|  |
| --- |
| diyetisyeninibul  Version 0.0.1-SNAPSHOT  Code analysis |

|  |
| --- |
| **By: Administrator**  **2022-12-20** |

# Content

[Content 1](#_Toc97156258)

[Introduction 2](#_Toc97156259)

[Configuration 2](#_Toc97156260)

[Synthesis 3](#_Toc97156261)

[Analysis Status 3](#_Toc97156262)

[Quality gate status 3](#_Toc97156263)

[Metrics 3](#_Toc97156264)

[Tests 3](#_Toc97156265)

[Detailed technical debt 3](#_Toc97156266)

[Metrics Range 5](#_Toc97156267)

[Volume 5](#_Toc97156268)

[Issues 6](#_Toc97156269)

[Charts 6](#_Toc97156270)

[Issues count by severity and type 8](#_Toc97156271)

[Issues List 8](#_Toc97156272)

[Security Hotspots 9](#_Toc97156273)

[Security hotspots count by category and priority 9](#_Toc97156274)

[Security hotspots List 9](#_Toc97156275)

# Introduction

This document contains results of the code analysis of diyetisyeninibul.

Demo project for Spring Boot

# Configuration

* Quality Profiles
  + Names: Sonar way [Java]; Sonar way [XML];
  + Files: AYROK96TGBe5zV4LFtDo.json; AYROK-EZGBe5zV4LFtNT.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

## Analysis Status

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Security Review | Maintainability |
| B.png | **D.png** | **E.png** | **A.png** |

## Quality gate status

|  |  |
| --- | --- |
| Quality Gate Status | **OK.png** |

|  |  |
| --- | --- |
| Metric | Value |
| Reliability Rating on New Code | OK |
| Security Rating on New Code | OK |
| Maintainability Rating on New Code | OK |

## Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coverage | Duplication | Comment  density | Median number of lines of code per file | Adherence to coding standard |
| 0.0 % | **0.0 %** | **0.4 %** | **54.5** | **99.4 %** |

## Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total | Success Rate | Skipped | Errors | Failures |
| 1 | **100.0 %** | **0** | **0** | **0** |

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| 0d 0h 15min | 0d 0h 40min | 0d 2h 39min | 0d 3h 34min |

## Metrics Range

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 0.0 | 0.0 | 9.0 | 0.0 | 0.0 | 0.0 |
| Max | 96.0 | 18.0 | 596.0 | 1.3 | 0.0 | 0.0 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| Java | 596 |
| XML | 74 |
| Total | 670 |

