श्रवण,मनन,निदिध्यासन

Development Overview

Build microservice from scratch

Spring Hibernate

Vaibhav Zodge

7020616260

info.dnyanyog@gmail.com

https://www.dnyanyog.org

https://github.com/zodgevaibhav

श्रवण,मनन,निदिध्यासन

Why should you listen to me ?

📞 7020616260

github.com/zodgevaibhav

https://in.linkedin.com/in/vaibhav-zodge-679b6a81

Vaibhav Zodge

14+ Years of Experience in IT
Manual Testing > Automation Specialist > Test Managers > Test Automation Architect

6+ Years of Experience in Teaching (Local to Global)
Test Engineering, Selenium, Java, Python, C#, Unix, DevOps
Trained Fresher to 15+ years of experienced students

Coding using 11+ Programming Languages
Many of projects open sourced on gitHub

Conducted Webinar and Workshop for many colleges
Test Engineering, Test Automation, Cloud Architecture, Programming Fundamentals
Sinhgad, VIT, NMIET, MIT College

Associated with renowned institutions
Ex. Profound Edutech Hadapsar and Founder of Dnyanyog Education, Wagholi

DNYANYOG EDUCATION

Spring Feature

Dependency Injection

Aspect Oriented Programming

Auto Configuration (from dependencies)

