SPRING REACTIVE PLAYBOOK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Reviewed by | Comments |
| 1.0 | 04/16/2023 | Ram Prakash Singh  Vandana Lalit | Padma |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

**Reactive Programming**3

**Prerequisites Software & Tools** 6

**Project Structure**11

**Configuration Files**13

**Key Classes** 16

**Reactive Application Up & Running** 22

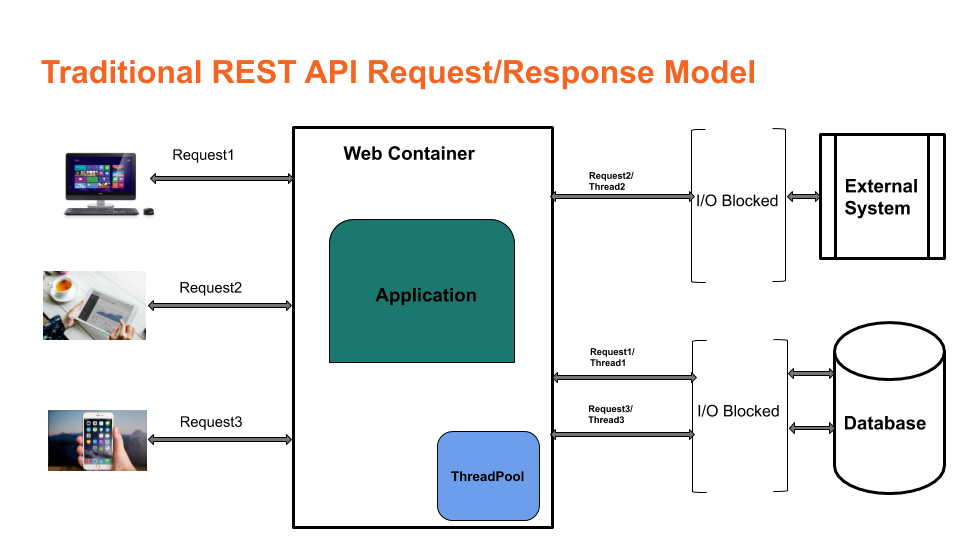
Reactive Programming:

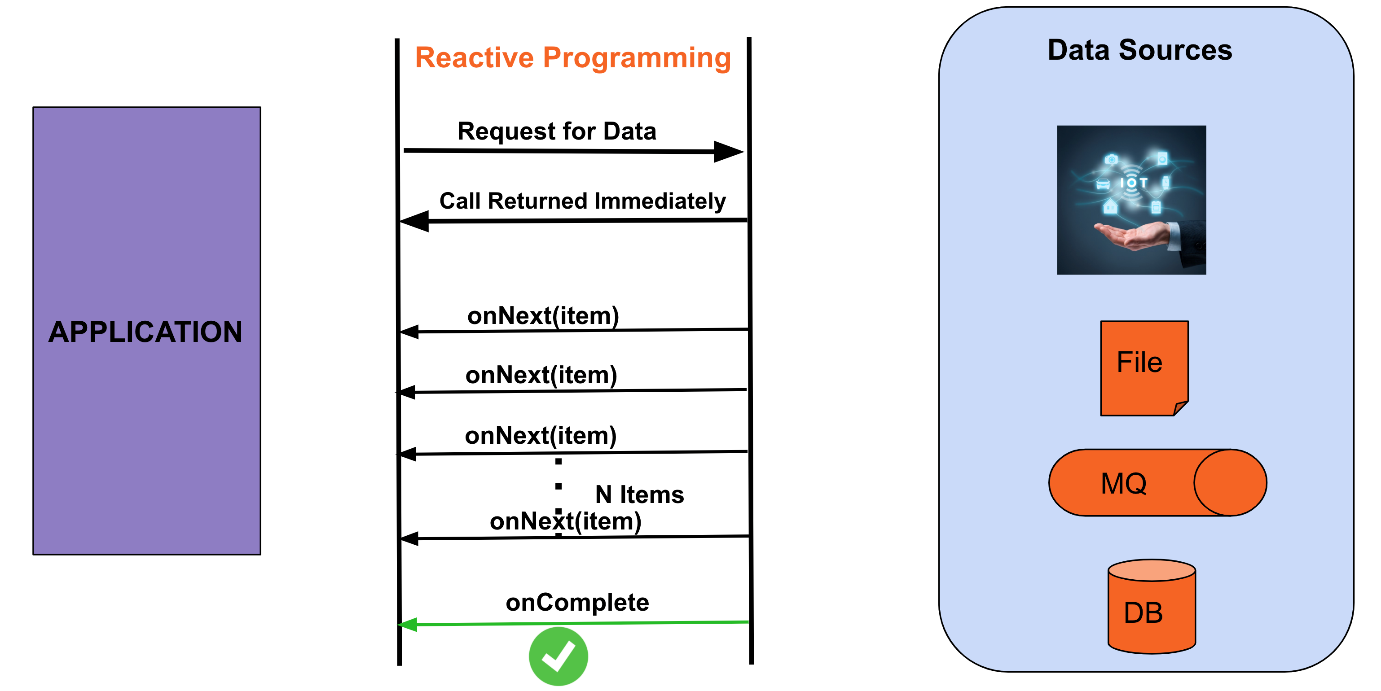
* **Non-Blocking: Reactive programming is a declarative programming paradigm that is based on the idea of asynchronous event processing and data streams.**
* **Asynchronous: Asynchronous processing means that the processing of an event does not block the processing of other events.**
* **Back Pressure: The ability of the consumer to request items at different rates or only when it is ready to process them.**
* **Event Driven: Web applications adopt non-blocking servers based on the event loop model.**

**Need for Reactive Programming:**

**Most applications in a fast-paced world need to be reactive and users are expecting results within the next second after submitting the request. Hence there is a need to build a system that is highly reactive and executes the requests in parallel.**

**Spring reactive programming utilizes the netty server event loop model which saves the number of threads executing in memory, unlike the Spring MVC thread per request model.**





Spring Web flux:

**Spring web flux provides reactive programming support for web applications.**

A picture containing diagram

Description automatically generated

**Traditional Spring MVC Thread per request model**

Diagram, schematic

Description automatically generated

**Spring Web flux Event loop Model**

**Benefits Of Reactive Programming:**

* Move away from the thread per request model and can handle more requests with a limited thread pool.
* Prevent threads from blocking while waiting for I/O operations to complete hence maximum throughput can be achieved.
* Make it easy to do parallel calls.
* Support “back pressure”, giving the client a possibility to inform the server on how much load it can handle.
* Better user experience.