# Issues

## Charts

## Issues count by severity and type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type / Severity | INFO | MINOR | MAJOR | CRITICAL | BLOCKER |
| BUG | 0 | 1 | 0 | 0 | 0 |
| VULNERABILITY | 0 | 0 | 0 | 4 | 0 |
| CODE\_SMELL | 0 | 16 | 10 | 2 | 1 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| Return values should not be ignored when they contain the operation status code | When the return value of a function call contains the operation status code, this value should be tested to make sure the operation completed successfully. This rule raises an issue when the return values of the following are ignored: java.io.File operations that return a status code (except mkdirs) Iterator.hasNext() Enumeration.hasMoreElements() Lock.tryLock() non-void Condition.await\* methods CountDownLatch.await(long, TimeUnit) Semaphore.tryAcquire BlockingQueue: offer, remove Noncompliant Code Example public void doSomething(File file, Lock lock) { file.delete(); // Noncompliant // ... lock.tryLock(); // Noncompliant } Compliant Solution public void doSomething(File file, Lock lock) { if (!lock.tryLock()) { // lock failed; take appropriate action } if (!file.delete()) { // file delete failed; take appropriate action } } See CERT, EXP00-J. - Do not ignore values returned by methods CERT, FIO02-J. - Detect and handle file-related errors MITRE, CWE-754 - Improper Check for Unusual Exceptional Conditions | BUG | MINOR | 1 |
| Tests should include assertions |  | CODE\_SMELL | BLOCKER | 1 |
| String literals should not be duplicated | Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: public void run() { prepare("action1"); // Noncompliant - "action1" is duplicated 3 times execute("action1"); release("action1"); } @SuppressWarning("all") // Compliant - annotations are excluded private void method1() { /\* ... \*/ } @SuppressWarning("all") private void method2() { /\* ... \*/ } public String method3(String a) { System.out.println("'" + a + "'"); // Compliant - literal "'" has less than 5 characters and is excluded return ""; // Compliant - literal "" has less than 5 characters and is excluded } Compliant Solution private static final String ACTION\_1 = "action1"; // Compliant public void run() { prepare(ACTION\_1); // Compliant execute(ACTION\_1); release(ACTION\_1); } Exceptions To prevent generating some false-positives, literals having less than 5 characters are excluded. | CODE\_SMELL | CRITICAL | 2 |
| Methods should not have too many parameters | A long parameter list can indicate that a new structure should be created to wrap the numerous parameters or that the function is doing too many things. Noncompliant Code Example With a maximum number of 4 parameters: public void doSomething(int param1, int param2, int param3, String param4, long param5) { ... } Compliant Solution public void doSomething(int param1, int param2, int param3, String param4) { ... } Exceptions Methods annotated with : Spring’s @RequestMapping (and related shortcut annotations, like @GetRequest) JAX-RS API annotations (like @javax.ws.rs.GET) Bean constructor injection with @org.springframework.beans.factory.annotation.Autowired CDI constructor injection with @javax.inject.Inject @com.fasterxml.jackson.annotation.JsonCreator may have a lot of parameters, encapsulation being possible. Such methods are therefore ignored. | CODE\_SMELL | MAJOR | 1 |
| Utility classes should not have public constructors | Utility classes, which are collections of static members, are not meant to be instantiated. Even abstract utility classes, which can be extended, should not have public constructors. Java adds an implicit public constructor to every class which does not define at least one explicitly. Hence, at least one non-public constructor should be defined. Noncompliant Code Example class StringUtils { // Noncompliant public static String concatenate(String s1, String s2) { return s1 + s2; } } Compliant Solution class StringUtils { // Compliant private StringUtils() { throw new IllegalStateException("Utility class"); } public static String concatenate(String s1, String s2) { return s1 + s2; } } Exceptions When class contains public static void main(String[] args) method it is not considered as utility class and will be ignored by this rule. | CODE\_SMELL | MAJOR | 1 |
| Sections of code should not be commented out | Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. | CODE\_SMELL | MAJOR | 2 |
| "java.nio.Files#delete" should be preferred | When java.io.File#delete fails, this boolean method simply returns false with no indication of the cause. On the other hand, when java.nio.file.Files#delete fails, this void method returns one of a series of exception types to better indicate the cause of the failure. And since more information is generally better in a debugging situation, java.nio.file.Files#delete is the preferred option. Noncompliant Code Example public void cleanUp(Path path) { File file = new File(path); if (!file.delete()) { // Noncompliant //... } } Compliant Solution public void cleanUp(Path path) throws NoSuchFileException, DirectoryNotEmptyException, IOException { Files.delete(path); } | CODE\_SMELL | MAJOR | 1 |
| Nullness of parameters should be guaranteed | When using null-related annotations at global scope level, for instance using javax.annotation.ParametersAreNonnullByDefault (from JSR-305) at package level, it means that all the parameters to all the methods included in the package will, or should, be considered Non-null. It is equivalent to annotating every parameter in every method with non-null annotations (such as @Nonnull). The rule raises an issue every time a parameter could be null for a method invocation, where the method is annotated as forbidding null parameters. Noncompliant Code Example @javax.annotation.ParametersAreNonnullByDefault class A { void foo() { bar(getValue()); // Noncompliant - method 'bar' do not expect 'null' values as parameter } void bar(Object o) { // 'o' is by contract expected never to be null // ... } @javax.annotation.CheckForNull abstract Object getValue(); } Compliant Solution Two solutions are possible: The signature of the method is correct, and null check should be done prior to the call. The signature of the method is not coherent and should be annotated to allow null values being passed as parameter @javax.annotation.ParametersAreNonnullByDefault abstract class A { void foo() { Object o = getValue(); if (o != null) { bar(o); // Compliant - 'o' can not be null } } void bar(Object o) { // ... } @javax.annotation.CheckForNull abstract Object getValue(); } or @javax.annotation.ParametersAreNonnullByDefault abstract class A { void foo() { bar(getValue()); } void bar(@javax.annotation.Nullable Object o) { // annotation was missing // ... } @javax.annotation.CheckForNull abstract Object getValue(); } | CODE\_SMELL | MAJOR | 5 |
| Local variables should not be declared and then immediately returned or thrown | Declaring a variable only to immediately return or throw it is a bad practice. Some developers argue that the practice improves code readability, because it enables them to explicitly name what is being returned. However, this variable is an internal implementation detail that is not exposed to the callers of the method. The method name should be sufficient for callers to know exactly what will be returned. Noncompliant Code Example public long computeDurationInMilliseconds() { long duration = (((hours \* 60) + minutes) \* 60 + seconds ) \* 1000 ; return duration; } public void doSomething() { RuntimeException myException = new RuntimeException(); throw myException; } Compliant Solution public long computeDurationInMilliseconds() { return (((hours \* 60) + minutes) \* 60 + seconds ) \* 1000 ; } public void doSomething() { throw new RuntimeException(); } | CODE\_SMELL | MINOR | 6 |
| Composed "@RequestMapping" variants should be preferred | Spring framework 4.3 introduced variants of the @RequestMapping annotation to better represent the semantics of the annotated methods. The use of @GetMapping, @PostMapping, @PutMapping, @PatchMapping and @DeleteMapping should be preferred to the use of the raw @RequestMapping(method = RequestMethod.XYZ). Noncompliant Code Example @RequestMapping(path = "/greeting", method = RequestMethod.GET) // Noncompliant public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) { ... } Compliant Solution @GetMapping(path = "/greeting") // Compliant public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) { ... } | CODE\_SMELL | MINOR | 10 |
| Persistent entities should not be used as arguments of "@RequestMapping" methods | On one side, Spring MVC automatically bind request parameters to beans declared as arguments of methods annotated with @RequestMapping. Because of this automatic binding feature, it’s possible to feed some unexpected fields on the arguments of the @RequestMapping annotated methods. On the other end, persistent objects (@Entity or @Document) are linked to the underlying database and updated automatically by a persistence framework, such as Hibernate, JPA or Spring Data MongoDB. These two facts combined together can lead to malicious attack: if a persistent object is used as an argument of a method annotated with @RequestMapping, it’s possible from a specially crafted user input, to change the content of unexpected fields into the database. For this reason, using @Entity or @Document objects as arguments of methods annotated with @RequestMapping should be avoided. In addition to @RequestMapping, this rule also considers the annotations introduced in Spring Framework 4.3: @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping. Noncompliant Code Example import javax.persistence.Entity; @Entity public class Wish { Long productId; Long quantity; Client client; } @Entity public class Client { String clientId; String name; String password; } import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; @Controller public class WishListController { @PostMapping(path = "/saveForLater") public String saveForLater(Wish wish) { session.save(wish); } @RequestMapping(path = "/saveForLater", method = RequestMethod.POST) public String saveForLater(Wish wish) { session.save(wish); } } Compliant Solution public class WishDTO { Long productId; Long quantity; Long clientId; } import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; @Controller public class PurchaseOrderController { @PostMapping(path = "/saveForLater") public String saveForLater(WishDTO wish) { Wish persistentWish = new Wish(); // do the mapping between "wish" and "persistentWish" [...] session.save(persistentWish); } @RequestMapping(path = "/saveForLater", method = RequestMethod.POST) public String saveForLater(WishDTO wish) { Wish persistentWish = new Wish(); // do the mapping between "wish" and "persistentWish" [...] session.save(persistentWish); } } Exceptions No issue is reported when the parameter is annotated with @PathVariable from Spring Framework, since the lookup will be done via id, the object cannot be forged on client side. See OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures OWASP Top 10 2017 Category A5 - Broken Access Control MITRE, CWE-915 - Improperly Controlled Modification of Dynamically-Determined Object Attributes Two Security Vulnerabilities in the Spring Framework’s MVC by Ryan Berg and Dinis Cruz | VULNERABILITY | CRITICAL | 4 |