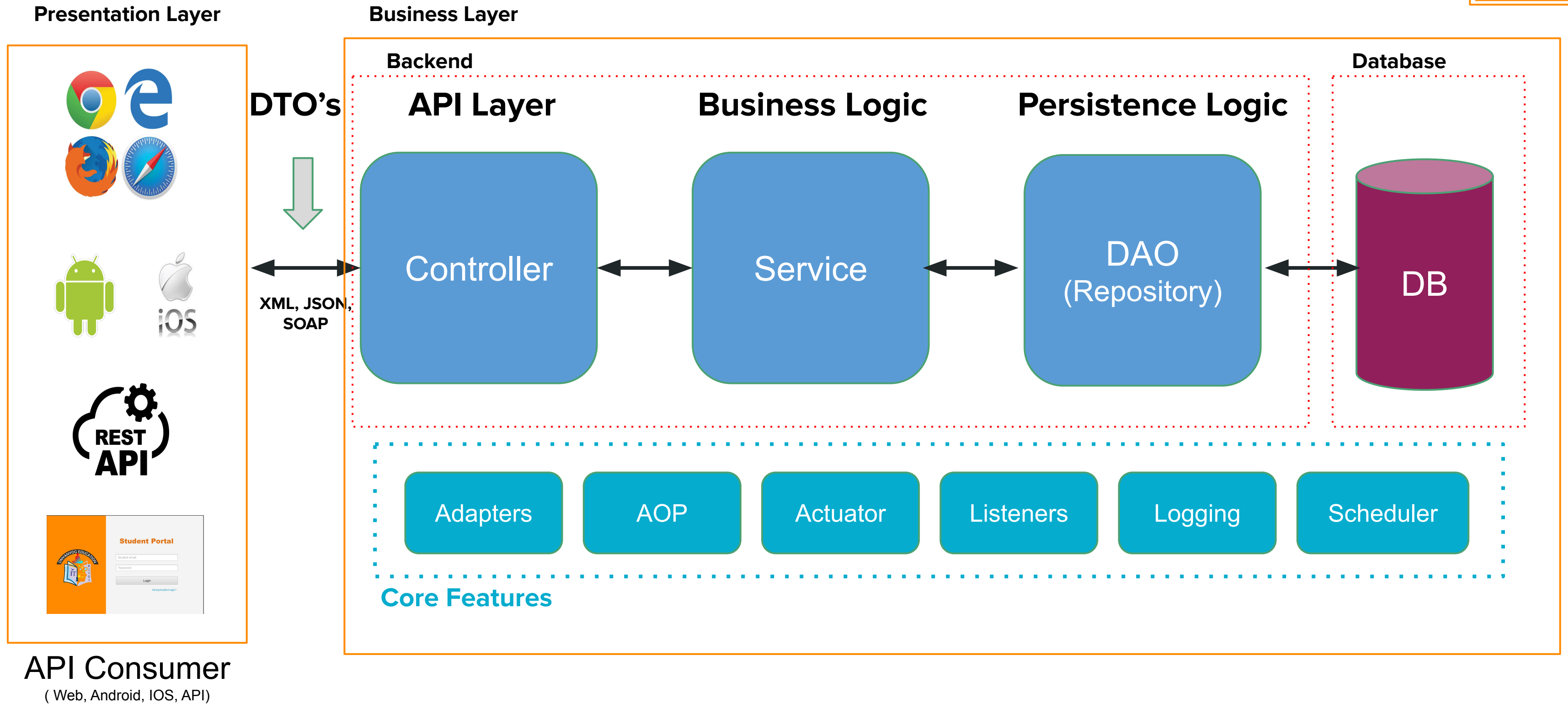
Component Scan

User Defined Configuration

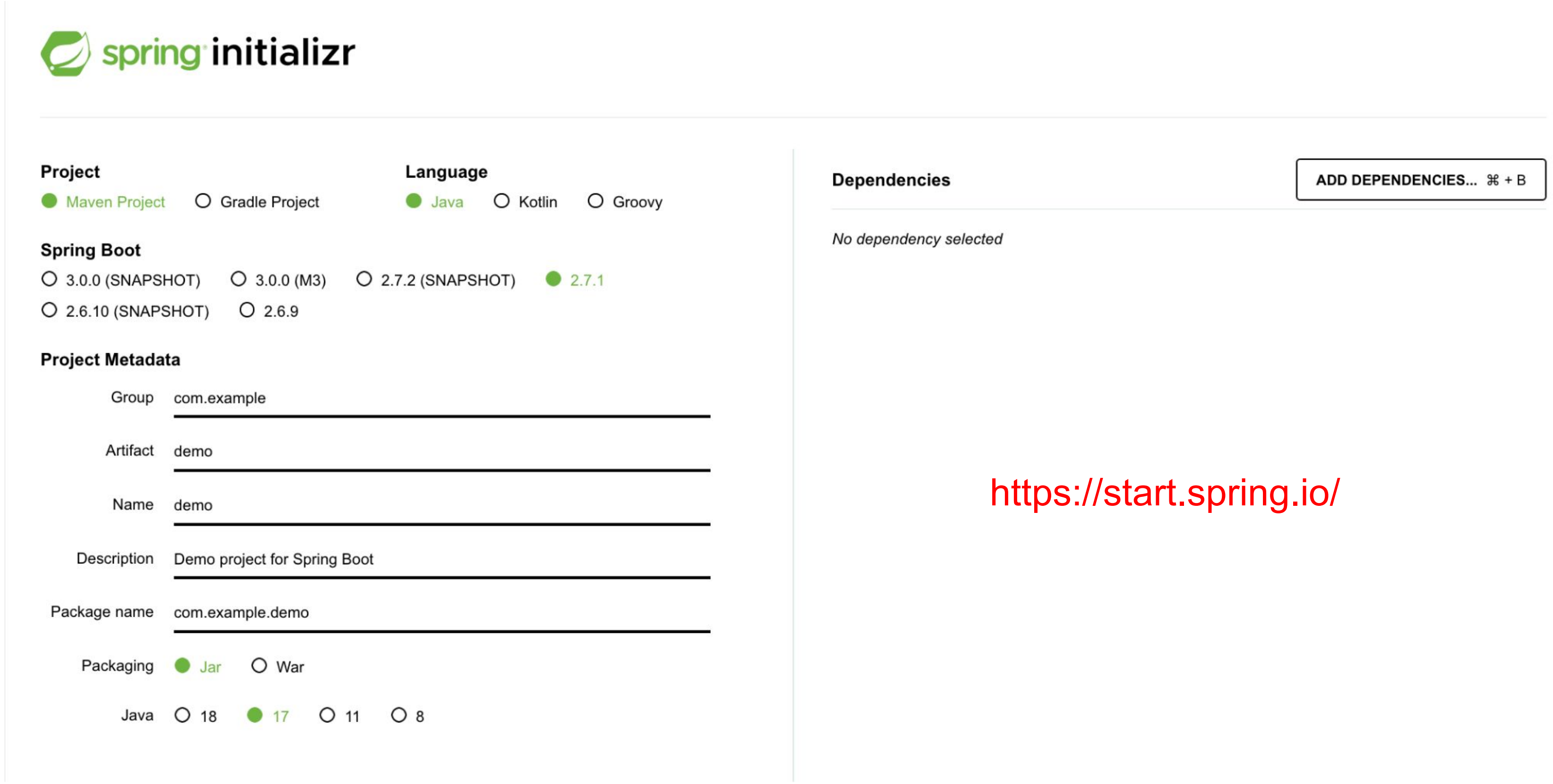
Actuator (Monitoring)

Security

Spring Boot Application Architecture & Flow



Spring Initializer



spring initializr

Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M3) 2.7.2 (SNAPSHOT) [✓ 2.7.1]
2.6.10 (SNAPSHOT) 2.6.9

Project Metadata

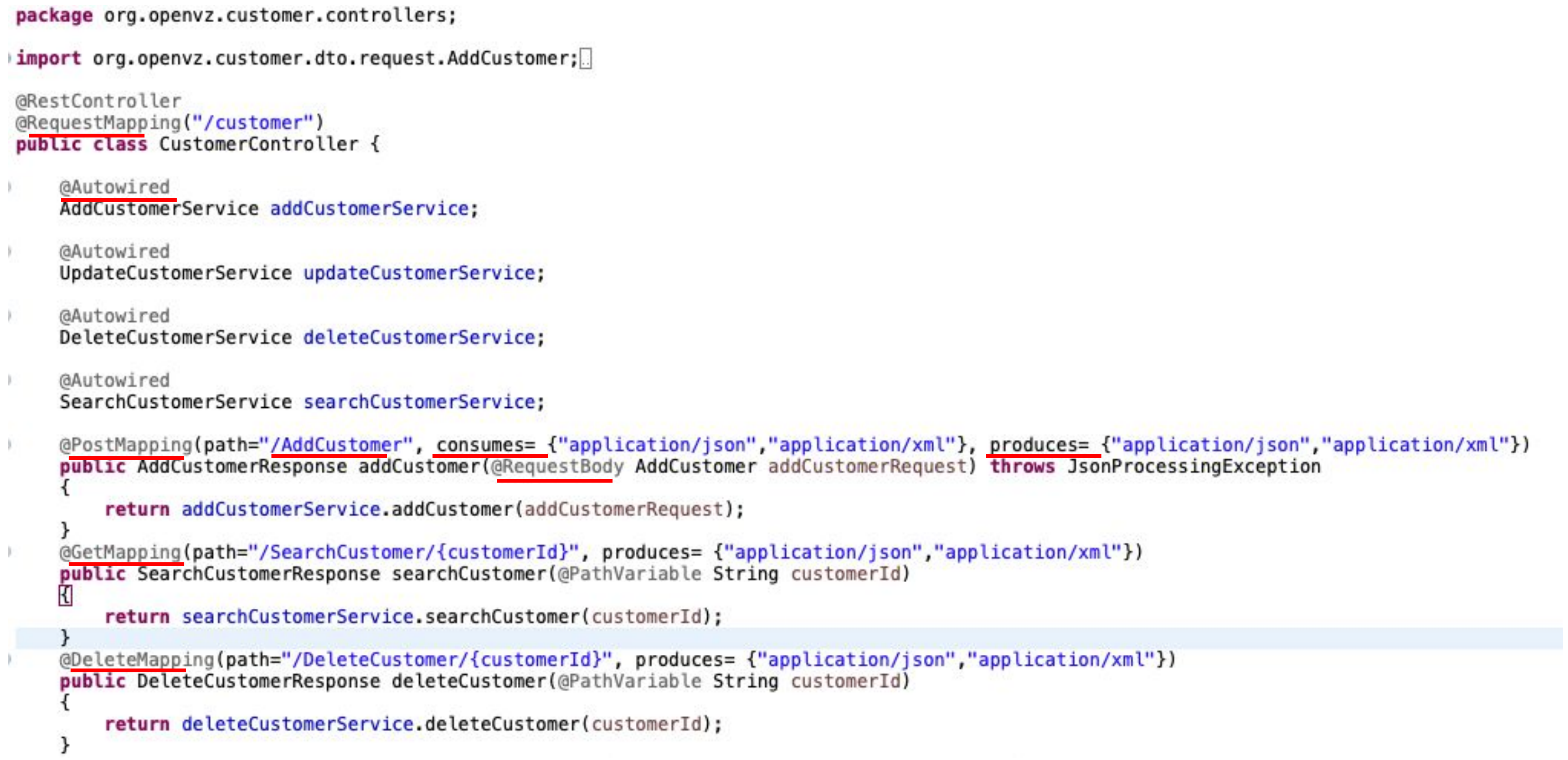
Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	18 [✓ 17] 11 8