Reactive Streams & Project Reactor:

**Spring Web Flux supports the Reactive Stream API, which is a standardized tool for processing asynchronous streams with non-blocking backpressure.**

**Reactive streams have a publisher (producer) — subscriber (consumer) model. The publisher emits an event, and a subscriber will read it. In the Reactive Streams API, there are four main interfaces:**

**Publisher — Emits events to subscribers based on the demands received from its subscribers. A publisher can serve multiple subscribers and it has only one method: subscribe**

**Subscriber — Receives events emitted by the Publisher. The subscribe has four methods to deal with the events received: onSubscribe, onNext, onError , and onComplete**

**Subscription — This represents the relationship between the subscriber and the publisher. It has methods that allow requesting for data request(long n) and to cancel the demand of events cancel()**

**Processor — Publisher and subscriber at the same time; rarely used.**

**Spring Web Flux internally uses Project Reactor and its publisher implementations, Flux and Mono.**

**Mono — A publisher that can emit 0 or 1 element.**

**Flux — A publisher that can emit 0..N elements.**

**Mono and Flux offer simple ways of creating streams of data:**

**Reactor offers several operators for working with Flux and Mono objects. Most commonly used are:**

**Map — Used to transform the publisher elements to other elements**

**FlatMap — Similar to map, but the transformation is asynchronous**

**FlatMapMany — Mono operator used to transform a Mono into a Flux**

**DelayElements — Delays the publishing of each element by a given duration**

**Concat — Used to combine publishers’ elements by keeping the sequence of the publishers**

**Merge — Used to combine publishers without keeping the publishers’ sequence, instead it interleaves the values**

**Zip — Used to combine two or more publishers by waiting on all the sources to emit one element and combining these elements into an output value.**

**doOnError- we can invoke whatever action that is needed when there is an error.**

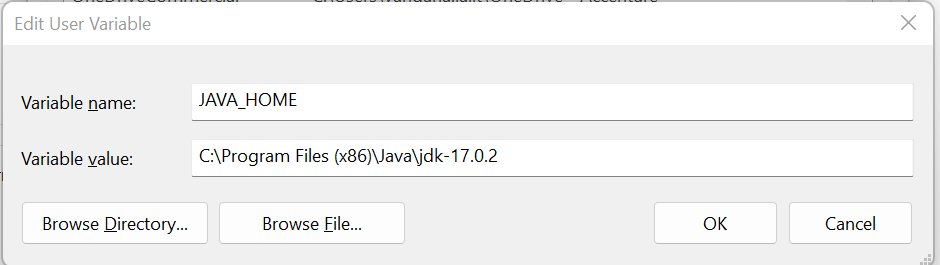
**onErrorResume: Gives a fallback stream when some exception occurs happens in the upstream.**

Prerequisites Software & Tools:

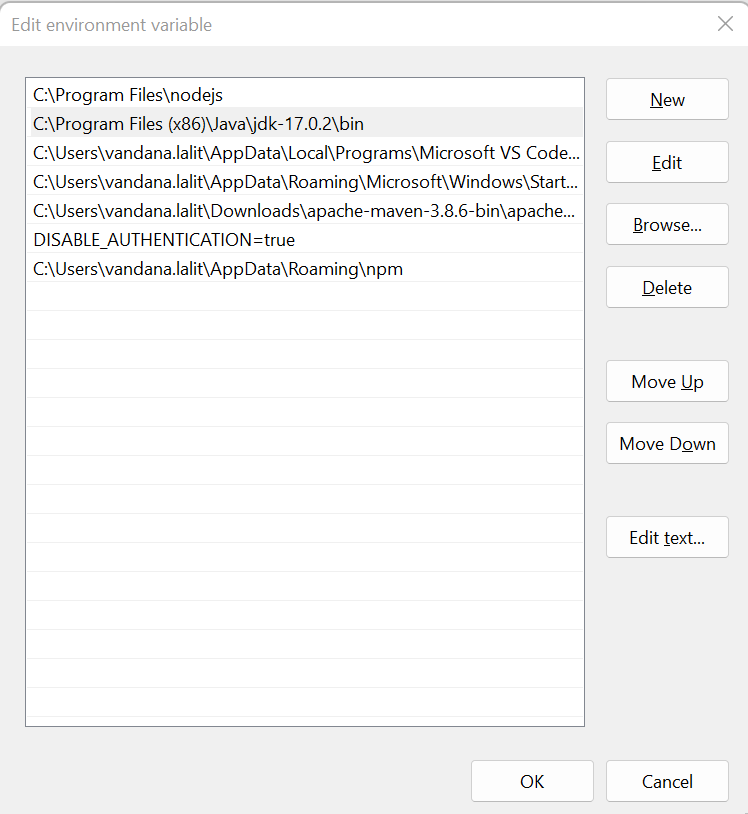
* **Java 17**
* **Maven**
* **Docker Desktop**
* **Postgres Database**
* **Eclipse(Or Any other IDE )**

**Java 17 Installation & Setup:**

1. **Download Java 17 from** [**Java Archive Downloads - Java SE 17 (oracle.com)**](https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)
2. Install Java 17 with the installer
3. Setup environment variable JAVA\_HOME