# Security Hotspots

## Security hotspots count by category and priority

|  |  |  |  |
| --- | --- | --- | --- |
| Category / Priority | LOW | MEDIUM | HIGH |
| LDAP Injection | 0 | 0 | 0 |
| Object Injection | 0 | 0 | 0 |
| Server-Side Request Forgery (SSRF) | 0 | 0 | 0 |
| XML External Entity (XXE) | 0 | 0 | 0 |
| Insecure Configuration | 0 | 0 | 0 |
| XPath Injection | 0 | 0 | 0 |
| Authentication | 0 | 0 | 0 |
| Weak Cryptography | 0 | 0 | 0 |
| Denial of Service (DoS) | 0 | 0 | 0 |
| Log Injection | 0 | 0 | 0 |
| Cross-Site Request Forgery (CSRF) | 0 | 0 | 3 |
| Open Redirect | 0 | 0 | 0 |
| Permission | 0 | 0 | 0 |
| SQL Injection | 0 | 0 | 0 |
| Encryption of Sensitive Data | 0 | 0 | 0 |
| Traceability | 0 | 0 | 0 |
| Buffer Overflow | 0 | 0 | 0 |
| File Manipulation | 0 | 0 | 0 |
| Code Injection (RCE) | 0 | 0 | 0 |
| Cross-Site Scripting (XSS) | 0 | 0 | 0 |
| Command Injection | 0 | 0 | 0 |
| Path Traversal Injection | 0 | 0 | 0 |
| HTTP Response Splitting | 0 | 0 | 0 |
| Others | 0 | 0 | 0 |

## Security hotspots List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Name | Priority | Severity | Count |
| Cross-Site Request Forgery (CSRF) | Allowing both safe and unsafe HTTP methods is security-sensitive | HIGH | MINOR | 3 |