Dependencies ADD DEPENDENCIES... ⌘ + B

No dependency selected

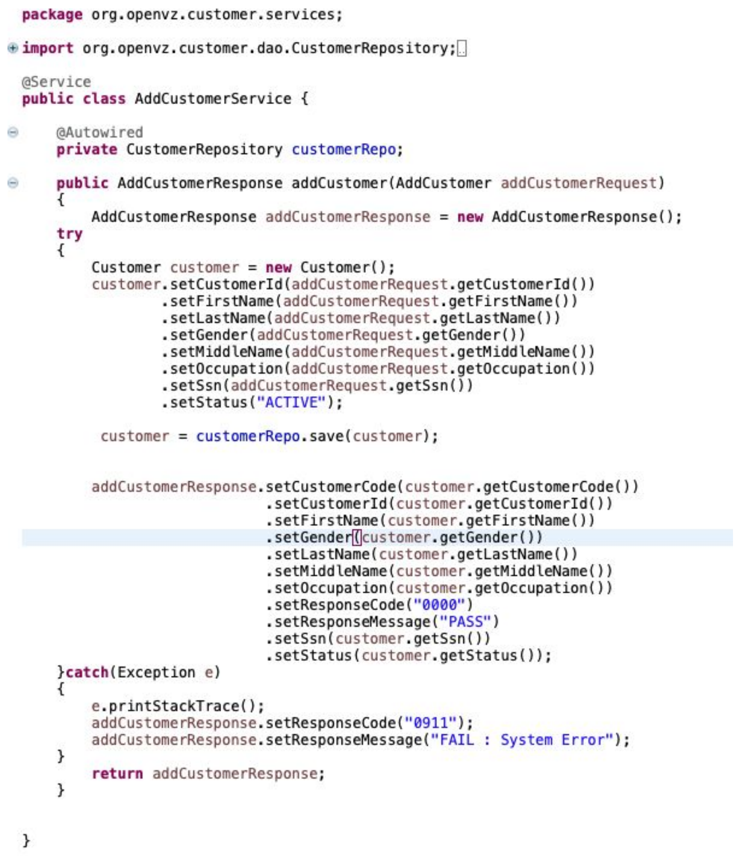
https://start.spring.io/

Controller



```
package org.openvz.customer.controllers;

import org.openvz.customer.dto.request.AddCustomer;

@RestController
@RequestMapping("/customer")
public class CustomerController {

    @Autowired
    AddCustomerService addCustomerService;

    @Autowired
    UpdateCustomerService updateCustomerService;

    @Autowired
    DeleteCustomerService deleteCustomerService;

    @Autowired
    SearchCustomerService searchCustomerService;

    @PostMapping(path="/AddCustomer", consumes= {"application/json","application/xml"}, produces= {"application/json","application/xml"})
    public AddCustomerResponse addCustomer(@RequestBody AddCustomer addCustomerRequest) throws JsonProcessingException
    {
        return addCustomerService.addCustomer(addCustomerRequest);
    }
    @GetMapping(path="/SearchCustomer/{customerId}", produces= {"application/json","application/xml"})
    public SearchCustomerResponse searchCustomer(@PathVariable String customerId)
    {
        return searchCustomerService.searchCustomer(customerId);
    }
    @DeleteMapping(path="/DeleteCustomer/{customerId}", produces= {"application/json","application/xml"})
    public DeleteCustomerResponse deleteCustomer(@PathVariable String customerId)
    {
        return deleteCustomerService.deleteCustomer(customerId);
    }
```

Service



```
package org.openvz.customer.services;

import org.openvz.customer.dao.CustomerRepository;

@Service
public class AddCustomerService {

    @Autowired
    private CustomerRepository customerRepo;

    public AddCustomerResponse addCustomer(AddCustomer addCustomerRequest)
    {
        AddCustomerResponse addCustomerResponse = new AddCustomerResponse();
    try
    {
        Customer customer = new Customer();
        customer.setCustomerId(addCustomerRequest.getCustomerId())
                .setFirstName(addCustomerRequest.getFirstName())
                .setLastName(addCustomerRequest.getLastName())
                .setGender(addCustomerRequest.getGender())
                .setMiddleName(addCustomerRequest.getMiddleName())
                .setOccupation(addCustomerRequest.getOccupation())
                .setSsn(addCustomerRequest.getSsn())
                .setStatus("ACTIVE");

         customer = customerRepo.save(customer);


        addCustomerResponse.setCustomerCode(customer.getCustomerCode())
                           .setCustomerId(customer.getCustomerId())
                           .setFirstName(customer.getFirstName())
                           .setGender(customer.getGender())
                           .setLastName(customer.getLastName())
                           .setMiddleName(customer.getMiddleName())
                           .setOccupation(customer.getOccupation())
                           .setResponseCode("0000")
                           .setResponseMessage("PASS")
                           .setSsn(customer.getSsn())
                           .setStatus(customer.getStatus());
    }catch(Exception e)
    {
        e.printStackTrace();
        addCustomerResponse.setResponseCode("0911");
        addCustomerResponse.setResponseMessage("FAIL : System Error");
    }
        return addCustomerResponse;
    }

}
```

Contain Business Logic

Consumed by Controller

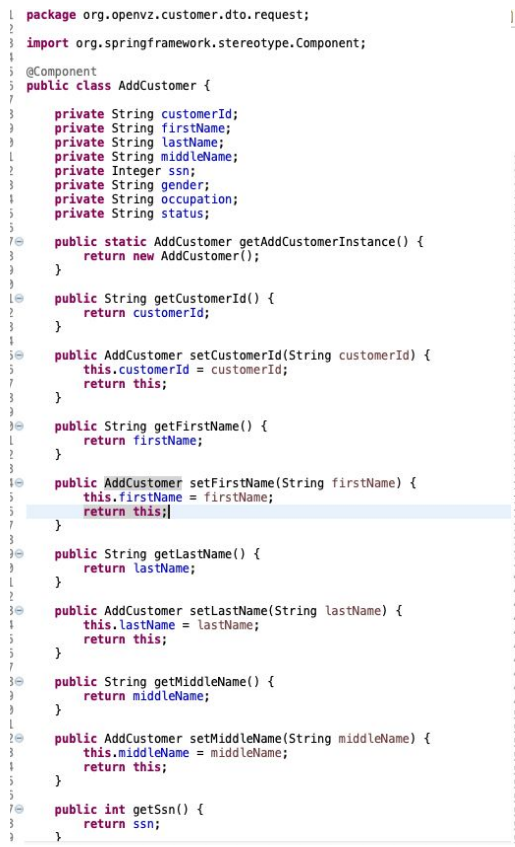
Annotated with @Service (Dependency Injection)