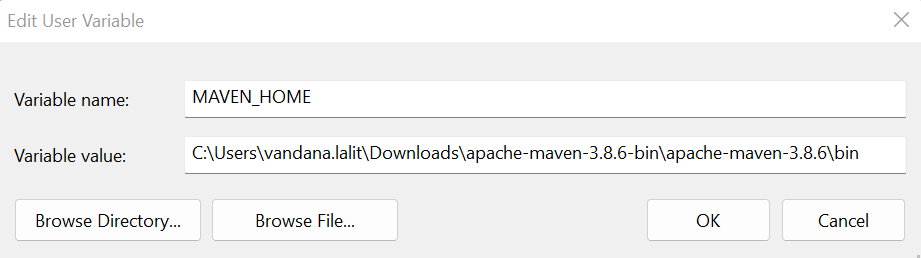


1. **Add Java to the path variable. For environment variables click on Path and edit.**

****

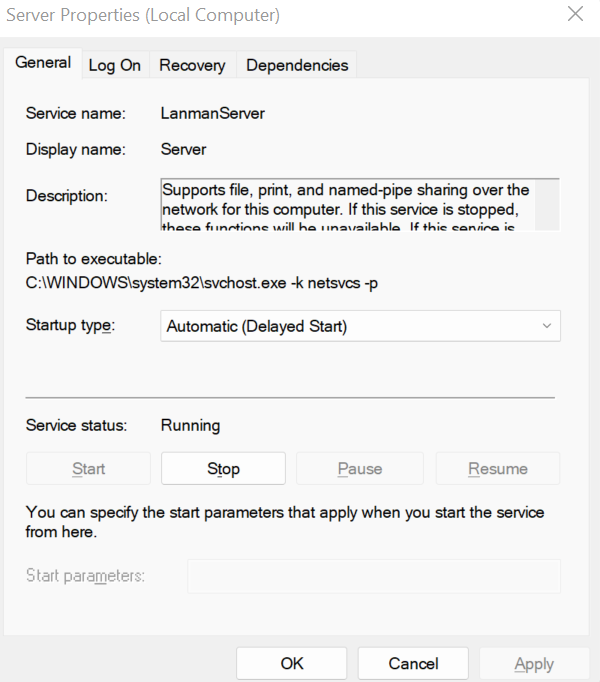
**Maven Installation & Setup:**

1. **Download maven latest version from** [Maven – Download Apache Maven](https://maven.apache.org/download.cgi)
2. Install maven with installer
3. Setup environment variable MAVEN\_HOME



**Docker Desktop Installation & Setup:**

1. **Download docker desktop latest version from** [Download Docker Desktop | Docker](https://www.docker.com/products/docker-desktop/)
2. Install docker desktop with installer
3. Enable the server in services.msc



