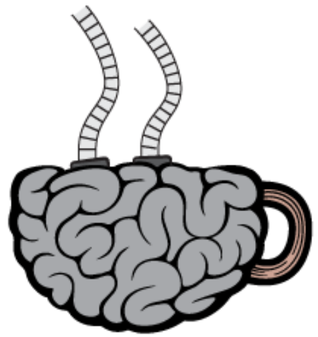
Building a REST API with Functional Endpoints

Esteban Herrera
Author | Developer | Consultant

@eh3rrera eherrera.net



WIRED BRAIN

COFFEE

Product Catalog

- Get all products
- Get a specific product
- Register a new product
- Update a product
- Delete a product
- Delete all products
- Events

Overview

- Spring WebFlux functional endpoints
- Setting up the project
- Building the handler functions
- Building the router functions

Functional Endpoints

Handler Function

```
public Mono<ServerResponse> myHandlerFunction(ServerRequest request) {



}
```

Handler Function

```
public Mono<ServerResponse> myHandlerFunction(ServerRequest request) {
  Mono<Product> product = request.bodyToMono(Product.class);



}
```

Handler Function

```
public Mono<ServerResponse> myHandlerFunction(ServerRequest request) {
  Mono<Product> product = request.bodyToMono(Product.class);
  // Flux<Product> product = request.bodyToFlux(Product.class);


}
```

Handler Function

```
public Mono<ServerResponse> myHandlerFunction(ServerRequest request) {
  Mono<Product> product = request.bodyToMono(Product.class);
  // Flux<Product> product = request.bodyToFlux(Product.class);
  return  ServerResponse.ok()
            .contentType(MediaType.APPLICATION_JSON).body(product);
}
```

Handler Function

```
class ProductHandler {

  public Mono<ServerResponse> myHandlerFunction(ServerRequest request) {
    Mono<Product> product = request.bodyToMono(Product.class);
    // Flux<Product> product = request.bodyToFlux(Product.class);
    return  ServerResponse.ok()
              .contentType(MediaType.APPLICATION_JSON).body(product);
  }

}
```

Router Function

```
public Mono<HandlerFunction> myRouterFunction(ServerRequest request) {
  // ...
}
```

Router Function

```
RouterFunctions.route(RequestPredicate, HandlerFunction)
```

RequestPredicates Class

```
public abstract class RequestPredicates {
  static RequestPredicate accept(MediaType... mediaTypes)
  static RequestPredicate GET(String pattern)
  static RequestPredicate method(HttpMethod httpMethod)
  static RequestPredicate path(String pattern)
  // ...
}
```

HandlerFunction Interface

```
interface HandlerFunction<T extends ServerResponse> {
  Mono<T> handle(ServerRequest request)
}
```

Router Function

```
RouterFunction<ServerResponse> myRoute =
  RouterFunctions.route(
    RequestPredicates.path("/product"),
    request -> Response.ok().body(productFlux)
  );
```

Router Function

```
RouterFunction<ServerResponse> myRoute =
  RouterFunctions.route(
    RequestPredicates.path("/product"),
    handler::getProduct
  );
```

Router Function

```
RouterFunction<ServerResponse> myRoute =
  RouterFunctions.route(
    RequestPredicates.path("/product"),
    handler::getProduct
  )
  .andRoute(RequestPredicates.POST("/product"),
            handler::saveProduct
  );
```

Router Function

```
RouterFunction<ServerResponse> myRoute =
  RouterFunctions.route()
    .RequestPredicates.GET("/product", handler::getProduct)
    .RequestPredicates.POST("/product", handler::saveProduct)
    .build();
```

Router functions are evaluated
in order

Declare more specific routes before general ones

Things to Remember

Functional programming model

- **Router function**
- **Handler functions**

Same Spring Boot/Spring Data configuration as the annotation-based model

Reactive programming is the foundation