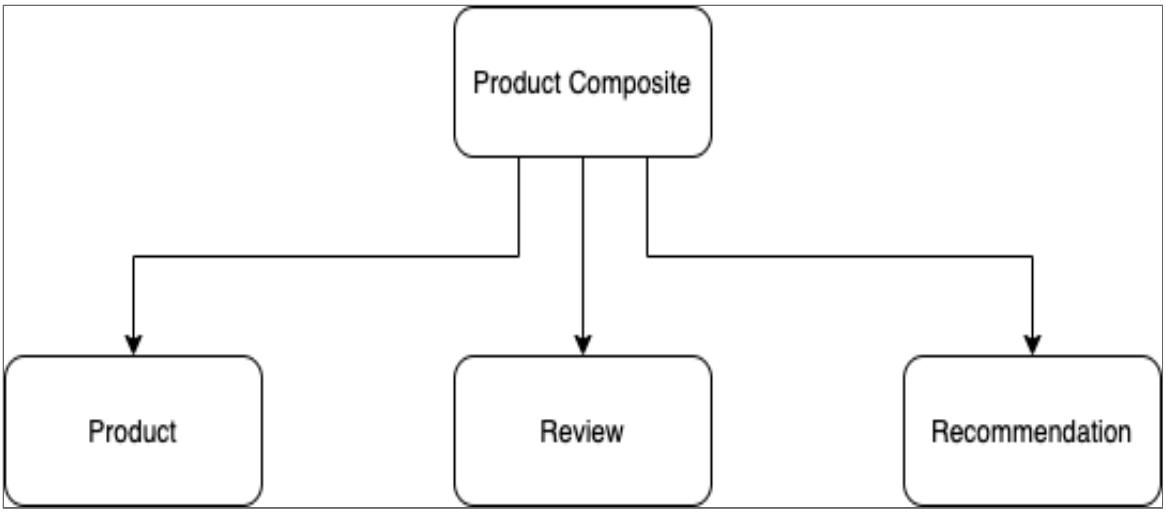
RESTful API

Gene Kuo

Topics

- Cooperating microservices
- Service addresses and discovery in code
- RESTful API
- Composite microservice
- Error handling
- Testing API
- Automatic tests
- OpenAPI/Swagger and SpringFox

Cooperating microservices



Service addresses and discovery in code

- serviceAddress
- application.yml

Main application classes

- `@SpringBootApplication`
- `@RunWith(SpringRunner.class)` and `@SpringBootTest`

Multi-project build

- `settings.gradle`
- Pros and cons

`RESTful API`

- api project: seperating the API definition from its implementation
  - Spring Framework greater than v5.1.0
- util project: helper classes shared by microservices
- `dependencyManagement` in gradle file
- `ProjectService.getProduct`
- POJO and entity
- util: `InvalidException`, `NotFoundException`, `GlobalException`, `HttpErrorInfo`, `ServiceUtil`

`ProductServiceImpl`

- `ServiceUtil`
- Implement API
- Runtime properties: application.yml

Composite microservice

- APIs, Integration component and composite service implementation
  - `application.yml`
  - `ProductCompositeIntegration` implements microservice interfaces
  - `ProductCompositeIntegration`, `RestTemplate` and `ObjectMapper` for error handling
  - `RestTemplate.exchange` and `ParameterizedTypeReference`
  - `ProductCompositeServiceImpl`

`Error handling`

- Separate protocol-specific errors from business logic errors
- `GlobalControllerExceptionHandler`: from business logic exception to HTTP error response
- `ProductCompositeIntergarion` error handling is reversed
- Best-effort error handling in `ProductCompositeIntergarion`

Testing API

Automatic tests

- Positive and negative tests
- `WebTestClient`
- Tests for composite microservices: mock its dependencies
  - `@MockBean`
- `test-all.sh`

Docker and microservices

- spring.profiles: docker

```
./gradlew :microservices:product-
service:build
docker build -t product-service .
docker images | grep product-service
docker run -d -p8080:8080
  -e "SPRING_PROFILES_ACTIVE=docker"
  --name prd-srv product-service
curl localhost:8080/product/3
docker ps
docker logs prd-svc -f
docker rm -f prd-svc
```

Docker Compose

- When running in Docker, each service has its own hostname and the same port (8080)
- In `docker-compose.yml`, the hostname of each service is specified.

```
./gradlew build && docker-compose  build
docker images | grep
docker-compose up -d
docker-compose logs -f
curl localhost:8080/product-compose/123 -s
| jq .
docker-compose down
test-all.sh start stop
```

Troubleshotting

```
docker-compose ps
docker-compose logs product
docker system prune -f volumes
docker-compose up -d --sacle product=0`
and then `docer-compose up -d --scale
product=1
./gradlew clean build && dcoker-compose
build && ./test-all start stop
```

OpenAPI/Swagger and SpringFox

- Combine documentation of APIs with the source code to the Java interfaces
- Depends on Spring WebFlux and Swagger
- `product-composite-service` describes general info about the API
  - `io.springfox:springfox-swagger2`, `io.springfox:springfox-swagger-ui`, and `io.springfox:springfox-spring-webflux`
  - `@EnableSwagger2WebFlux`, `Docket`, `apis`, `paths`

OpenAPI/Swagger and SpringFox

- `api` describes each RESTful service
  - `@Api`, `@ApiOperation`, `@ApiResponses`

Testing API

```
docker ps --format {{.Names}}
docker-compose down (cd to the directory)
./gradlew build && docker-compose build &&
docker-compose up -d
./test-all.sh
http://localhost:8080/swagger-ui.html
http://localhost:8080/v2/api-docs
curl -X GET
"http://localhost:8080/product-
composite/123" -H "accept:
application/json"
```