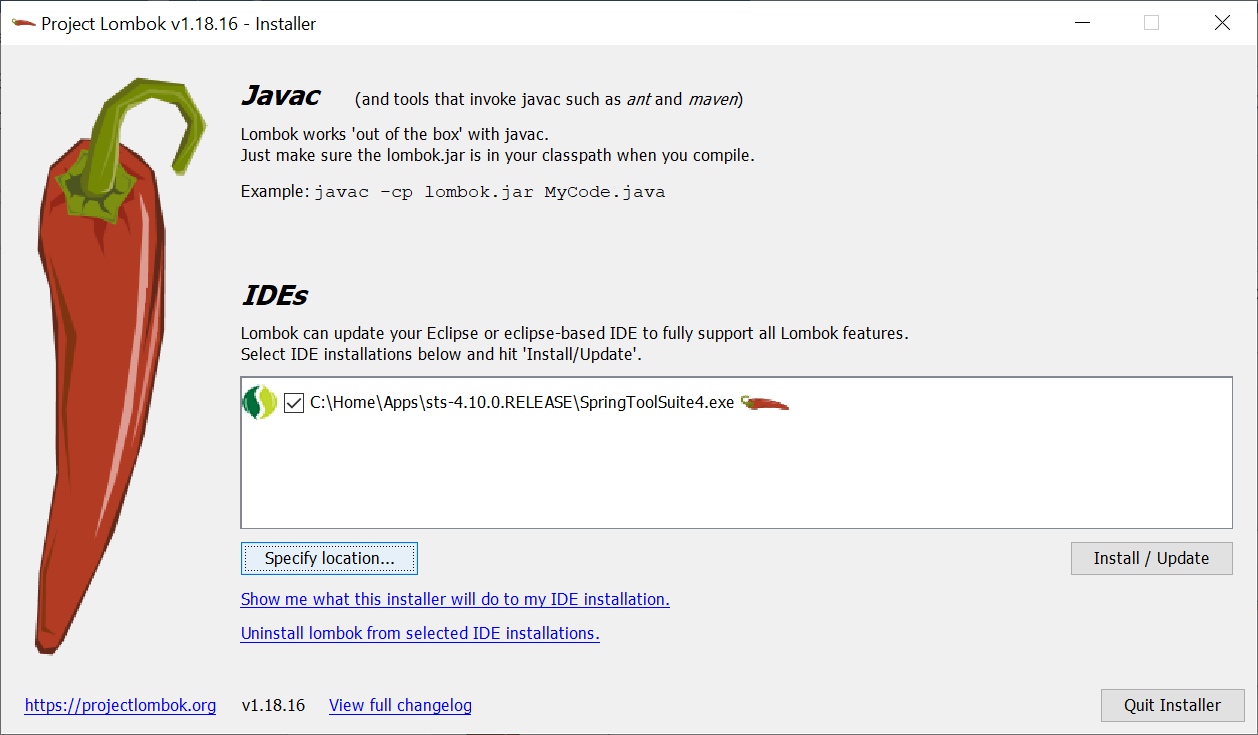
**PostgreSQL Installation & Setup:**

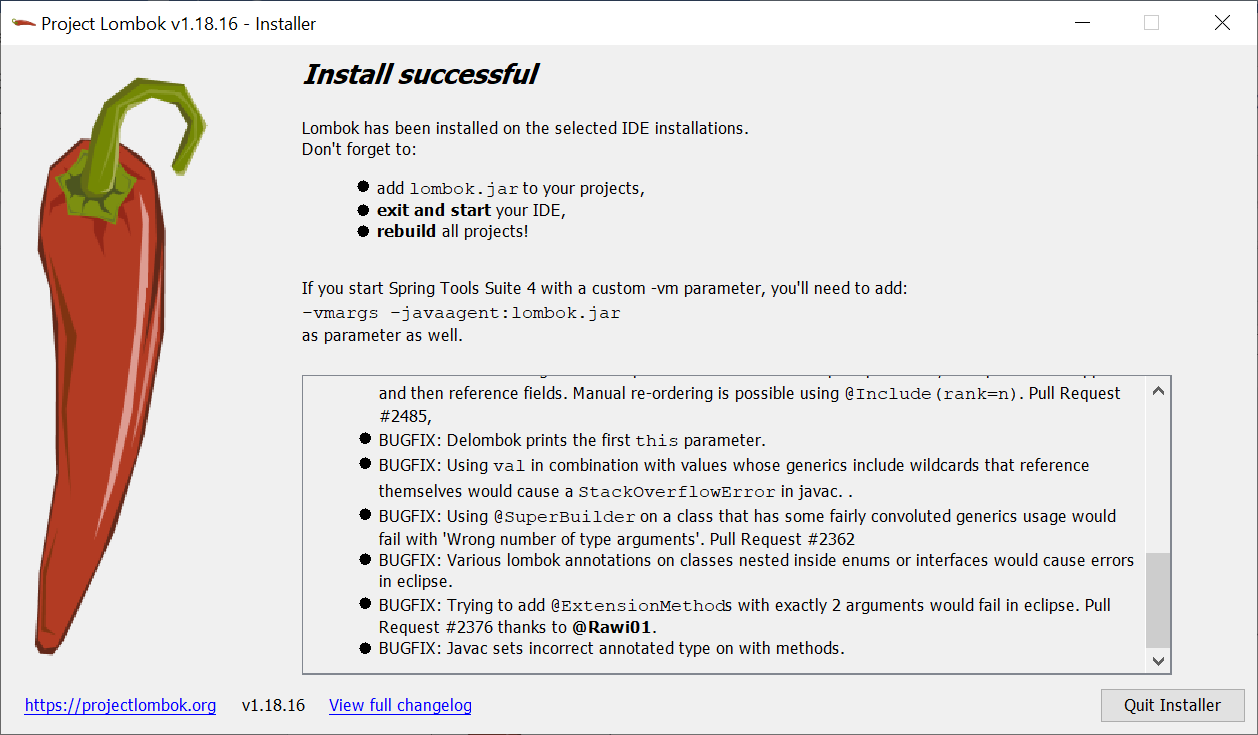
1. **Download** Postgres **latest version from** [PostgreSQL: Downloads](https://www.postgresql.org/download/)
2. Install Postgres with the installer

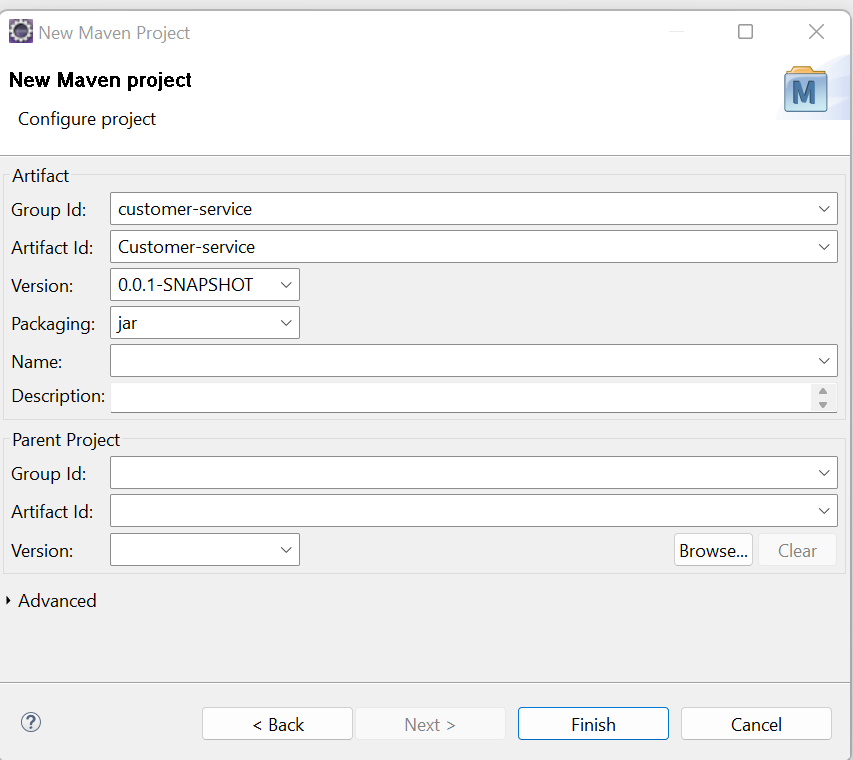
**Eclipse IDE Installation & Setup:**

1. **Download** eclipse **latest version from** [Eclipse Downloads | The Eclipse Foundation](https://www.eclipse.org/downloads/)
2. Install eclipse with the installer
3. Configure projectlombok from following command