DAO, Entities, Utilities can be used to server

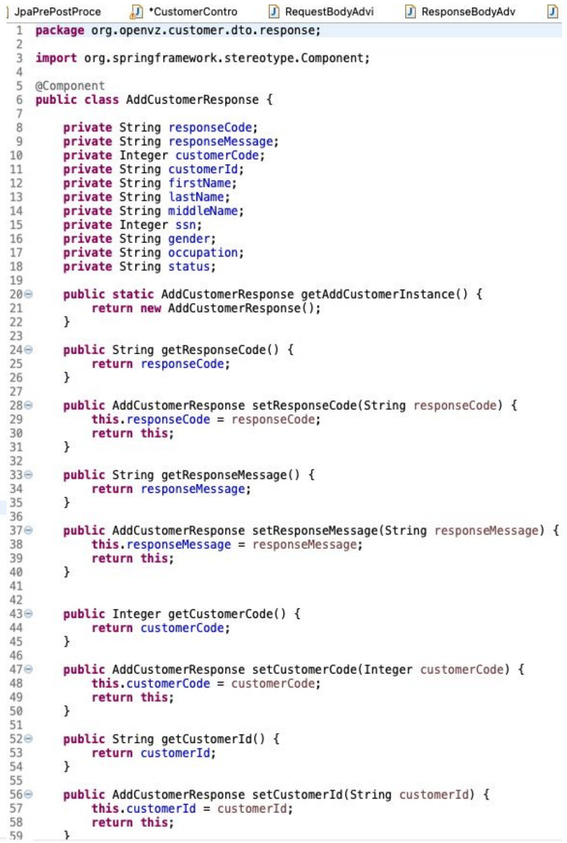
Should not throw exception

IN & OUT are generally DTO’s

Data Transfer Object (DTO)



```java
package org.openvz.customer.dto.request;

import org.springframework.stereotype.Component;

@Component
public class AddCustomer {

    private String customerId;
    private String firstName;
    private String lastName;
    private String middleName;
    private Integer ssn;
    private String gender;
    private String occupation;
    private String status;

    public static AddCustomer getAddCustomerInstance() {
        return new AddCustomer();
    }

    public String getCustomerId() {
        return customerId;
    }

    public AddCustomer setCustomerId(String customerId) {
        this.customerId = customerId;
        return this;
    }

    public String getFirstName() {
        return firstName;
    }

    public AddCustomer setFirstName(String firstName) {
        this.firstName = firstName;
        return this;
    }

    public String getLastName() {
        return lastName;
    }

    public AddCustomer setLastName(String lastName) {
        this.lastName = lastName;
        return this;
    }

    public String getMiddleName() {
        return middleName;
    }

    public AddCustomer setMiddleName(String middleName) {
        this.middleName = middleName;
        return this;
    }

    public int getSsn() {
        return ssn;
    }
```



```java
package org.openvz.customer.dto.response;

import org.springframework.stereotype.Component;

@Component
public class AddCustomerResponse {

    private String responseCode;
    private String responseMessage;
    private Integer customerCode;
    private String customerId;
    private String firstName;
    private String lastName;
    private String middleName;
    private Integer ssn;
    private String gender;
    private String occupation;
    private String status;

    public static AddCustomerResponse getAddCustomerInstance() {
        return new AddCustomerResponse();
    }

    public String getResponseCode() {
        return responseCode;
    }

    public AddCustomerResponse setResponseCode(String responseCode) {
        this.responseCode = responseCode;
        return this;
    }

    public String getResponseMessage() {
        return responseMessage;
    }

    public AddCustomerResponse setResponseMessage(String responseMessage) {
        this.responseMessage = responseMessage;
        return this;
    }


    public Integer getCustomerCode() {
        return customerCode;
    }

    public AddCustomerResponse setCustomerCode(Integer customerCode) {
        this.customerCode = customerCode;
        return this;
    }

    public String getCustomerId() {
        return customerId;
    }

    public AddCustomerResponse setCustomerId(String customerId) {
        this.customerId = customerId;
        return this;
    }
```

Data Transfer Object

Based on API Contracts/Def (XSD, Proto)

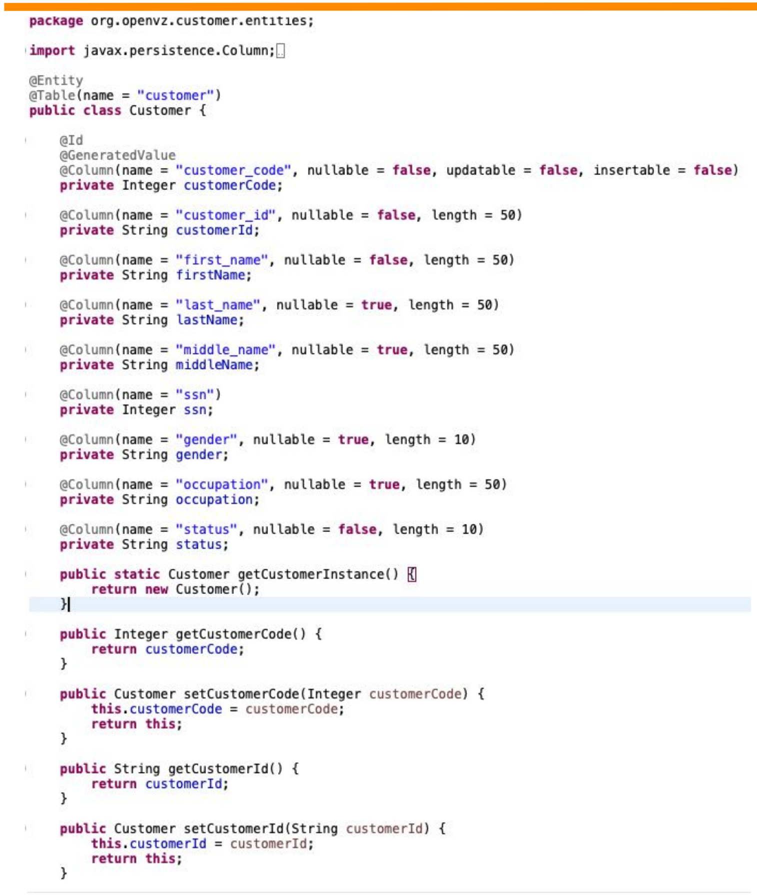
Request and Response DTO

Not same as Entity

Better to use Builder Design Pattern

As it is simple POJO, we can create java object using new keyword

Entity



```java
package org.openvz.customer.entities;

import javax.persistence.Column;

@Entity
@Table(name = "customer")
public class Customer {

    @Id
    @GeneratedValue
    @Column(name = "customer_code", nullable = false, updatable = false, insertable = false)
    private Integer customerCode;

    @Column(name = "customer_id", nullable = false, length = 50)
    private String customerId;

    @Column(name = "first_name", nullable = false, length = 50)
    private String firstName;

    @Column(name = "last_name", nullable = true, length = 50)
    private String lastName;

    @Column(name = "middle_name", nullable = true, length = 50)
    private String middleName;

    @Column(name = "ssn")
    private Integer ssn;

    @Column(name = "gender", nullable = true, length = 10)
    private String gender;

    @Column(name = "occupation", nullable = true, length = 50)
    private String occupation;

    @Column(name = "status", nullable = false, length = 10)
    private String status;

    public static Customer getCustomerInstance() {
        return new Customer();
    }

    public Integer getCustomerCode() {
        return customerCode;
    }

    public Customer setCustomerCode(Integer customerCode) {
        this.customerCode = customerCode;
        return this;
    }

    public String getCustomerId() {
        return customerId;
    }

    public Customer setCustomerId(String customerId) {
        this.customerId = customerId;
        return this;
    }
```

