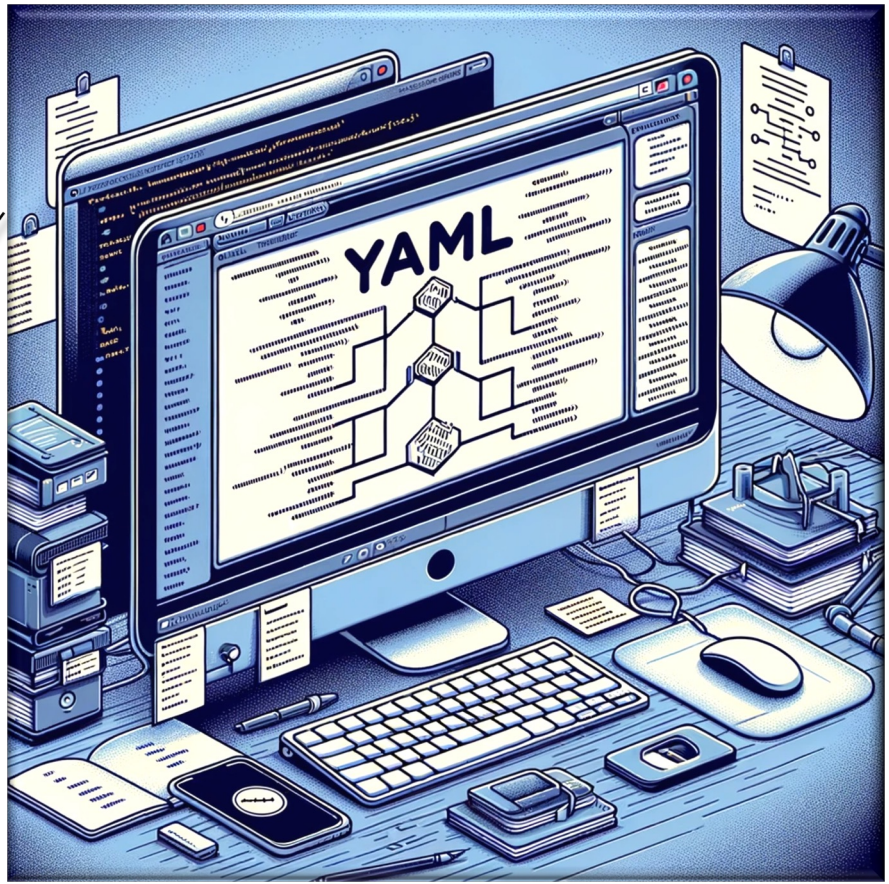
Remove the first path segment (e.g., "handey" or "zippy") from URI before forwarding the request to the downstream microservice

```
routes:
  - id: handey
    uri: http://localhost:9100
    predicates:
      - Path= /handey/**
    filters:
      - StripPrefix=1 # ...
  - id: zippy
    uri: http://localhost:9101
    predicates:
      - Path= /zippy/**
    filters:
      - StripPrefix=1
```

Structure & Functionality of the Gateway Microservice

Structure & Functionality of the Gateway Microservice

- The API Gateway is configured largely using declarative YAML files



YAML = "YAML Ain't Markup Language"!!

See en.wikipedia.org/wiki/YAML

Structure & Functionality of the Gateway Microservice

- This API Gateway is configured largely using declarative YAML files
  - application.yml

```
server:
  port: 8080
spring:
  profiles:
    active: path
  application:
    name: gateway
  ...
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    serviceUrl:
      defaultZone:
        http://localhost:8761/eureka
```

YAML data file configures a Gateway microservice & registers it with Eureka

See WebMVC/ex4/gateway/src/main/resources/application.yml

Structure & Functionality of the Gateway Microservice

- This API Gateway is configured largely using declarative YAML files
  - application.yml
  - application-path.yml

This YAML data file configures the routes to Quotes microservices that are handled automatically by the Gateway microservice

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: handey
          uri: http://localhost:9100
          predicates:
            - Path= /handey/**
          filters:
            - StripPrefix=1 # ...
        - id: zippy
          uri: http://localhost:9101
          predicates:
            - Path= /zippy/**
          filters:
            - StripPrefix=1 # ...
```

See WebMVC/ex4/gateway/src/main/resources/application-path.yml

End of the QuoteServices App Case Study: Gateway MicroService Structure & Functionality