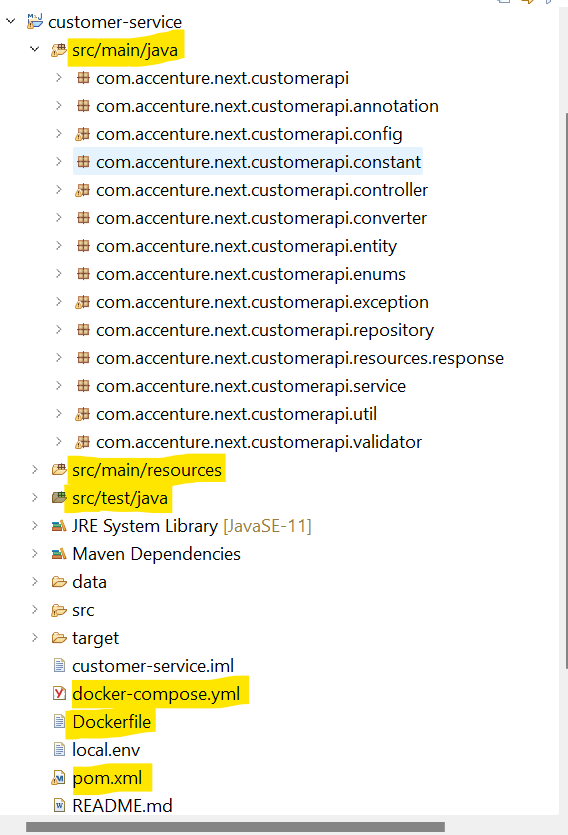
C:\Users\vandana.lalit\.m2\repository\org\projectlombok\lombok\1.18.16>java -jar lombok-1.18.16.jar





Project Structure: **Create a new Maven Project****t**

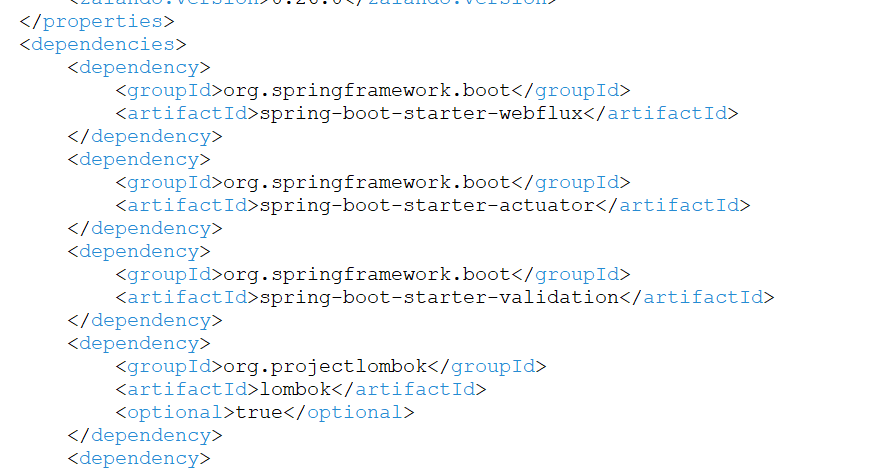
**Create the packages main (along with service package structure) and test packages.**

****

**Configuration Files:**

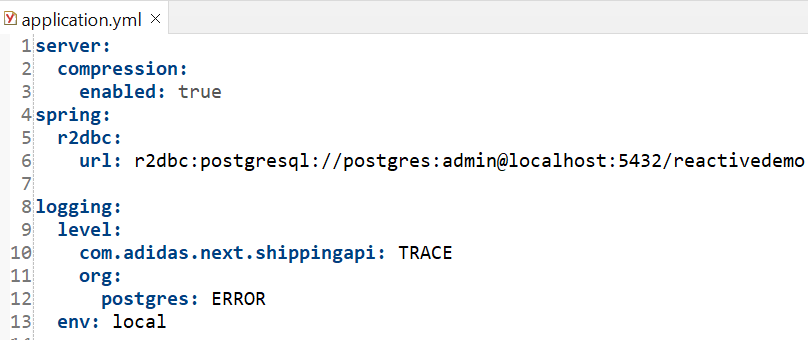
**Pom.xml**

* **Pom file includes all the project specific dependencies like Java 17, SpringWebflux, Project reactor, Zalando, PostgreSQL, r2dbc, Spring Boot validation, Junit, and Lombok.**
* **This file will download all the Maven dependencies.**
* **Spring Webflux, Project reactor, and R2DBC dependency need to be added which supports reactive programming.**

****

**Application.yml**

* **It is a file wherein we define the configuration data required for the project. AT application startup all the configurations are read from this file.**
* **We can define configurations like database servers, logging ,change in port number etc.**
* **Define PostgreSQL database and logging configuration.**

****

**Application-ci.yml**

* **Define PostgreSQL database and logging configuration.**
* **While running the application via Docker desktop, the configuration will be picked up from the application-ci.yml file.**

**Graphical user interface, text, application

Description automatically generated**

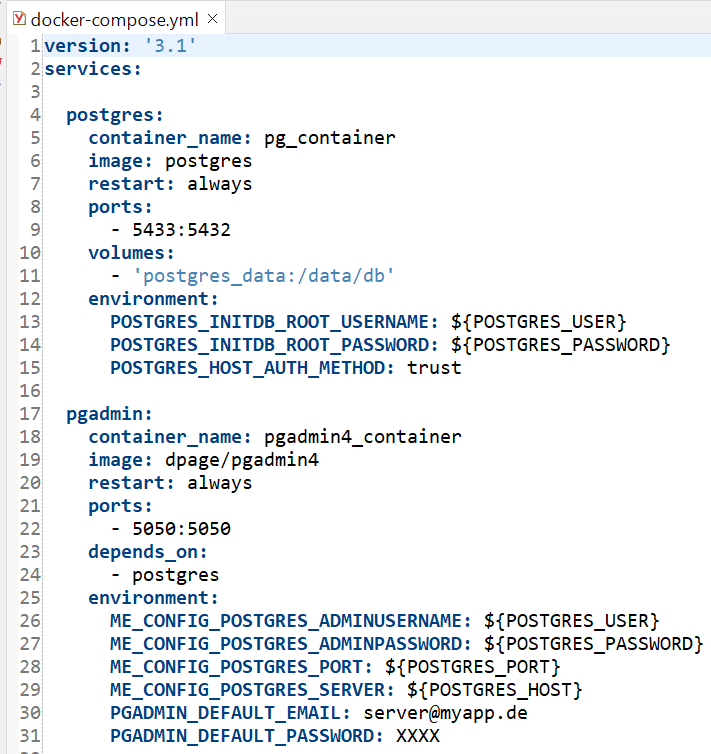
**Dockerfile.yml**

* **A Docker file is a text document that contains commands that are used to assemble an image.**
* **Docker builds images automatically by reading the instructions from the Docker file. For example, jdk, maven etc.**