Reflection of Database Table

DB First Approach becomes easy

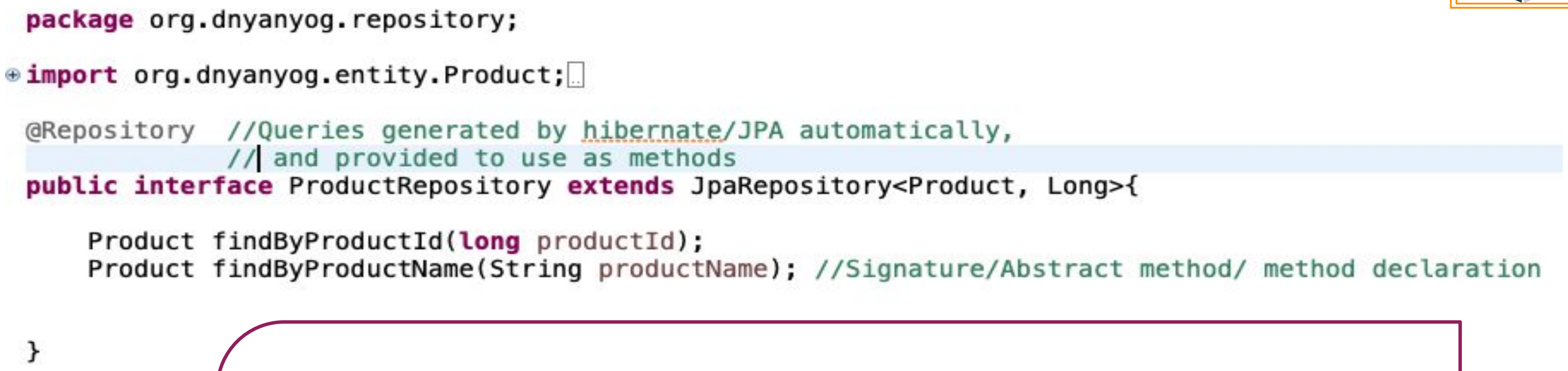
JPA Interface should be referred

Should contain DB Validation, Constraints, Joins

Better to use Builder Design Pattern

Annotated with @Entity (Dependency Injection)

Data Access Object (DAO)



```java
package org.dnyanyog.repository;

import org.dnyanyog.entity.Product;

@Repository  //Queries generated by hibernate/JPA automatically,
             //| and provided to use as methods
public interface ProductRepository extends JpaRepository<Product, Long>{

    Product findByProductId(long productId);
    Product findByProductName(String productName); //Signature/Abstract method/ method declaration


}
```

Data Access Object

Use to access data from DB

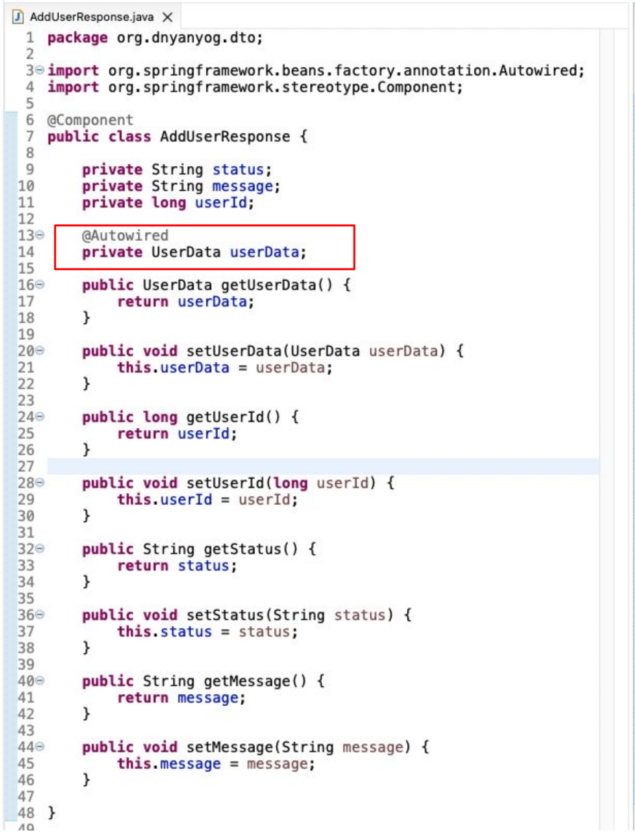
Contains many inbuilt methods to access data

Consider it like hibernate writing queries for us

Custom queries possible (HQL)

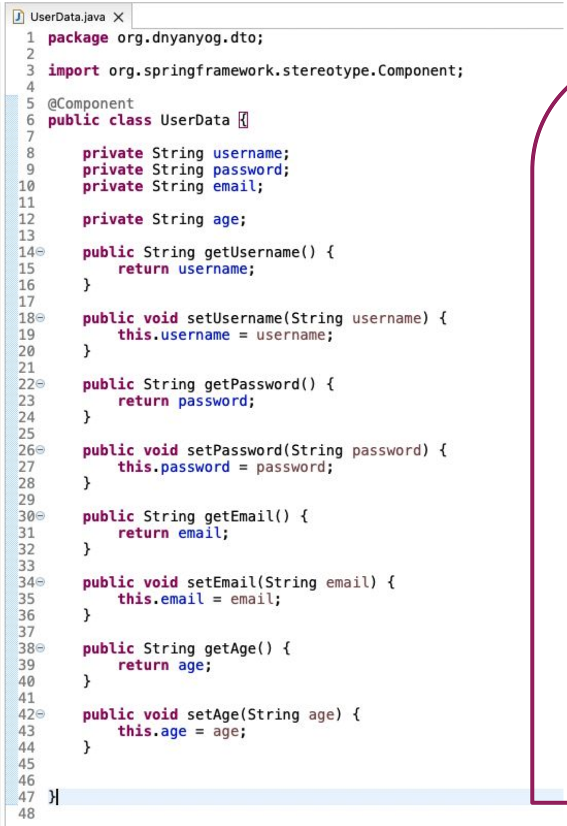
Should Refer JpaRepository

DTO : Nested Objects



AddUserResponse.java

```java
package org.dnyanyog.dto;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class AddUserResponse {

    private String status;
    private String message;
    private long userId;

    @Autowired
    private UserData userData;

    public UserData getUserData() {
        return userData;
    }

    public void setUserData(UserData userData) {
        this.userData = userData;
    }

    public long getUserId() {
        return userId;
    }

    public void setUserId(long userId) {
        this.userId = userId;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```



UserData.java