****

**Docker-compose.yml**

* **Define the services that make up your application in docker-compose.yml so they can be run together in an isolated environment.**
* **Docker-compose.yaml file is used to run images.**

****

**Text

Description automatically generated**

**Key Classes:**

**Controller layer**

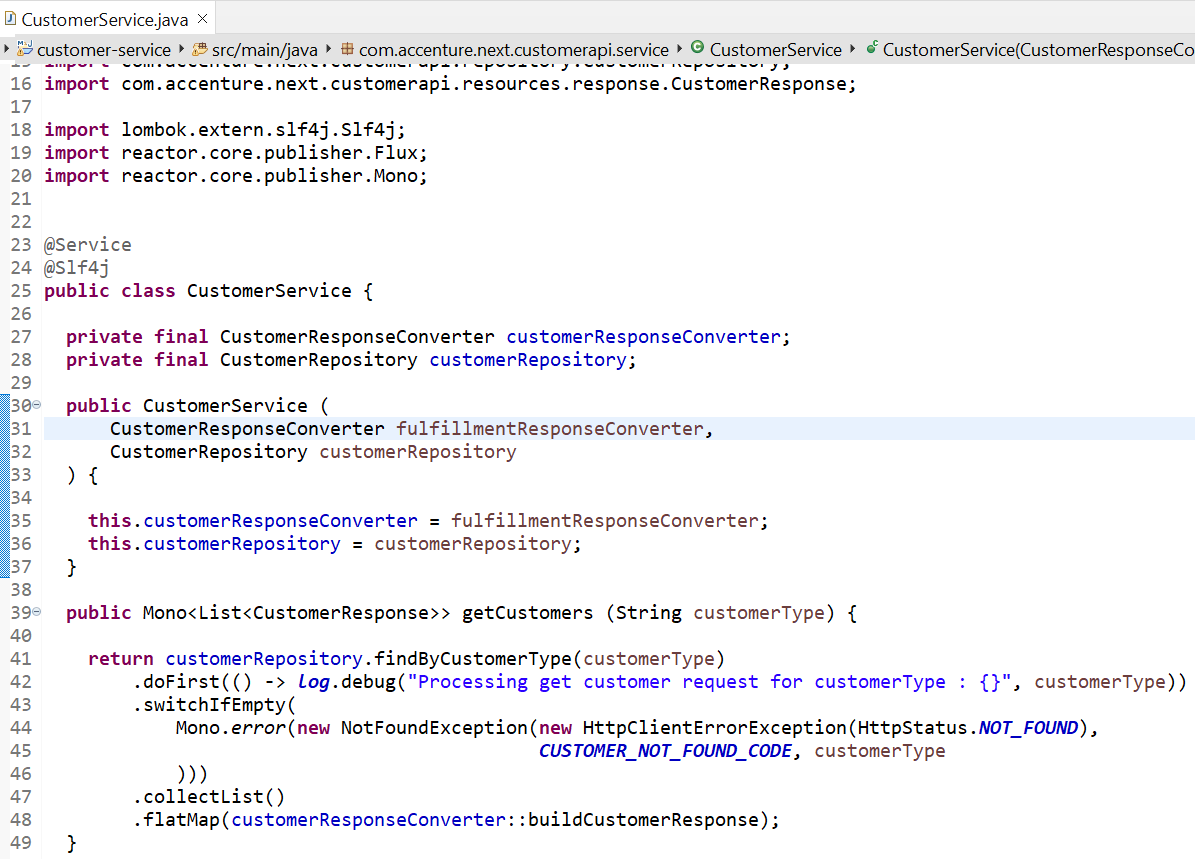
* **Controller layer is responsible for processing incoming REST API and returning the view to be rendered as a response.**
* **Define request mapping along with Get/Post/Put endpoints and calling service layer.**
* **@RestController annotation is used to mark the class as a controller below.**
* **We have defined a get endpoint that will return customer details based on Customer type.**

**Graphical user interface, text, application, email

Description automatically generated**

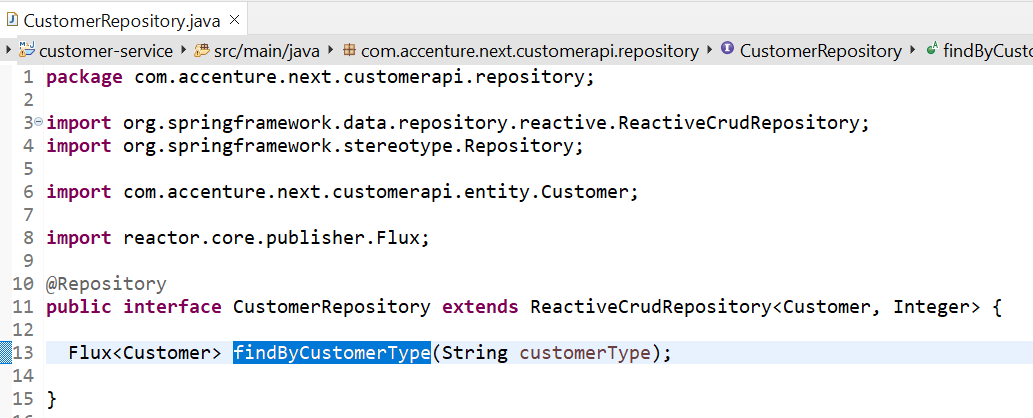
**Service layer**

* **Layer that facilitates the communication between the controller and the persistence layer.**
* **Holds the business logic.**
* **@Service annotation used to annotate this as a service class.**

****

**Repository layer**

* **Fetch a record from the database.**
* **We have defined the function to find the customers and the return type will be Flux since there can be more than one customer belonging to the same category.**
* **@Repository annotation is used to mark this class as a repository.**

****

**Exception Handling**

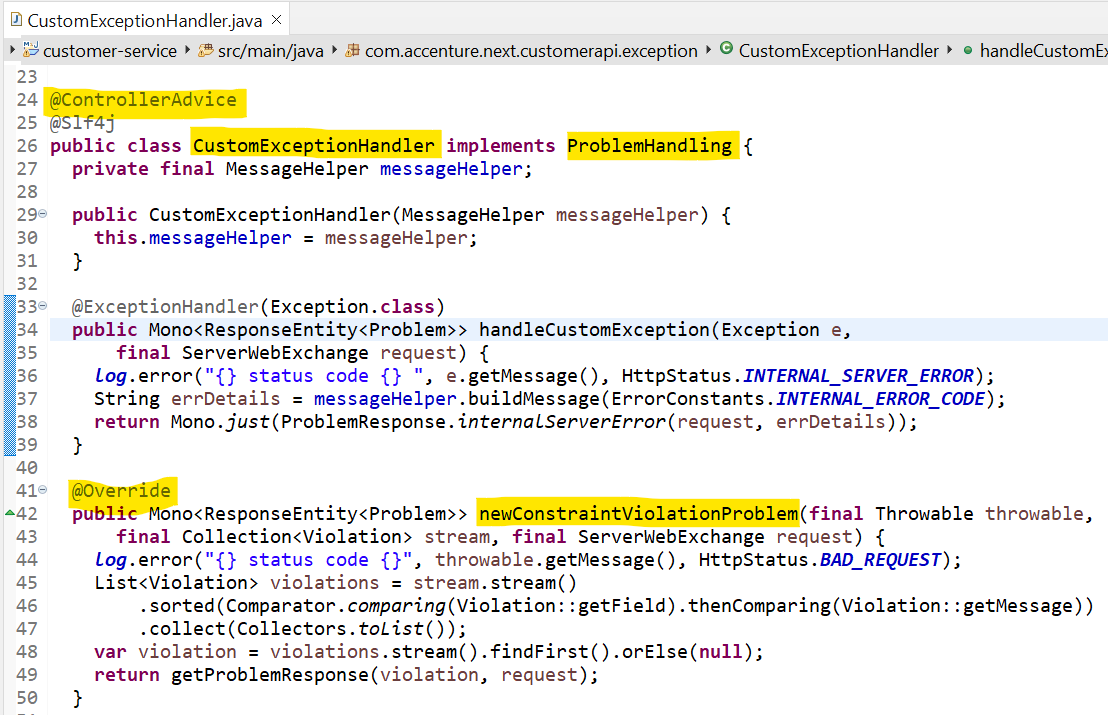
* **Produce application/problem+json responses using the Problem Spring Web library. This library helps us to avoid repetitive tasks related to error handling.**
* **By integrating Problem Spring Web into our Spring Boot application, we can simplify the way we handle exceptions within our project and generate responses accordingly.**

**ProblemConfiguration class: define a generic rule for an exception.**

**Graphical user interface, text, application

Description automatically generated**

**CustomExceptionHandler class: This is advice/interceptor class that implements ProblemHandling interface to handle exceptions.**