```java
package org.dnyanyog.dto;

import org.springframework.stereotype.Component;

@Component
public class UserData {

    private String username;
    private String password;
    private String email;

    private String age;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }


}
```

Modularity: Nested objects helps to encapsulating related complex data within a single DTO

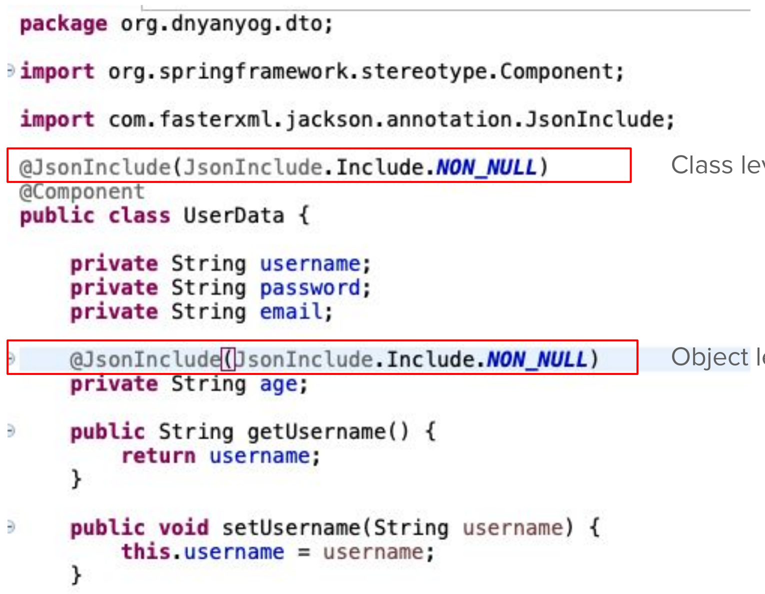
Clarity and Readability: Improve readability of complex DTO structure

Reduced Payload: It reduce the multiple API calls. But be cautious about embedding payloads (Single Responsibility Principle)

Sample Response:

```json
{
    "status": "Success",
    "message": "User found",
    "userId": 353,
    "userData": {
        "username": "gkulkarni",
        "password": "admin123",
        "email": "gkulkarni@gmail.com",
        "age": null

    }
}
```

DTO : Handling null or empty data



```java
package org.dnyanyog.dto;

import org.springframework.stereotype.Component;

import com.fasterxml.jackson.annotation.JsonInclude;

@JsonInclude(JsonInclude.Include.NON_NULL)
@Component
public class UserData {

    private String username;
    private String password;
    private String email;

    @JsonInclude(JsonInclude.Include.NON_NULL)
    private String age;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

Class level annotation

Object level annotation

@JsonInclude: Provides the data inclusion criteria during serialization.

Include.NON_NULL: Helps to send value filled nodes in response and skip with empty or null

JsonInclude.Include.ALWAYS: As name indicates it include all the node even if there is no data

Sample Response:

```
{
   "status": "Success",
   "message": "User found",
   "userId": 353,
   "userData": {
        "username": "gkulkarni",
        "password": "admin123",
        "email":
"gkulkarni@gmail.com "
   }
}
```

Spring Configuration

```java
@Configuration
public class JacksonConfig {

    @Bean
    public ObjectMapper objectMapper() {
        return new ObjectMapper()
                .setSerializationInclusion(JsonInclude.Include.NON_NULL);
    }

}

@Configuration
@ComponentScan(basePackages = "com.example")
class AppConfig {

}

@Configuration
@Import({ DatabaseConfig.class, SecurityConfig.class })
class ImportOtherConfigs {
    // This configuration imports the
    // bean definitions from other configurations.
}

@Configuration
@Profile("development")
class DevelopmentConfig {

    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

@Configuration: Represent the class have one or more custom beans (Object).

@Bean: Represents that the object being returned by the method should be managed by Spring.

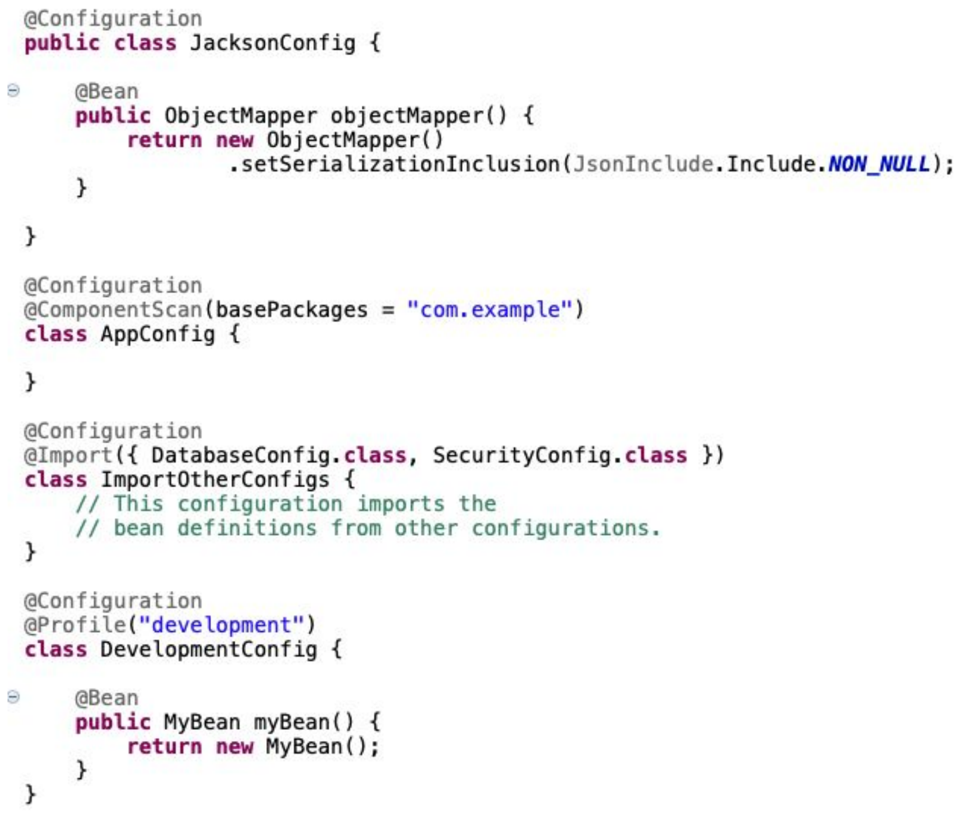
Bean created in conjunction with @Configuration are considered as “full” mode

Bean created in without @Configuration are considered as “lite” mode

When class annotated as @**Configuration** then certain features of Spring get’s enabled

- Conditional Bean Creation
- Inter-bean Dependencies
- Aspect-Oriented Programming (AOP)
- Scoped Beans
- Importing Other Configurations
- Profile-Specific Configurations

Custom Beans



```
@Configuration
public class JacksonConfig {

    @Bean
    public ObjectMapper objectMapper() {
        return new ObjectMapper()
                .setSerializationInclusion(JsonInclude.Include.NON_NULL);
    }

}

@Configuration
@ComponentScan(basePackages = "com.example")
class AppConfig {

}

@Configuration
@Import({ DatabaseConfig.class, SecurityConfig.class })
class ImportOtherConfigs {
    // This configuration imports the
    // bean definitions from other configurations.
}

@Configuration
@Profile("development")
class DevelopmentConfig {

    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

@Configuration: Represent the class have one or more custom beans (Object).

@Bean: Represents that the object being returned by the method should be managed by Spring.

Bean created in conjunction with @Configuration are considered as "full" mode

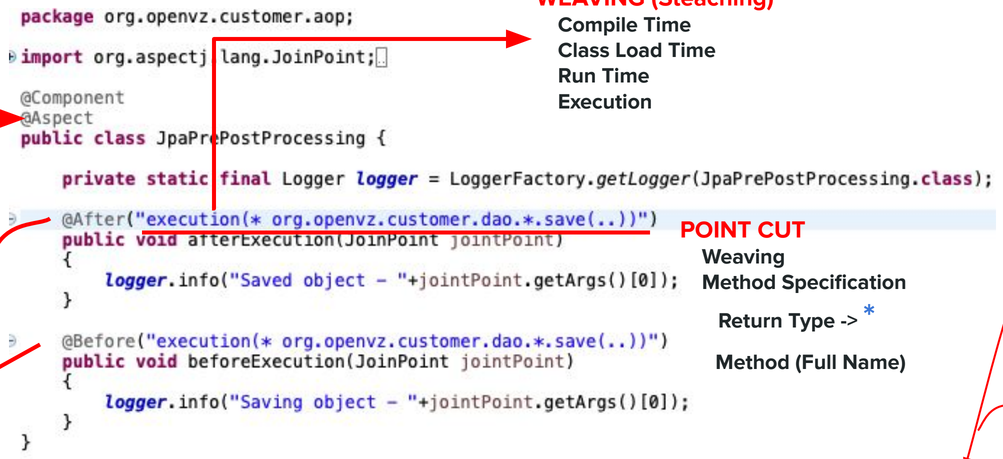
Bean created in without @Configuration are considered as "lite" mode

When class annotated as @**Configuration** then certain features of Spring get's enabled

- Conditional Bean Creation
- Inter-bean Dependencies
- Aspect-Oriented Programming (AOP)
- Scoped Beans
- Importing Other Configurations
- Profile-Specific Configurations

Aspect Oriented Programming

ASPECT (annotation defines it's not just pojo class it's aspect)

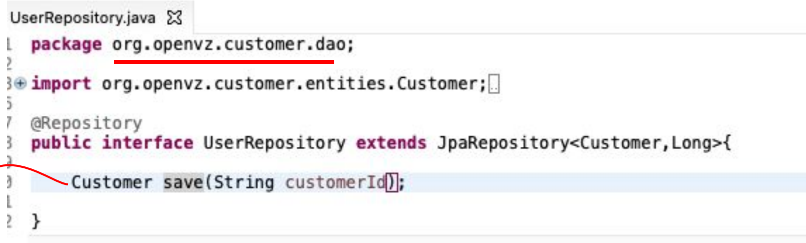


```java
package org.openvz.customer.aop;

import org.aspectj.lang.JoinPoint;

@Component
@Aspect
public class JpaPrePostProcessing {

    private static final Logger logger = LoggerFactory.getLogger(JpaPrePostProcessing.class);

    @After("execution(* org.openvz.customer.dao.*.save(..))")
    public void afterExecution(JoinPoint jointPoint)
    {
        logger.info("Saved object - "+jointPoint.getArgs()[0]);
    }

    @Before("execution(* org.openvz.customer.dao.*.save(..))")
    public void beforeExecution(JoinPoint jointPoint)
    {
        logger.info("Saving object - "+jointPoint.getArgs()[0]);
    }
}
```

WEAVING (Steaching)
- Compile Time
- Class Load Time
- Run Time
- Execution

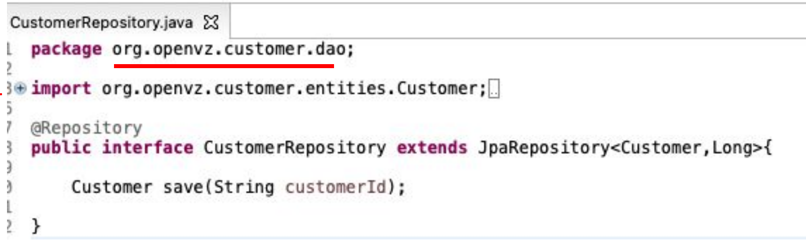
POINT CUT
- Weaving
- Method Specification
  - Return Type -> *
  - Method (Full Name)

ADVICE
- Before
- After
- After Returning
- After Throwing
- Around



UserRepository.java

```java
package org.openvz.customer.dao;

import org.openvz.customer.entities.Customer;

@Repository
public interface UserRepository extends JpaRepository<Customer,Long>{

    Customer save(String customerId);
}
```

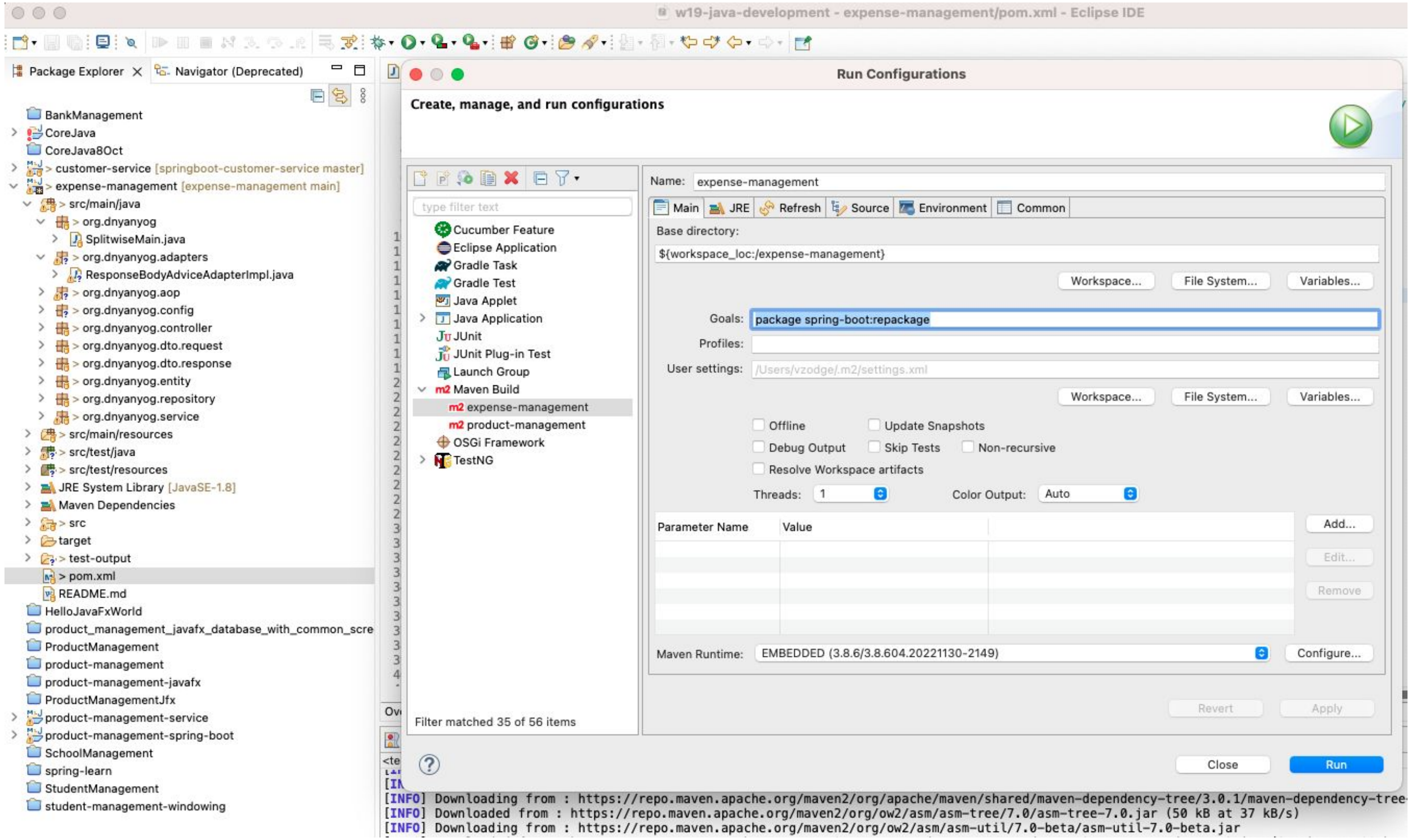


CustomerRepository.java

```java
package org.openvz.customer.dao;

import org.openvz.customer.entities.Customer;

@Repository
public interface CustomerRepository extends JpaRepository<Customer,Long>{

    Customer save(String customerId);
}
```

JOINT POINTS
Point where Aspect will be plugged

Runnable Jar



w19-java-development - expense-management/pom.xml - Eclipse IDE

Package Explorer | Navigator (Deprecated)

- BankManagement
- CoreJava
- CoreJava8Oct
- > customer-service [springboot-customer-service master]
- > expense-management [expense-management main]
  - > src/main/java
    - > org.dnyanyog
      - SplitwiseMain.java
    - > org.dnyanyog.adapters
      - ResponseBodyAdviceAdapterImpl.java
    - > org.dnyanyog.aop
    - > org.dnyanyog.config
    - > org.dnyanyog.controller
    - > org.dnyanyog.dto.request
    - > org.dnyanyog.dto.response
    - > org.dnyanyog.entity
    - > org.dnyanyog.repository
    - > org.dnyanyog.service
  - > src/main/resources
  - > src/test/java
  - > src/test/resources
  - JRE System Library [JavaSE-1.8]
  - Maven Dependencies
  - > src
  - target
  - > test-output
  - > pom.xml
  - README.md
- HelloJavaFxWorld
- product_management_javafx_database_with_common_scre
- ProductManagement
- product-management
- product-management-javafx
- ProductManagementJfx
- product-management-service
- product-management-spring-boot
- SchoolManagement
- spring-learn
- StudentManagement
- student-management-windowing

Run Configurations

Create, manage, and run configurations

type filter text

- Cucumber Feature
- Eclipse Application
- Gradle Task
- Gradle Test
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- Maven Build
  - expense-management
  - product-management
- OSGi Framework
- TestNG

Filter matched 35 of 56 items

Name: expense-management

Main | JRE | Refresh | Source | Environment | Common

Base directory: ${workspace_loc:/expense-management}

Goals: package spring-boot:repackage

Profiles:

User settings: /Users/vzodge/.m2/settings.xml

- [ ] Offline
- [ ] Update Snapshots
- [ ] Debug Output
- [ ] Skip Tests
- [ ] Non-recursive
- [ ] Resolve Workspace artifacts

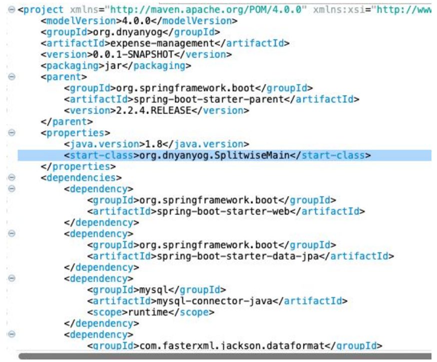
Threads: 1 Color Output: Auto

Parameter Name	Value

Maven Runtime: EMBEDDED (3.8.6/3.8.604.20221130-2149)

Revert | Apply | Close | Run

```
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/3.0.1/maven-dependency-tree-
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/7.0/asm-tree-7.0.jar (50 kB at 37 kB/s)
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/ow2/asm/asm-util/7.0-beta/asm-util-7.0-beta.jar
```



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.dnyanyog</groupId>
    <artifactId>expense-management</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.4.RELEASE</version>
    </parent>
    <properties>
        <java.version>1.8</java.version>
        <start-class>org.dnyanyog.SplitwiseMain</start-class>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
```

Spring Security : Enabled and applied by default

Inbuilt security by just **adding jar** in context

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

By default all endpoints are secured

Needs to add configuration to define secure/private and public config

Default username is *"user"* and password printed in console

We can set default username and password using properties

```
spring.security.user.name=admin
spring.security.user.password=Password@123
```

Spring Security : Configuring http Security (endpoint security)

Add spring configuration to enable SpringWebSecurity

Legacy Implementation

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.authorizeRequests()
                .antMatchers("/public").permitAll();
                .antMatchers("/private").authenticated();
                super.configure(http);
        }
}
```

Lambda Implementation (version 5.7* & above)

```
@Configuration
@EnableWebSecurity
public class SecurityConfig{

@Bean
        public SecurityFilterChain getSecurityFilterChain(HttpSecurity http) throws
Exception {
                return http.authorizeHttpRequests(authz -> {
                        authz.requestMatchers("/public").permitAll()
                                .requestMatchers("/private").authenticated()
                                .anyRequest().authenticated();
                }).formLogin(withDefaults()).build();
        }
```