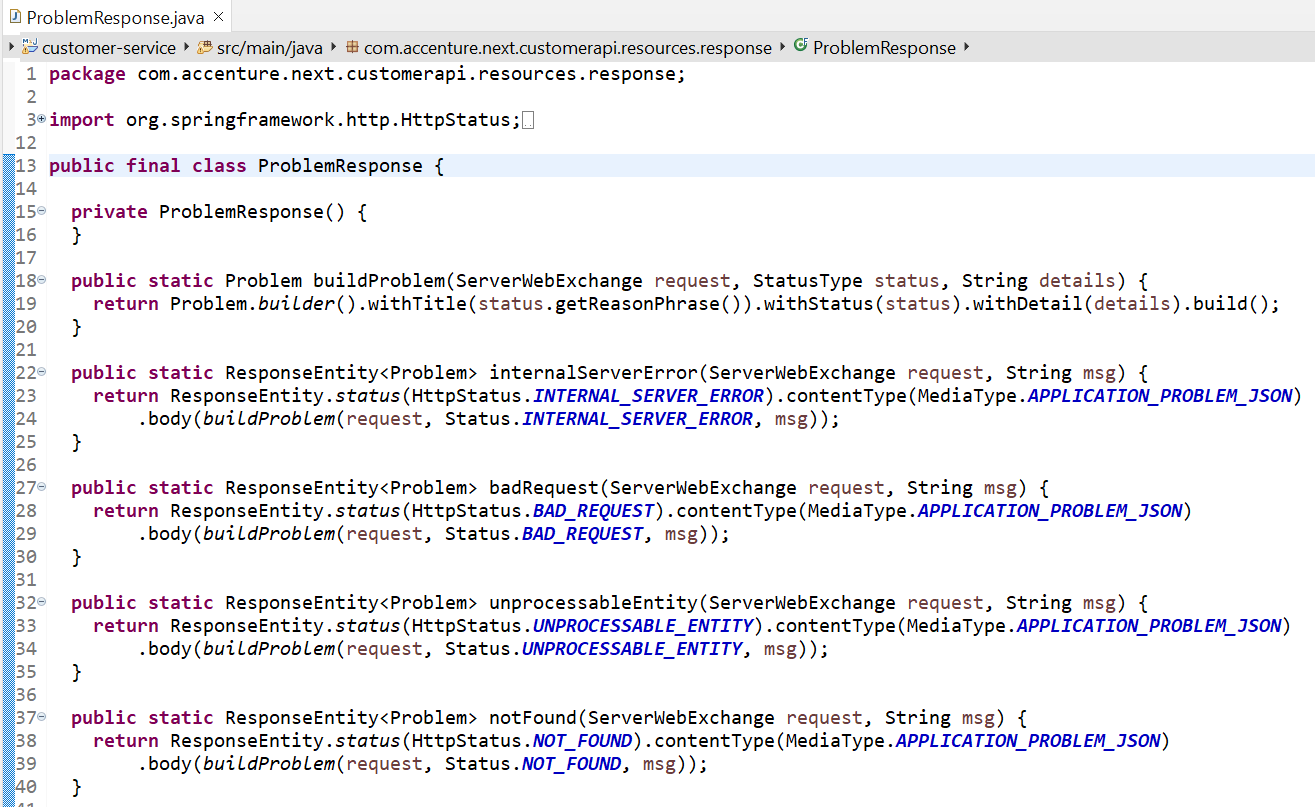
****

**Graphical user interface, text, application

Description automatically generated**

**Problem Response class:**

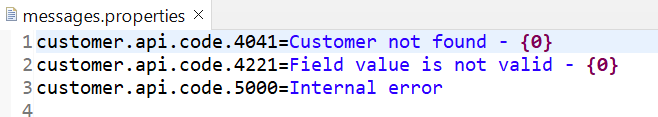
* **Preparing error responses with custom fields like Error title, status, and code.**

****

**Text

Description automatically generated**

**Message.properties file: define all the error messages in key-value format.**

****

**Local.env file: Define PostgreSQL database and logging configuration that is being picked-up when we run application using docker desktop.**

**Reactive Application Up & Running:**

**We can run applications either using spring boot main class or docker desktop.To run docker-compose locally, run the following commands**

* **cp local.env .env**
* **docker-compose up -d --build**
* **To stop running containers, run following commands**
* **docker-compose down**

**Docker container images**

Graphical user interface, text, application, email

Description automatically generated

**Application log inside docker container**

Graphical user interface, text

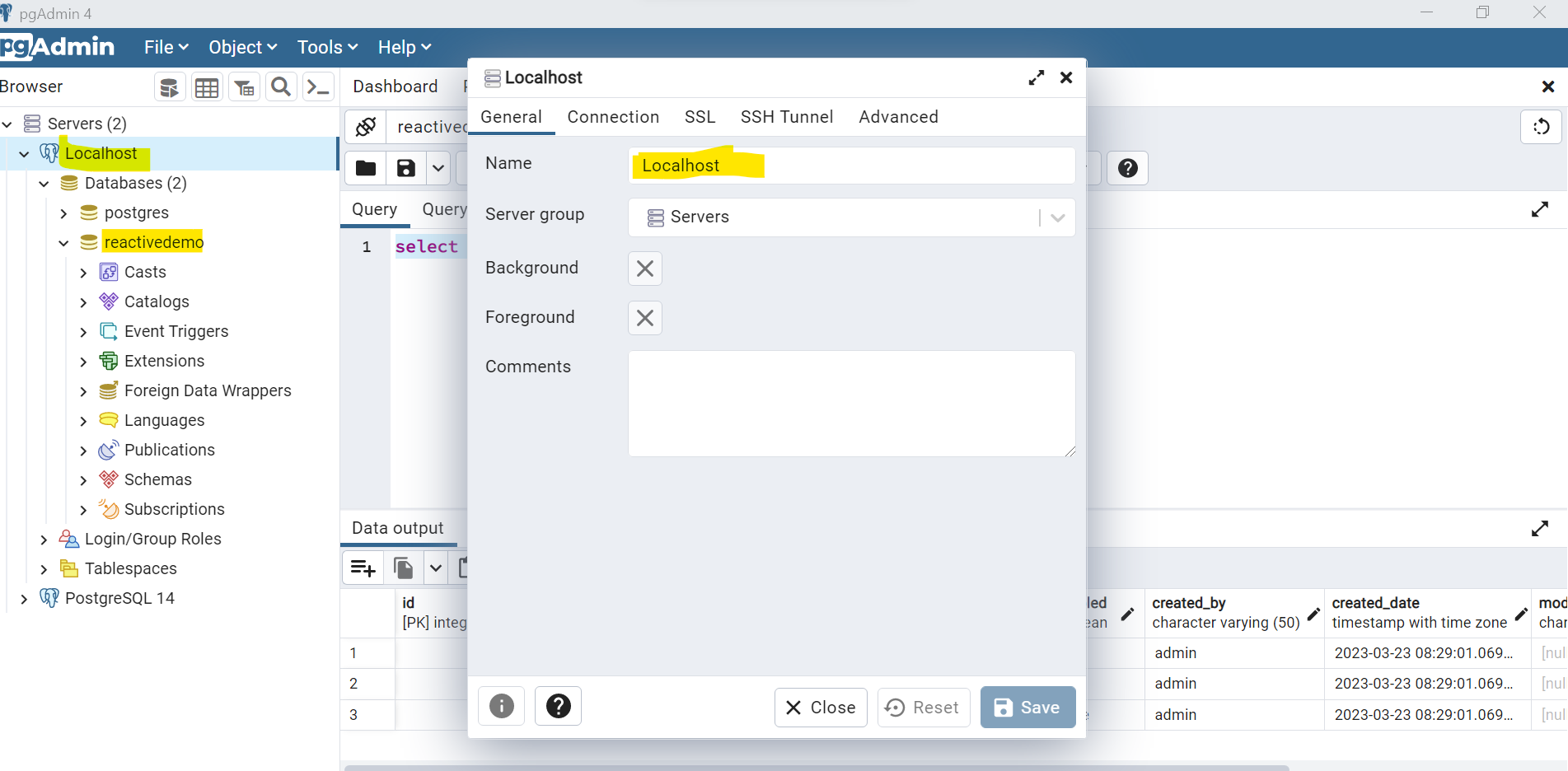
Description automatically generated

**Response using postman collection, application is running on docker 7070 port**

Graphical user interface, text, application

Description automatically generated

**Connect running postgres images with your local pgAdmin. Postgres port in docker container should be different from running in your local machine.**



Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated