README

Developed by Beatriz Vaz (1190429)

This folder includes all artifacts developed for the First Part of QESOFT Project.

This file is structured as follows:

1. Introduction

The first phase of the QESOFT project analyzes the project in an operational environment to determine whether the project is reusable.

The main focus are the quality attributes and architectural characteristics that must be considered throughout the study, such as maintenance, performance, safety vulnerabilities, construction compliance and adequate testing, are key issues

In order to identify and select the best materials, this research will take an objective approach.

The paper also describes the project's learning environment, including how to choose a Java service-based application with multiple executable REST controllers or DDD sets with multiple endpoints that can allow for distribution of work among team members between in this application. The software should also be SMART compliant and enable complete data generation.

The project under analysis is the Trebol.

In this particular file, the quality attributes and architectural characteristics that were analysed concern the flow of information of *DataUsersController*.

2. Maintainability

It was used the IntelliJ plugin MetricsTree to measure Coupling and Structural Erosion (CSE) and Size and Complexity (SC).

MetricsTree is an IDE extension that helps to evaluate quantitative properties of java code. It supports the most common sets of metrics at the project, package, class, and method levels. [1]

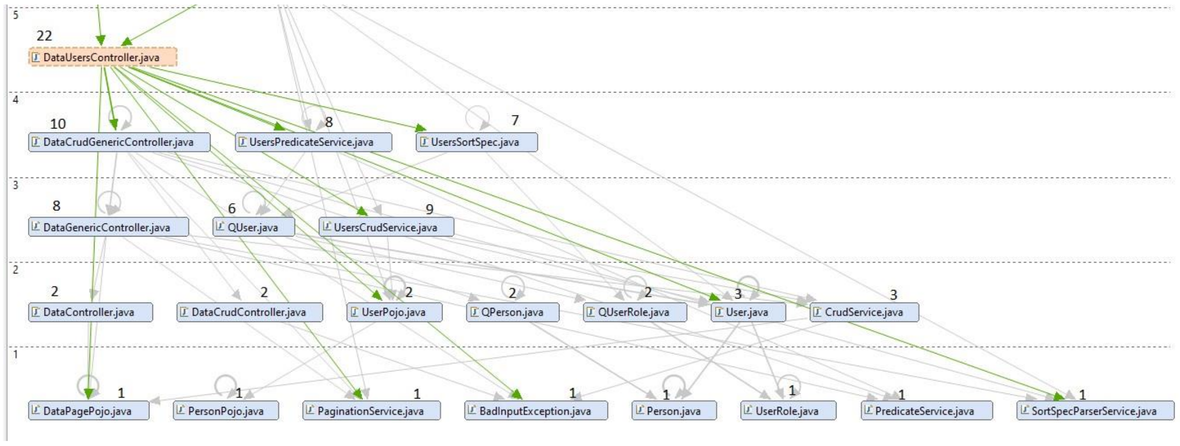
To measure CSE were used the metrics: average component dependency, propagation cost, cyclicity and relative cyclicity and maintainability level.

To measure SC were used the metrics: size metric, cyclomatic complexity and indentation debt.

2.1 Average Component Dependency (ACD)

ACD can be obtained by dividing CCD by the number of components to get the ACD.

This way, it was used the tool Sonargraph to calculate the components to get the ACD and the value of CCD. To do this, in sonargraph it was selected the class DataUsersController and then selected the option to show in graph view. The graph obtained is in the next image.



The number above each class represents the value from the metric Depends Upon. This metric was also available in the tool Sonargraph.

Having all the values from the Depends Upon metric of all components of the class DataUsersController, the value CCD can be calculated as the sum of these values. It was determined that CCD has a value of 94.

As it was said before, ACD can be obtained by dividing CCD by the number of components. This way:

```
ACD = 94/22 = 4,27
```

The maximum value is equal to the number of components, in this case it that would be 22 On average, a component depends on 4,27 components. Comparing to the maximum value, it can be said that in terms of coupling, this number is fairly low.

2.2 Propagation Cost (PC)

The metric propagation cost measures the potential impact a change in a component will cause on other components. A higher propagation cost will likely lead to an increase in complexity of the system therefore making it difficult to maintain.

PC can also be calculated by dividing the ACD once more by the number of nodes (components).

```
PC = 4,27/22 = 0,19
```

Since we are working with a small part of the system, this value shouldn't be concerning.

2.3 Maintainability Level

The Maintainability Index first appeared in 1992 when it was proposed by Paul Oman and Jack Hagemeister at the International Conference on Software Maintenance with the goal of establishing automated software development metrics to guide software related decision making. The Maintainability Index tries to give a holistic view of the relative maintenance burden for different sections of a project by blending together a series of different metrics. [2] Here we use a more modern adaptation of the original formula proposed in 2011 by Microsoft:

```
Maintainability Index = MAX(0,(171 - 5.2 * ln(Halstead Volume) - 0.23 *
(Cyclomatic Complexity) - 16.2 * ln(Lines of Code))*100 / 171)
```

The DataUsersController class has a Maintainability Index of 46,541

About the methods of this class, their maintainability levels are:

Method	Maintainability Index
DataUsersController()	70,2303
create()	71,432
delete()	65,063
getOrderSpecMap()	82,6511
readMany()	73,3612
update()	70,9324

2.4 Size metric

Lines of Code (LoC) per file counts every line that contains actual code and skips empty lines and comment lines.

Total Lines metric counts every single line, including empty lines and comment lines.

Number of Statements verifies the statements, i.e., s a single complete action performed in the source code, usually terminated by a special character or a newline.

In the class DataUsersController it was possible to determine the following values:

- Lines of Code: 68
- Total Lines: 97
- Number of Statements: 9

These values can indicate of how much the component is doing and how complex it is. So, we can say that this component is not complex.

2.5 Cyclomatic complexity

Cyclomatic complexity was proposed in 1976, and it computes the number of different possible execution paths through a method or function, which is also a floor for the number of test cases needed to achieve 100% test coverage.

For this metric it was analyzed the metric Average Complexity that is described as the weighted average modified extended cyclomatic complexity for fully analyzes code.

- Average Complexity: 1,89

As the value calculated is very low we can say that the class DataUsersController is easy to understand and don't have a big risk associated when modifying the class.

3. Performance

Performance testing refers to a set of tests that evaluate how a system behaves and performs. This type of testing focuses on examining the responsiveness, stability, scalability, reliability, speed, and resource usage of your software and infrastructure. The specific data obtained from performance testing can vary depending on the type of test conducted.

To measure the performance it was used Apache JMeter. Apache JMeter is a pure Java, open-source application that enables load testing and performance measurement of functional behavior. Initially developed for testing web applications, JMeter has evolved to accommodate other types of testing as well.

JMeter can evaluate the performance of both static and dynamic resources, including web-based applications. It is capable of simulating a heavy load on a server, a group of servers, or an object to assess its strength or to analyze overall performance under various load types. [3]

3.1 Performance Tests

There are three main types of performance tests commonly used:

- **Load Tests:** These tests are designed to measure the performance of a system under high levels of user traffic or load. They simulate a large number of users or transactions to determine how the system responds.

- **Stress Tests:** Stress testing is similar to load testing but pushes the system beyond normal usage scenarios to test its limits. This involves testing how the system performs under extreme conditions.
- **Soak Tests:** Soak testing is used to evaluate how a system performs over an extended period of time. This type of testing involves subjecting the system to sustained load for several hours, days, or even weeks to observe its performance. Soak tests are especially useful for detecting memory leaks.

3.2 KPIs

KPIs are metrics that allow for measuring the results and success of an organization based on selected parameters that are deemed relevant and important. KPIs are commonly used to evaluate activities and track progress.

In JMeter, there are several recommended KPIs that can be used for load testing, including:

- **Number of Users -** Simulating concurrent active users through virtual users to evaluate if the website can handle heavy loads created by high user traffic.
- **Hits Per Second -** Measuring the average number of requests generated to the target server from users' actions.
- **Errors Per Second -** Identifying the number of errors per second, which can indicate a bottleneck that needs fixing before going live.
- **Response Time -** Measuring the amount of time it takes the system to process a request, which shows how the target site is performing from the user's point of view.
- **Latency -** Measuring the time it takes from just before sending the request to just after receiving the first response, which helps to understand the inherent network delay during transmission of data from client to server.
- **Connect Time -** Measuring the time it takes for the user to connect to the server and the server to respond, including SSL handshake, which is important for isolating SSL performance as a bottleneck.
- **Bytes/s (Throughput) -** Measuring the average bandwidth consumption that is generated by the test per second, which is important for ensuring network interface controllers are performing properly.

The primary KPI chosen by the team was Latency.

To run Jmeter tests, a test plan was designed with the following configuration:

```
.
└── Thread Group/
    ├── Http Request (Login)/
```

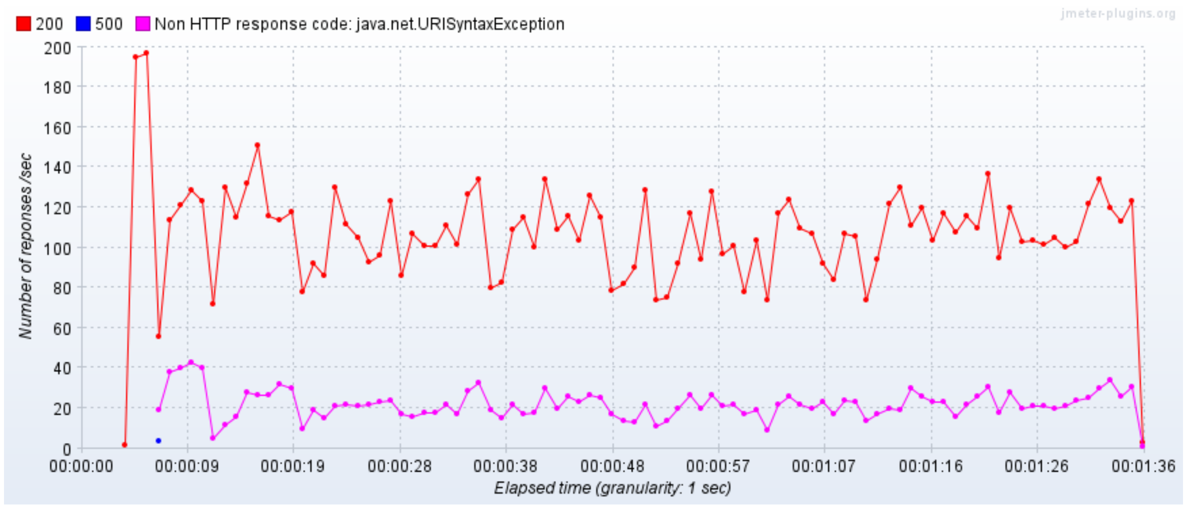
```
│    ├── View Results Tree
│    └── JSR223 PostProcessor
├── Http Request (Get Users)/
│    ├── Http Header Manager
│    └── View Results Tree
├── Http Request (Post Users)/
│    ├── Http Header Manager
│    ├── View Results Tree
│    ├── CSV Data Set Config
│    └── JSR223 PostProcessor
├── Debug Sampler
├── View Results Tree
├── Http Request (Put Users)/
│    ├── Http Header Manager
│    ├── JSR223 PreProcessor
│    └── View Results Tree
├── Debug Sampler
├── View Results Tree
├── jp@gc - Response Codes per Second
├── jp@gc - Transactions per Second
└── Summary Report
```

3.3 Load Test

To do a load test for the application, we'll simulate many users accessing the users page. The thread properties defined will be:

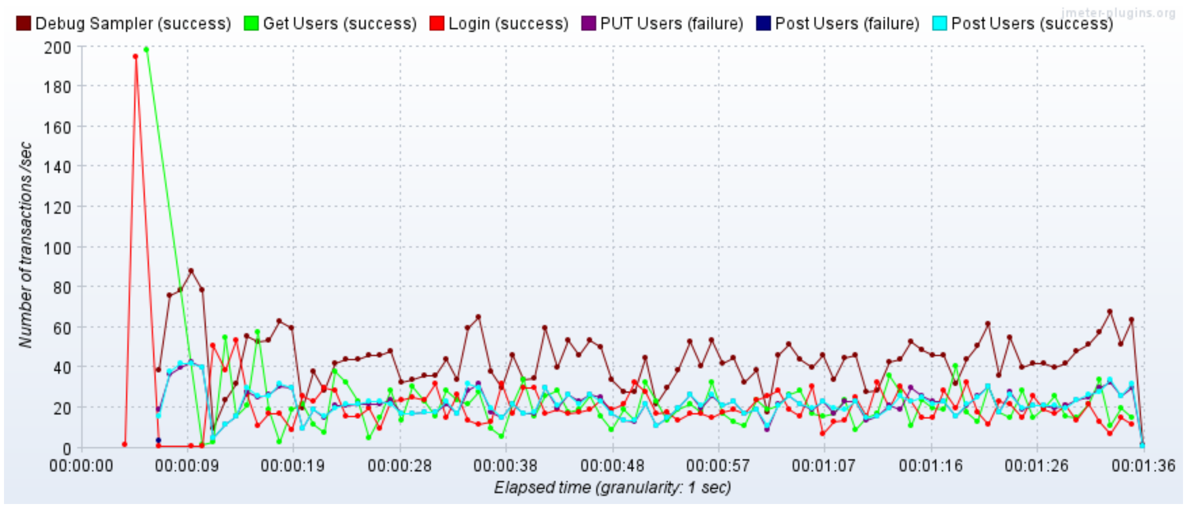
- **Number of Threads (users):** 200
- **Ramp-up period:** 1
- **Loop Count:** 10

Only 1 thread was defined for each test, so it'll only take 1 second to ramp it up. To see the results, a View Results Tree Listener and a Summary Report was used. It was also used a jp@gc - Response Codes per Second and jp@gc - Transactions per Second. These were possible after the download of these plugins.



By this image (jp@gc - Response Codes per Second), it is possible to see that in the same second several requests are being handled. Besides this, we can also see the codes from each request. The line in rose represents the debug sampler.

With the next image is possible to see that there some erros with the request post and put. The errors associated with the request post exits since the application is generating the id of each object beginning at the index 0. This is a problem because there are already users persisted in the database with the ids 1, 2, 3 and 4. The errors observed with the request put exist since these were not defined correctly in jmeter. It was not possible to pass the parameters to the path of the request.



To show the results, we used a Summary Report:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	2000	4152	12	152	1906.	0.000	21.0479	114.1	4.60	555

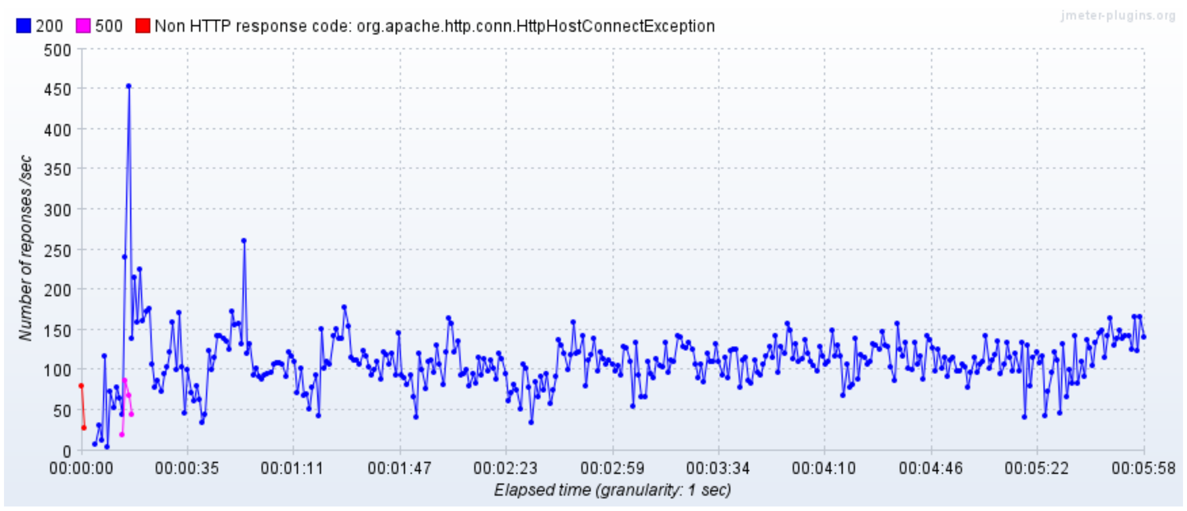
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
			0	23	33	%	8	2		2.0
Get Users	2000	2047	4	10921	1200.72	0.000%	22.29306	18.77	59.74	862.3
Post Users	2000	2629	120	10353	1354.84	0.200%	22.36686	8.70	62.58	398.3
Debug Sampler	4000	0	0	5	0.35	0.000%	45.15488	133.83	0.00	3034.9
PUT Users	2000	0	0	1	0.02	100.000%	22.59785	26.31	0.00	1192.0
TOTAL	12000	1471	0	15223	1926.43	16.700%	126.10208	288.87	119.72	2345.7

3.4 Stress Test

To do a stress test for the application, we'll simulate way more users than the usual accessing the users page. The thread properties defined will be:

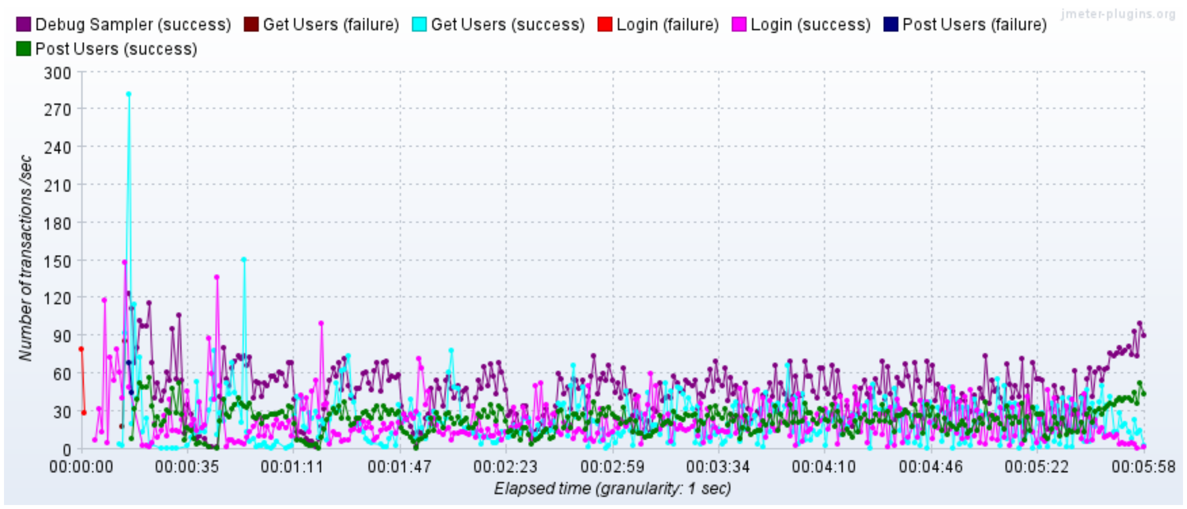
- **Number of Threads (users):** 800
- **Ramp-up period:** 1
- **Loop Count:** 10

Only 1 thread was defined for each test, so it'll only take 1 second to ramp it up. To see the results, a View Results Tree Listener and a Summary Report was used. It was also used a jp@gc - Response Codes per Second and jp@gc - Transactions per Second. These were possible after the download of these plugins.



By this image (jp@gc - Response Codes per Second), it is possible to see that in the same second several requests are being handled. Besides this, we can also see the codes from each request.

When comparing the results with the load tests there is only the difference that in this stress test, the number of transactions increased.



To show the results, we used a Summary Report:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	8000	13138	1	28010	3619.31	1.363 %	22.32722	120.19	4.82	5512.2
Get Users	8000	10428	7	29877	2792.08	1.363 %	22.41129	19.03	60.07	869.7
Post	8000	1107	12	252	2949.	1.413	23.1625	9.05	64.8	399.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Users		2	4	18	21	%	6		6	9
Debug Sampler	16000	0	0	56	1.13	0.000%	46.74563	138.89	0.00	3042.5
TOTAL	40000	6927	0	29877	6222.28	0.827%	111.59469	280.44	127.14	2573.4

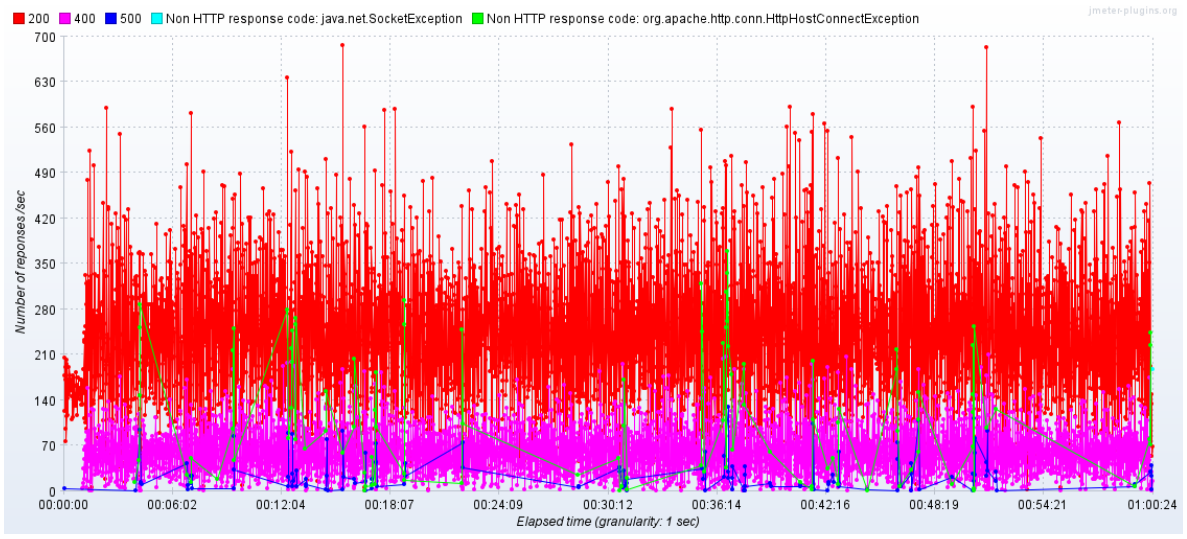
As expected the error, as the number of transactions increases but continues to be a small value, so it is not concerning.

3.5 Soak Test

To do a soak test for the application, we'll simulate 200 users accessing the users page, but for an hour.The thread properties defined will be:

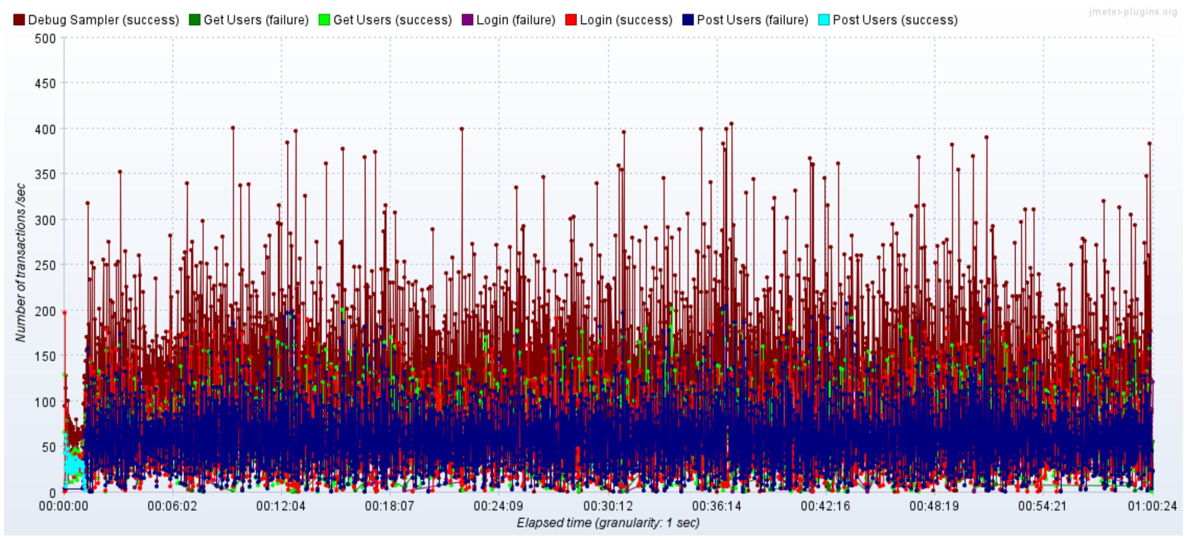
- **Number of Threads (users):** 200
- **Ramp-up period:** 1
- **Loop Count:** infinite

Only 1 thread was defined for each test, so it'll only take 1 second to ramp it up. To see the results, a View Results Tree Listener and a Summary Report was used. It was also used a jp@gc - Response Codes per Second and jp@gc - Transactions per Second. These were possible after the download of these plugins.

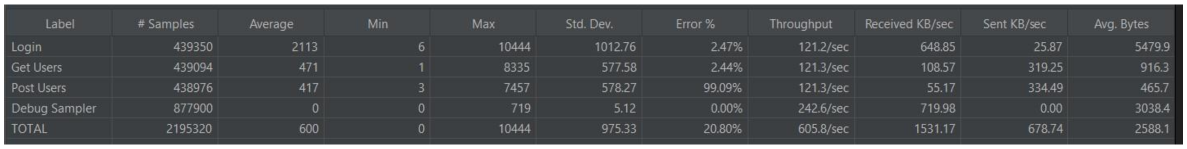


By this image (jp@gc - Response Codes per Second), it is possible to see that in the same second several requests are being handled. Besides this, we can also see the codes from each request.

When comparing the results, it is possible to see that there is an increase in the number of failures. These failures were the result of trying to make a request post to an object that already existed. The requests post developed in this soak test use a csv with several users (200). This fact can explain why there are so many failures and the error of this request is so high.



To show the results, we used a Summary Report:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	439350	2113	6	10444	1012.76	2.47%	121.2/sec	648.85	25.87	5479.9
Get Users	439094	471	1	8335	577.58	2.44%	121.3/sec	108.57	319.25	916.3
Post Users	438976	417	3	7457	578.27	99.09%	121.3/sec	55.17	334.49	465.7
Debug Sampler	877900	0	0	719	5.12	0.00%	242.6/sec	719.98	0.00	3038.4
TOTAL	2195320	600	0	10444	975.33	20.80%	605.8/sec	1531.17	678.74	2588.1

4. Security

To analyse the vulnerabilities found, their severity level and the amount of work needed to fix them, the group used the plugin SpotBugs from IntelliJ. [4]

IntelliJ SpotBugs plugin provides static byte code analysis to look for bugs in Java code from within IntelliJ IDEA. SpotBugs is a defect detection tool for Java that uses static analysis to look for more than 400 bug patterns, such as null pointer dereferences, infinite recursive loops, bad uses of the Java libraries and deadlocks.

4.1 Malicious code vulnerability

4.1.1 Method returning array may expose internal representation

Returning a reference to a mutable object value stored in one of the object’s fields exposes the internal representation of the object. If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, it is necessary to do something different like returning a new copy of the object.

This problem has a medium priority.

Class	Method	Field
DataPagePojo	getItems()	items
User	getUserRole()	userRole
User	getPerson()	person
UserPojo	getPerson()	person

4.1.2 Storing reference to mutable object

Storing a reference to an externally mutable object into the internal representation of the object may expose internal representation. If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, it is necessary to do something different like storing a copy of the object.

This problem has a medium priority.

Class	Method	Field
DataPagePojo	DataPagePojo()	items
DataPagePojo	setItems()	items
User	setUserRole()	userRole
User	User()	person
User	setPerson()	person
User	User()	userRole
User$UserBuilder	userRole()	userRole
User$UserBuilder	person()	person
UserPojo	UserPojo()	person
UserPojo	setPerson ()	person
UserPojo$UserPojoBuilder	person()	person
UsersCrudServiceImpl	UsersCrudServiceImpl()	securityProperties

5. Architecture Compliance

Architecture conformance is the degree to which software complies with its defined architecture, standards, and best practices. Software architecture is a comprehensive framework that defines the organization, components, interfaces, interactions and other important aspects of a software system. Architectural compliance can be an important part of the software quality assurance process and is a common practice in large, critical enterprise software development projects. Architectural compliance is important to ensure that software is developed and maintained in a consistent way, avoiding deviations and problems that can compromise its quality, security, performance, and scalability.

Using ArchUnit with JUnit5 some fitness functions were written to check and evaluate the compliance of the project architecture. [5] [6]

5.1 Controller

5.1.1 Non-Private Methods

```
  @ArchTest
  public static final ArchRule methodsShouldNotBePrivate =
    methods().that()
      .areDeclaredInClassesThat().areAnnotatedWith(RestController.class)
      .and()
      .areNotAnnotatedWith(ExtendWith.class)
      .should().notBePrivate();
```

The purpose of this test is to check that the methods in classes annotated with “@RestController” are not private (do not have the access modifier “private”).

This test failed because the method “getUserDetails” of class “AccessController” and the method “fetchProductListByCode” of class “DataProductListContentsController” violated this rule, as both have the access modifier “private”. This rule was violated 2 times in the code.

5.1.2 Class and package containment

```
  @ArchTest
  public static final ArchRule
controllerClassesShouldResideInControllerPackage =
    classes().that()
      .areAnnotatedWith(RestController.class)
      .and()
      .haveSimpleNameNotContaining("Test")
      .should().resideInAPackage("..controllers..");
```

This test is to verify that the classes annotated with “@RestController” are located within the “controllers” package.

This test failed. The error shows that the “SimpleController” class annotated with “@RestController” in the “JwtVerifiterFilterTest.java” file does not reside in the “…controllers…” package, violating the rule set.

5.1.3 Inheritance

```
  @ArchTest
  public static final ArchRule controllersShouldBeController =
    classes().that()
      .areAnnotatedWith(RestController.class)
      .and()
      .haveSimpleNameNotContaining("Test")
      .should().haveSimpleNameEndingWith("Controller");
```

This test is to verify that the classes annotated with "@RestController" have their name ending with the word "Controller".

This test passed without errors.

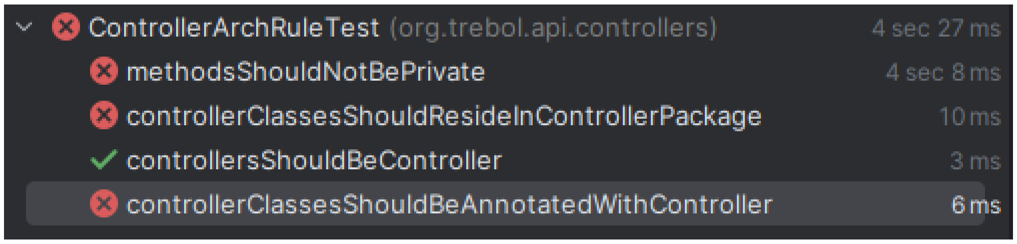
5.1.4 Annotation

```
  @ArchTest
  public static final ArchRule
controllerClassesShouldBeAnnotatedWithController =
    classes().that()
      .resideInAPackage("..controllers..")
      .and()
      .haveSimpleNameNotContaining("Test")
      .should().beAnnotatedWith(RestController.class);
```

This test aims to verify that the classes that reside in the "...controllers..." package have the "@RestController" annotation.

This test failed because the "SimplePrincipal" class needs to be annotated with "@RestController" to fit the architecture defined by the test.

Results



```
ControllerArchRuleTest (org.trebol.api.controllers)        4 sec 27 ms
  methodsShouldNotBePrivate                                 4 sec 8 ms
  controllerClassesShouldResideInControllerPackage               10 ms
  controllersShouldBeController                                   3 ms
  controllerClassesShouldBeAnnotatedWithController                6 ms
```

5.2 Service

5.2.1 Package dependency

```
  @ArchTest
  public static final ArchRule
serviceClassesShouldNotAccessControllerClasses =
    noClasses().that().resideInAnyPackage("..services..")
      .should().dependOnClassesThat().resideInAnyPackage( "controllers");
```

This test defines an architectural rule that ensures that service classes do not directly access controller classes. This rule helps maintain the separation of responsibilities between the service and control layers in a software architecture. This test passed without errors.

5.2.2 Class and package containment

```
  @ArchTest
  public static final ArchRule serviceClassesShouldResideInServicePackage
=
    classes().that()
      .areAnnotatedWith(Service.class)
      .and()
      .areNotInterfaces()
      .should().resideInAPackage("..services..");
```

This test checks that all classes that are annotated with @Service (which are service implementations) are located in the "...services..." package. This helps ensure good code organization by preventing service classes from being spread across different packages, which can make code maintenance difficult.

This test did not pass. The error message indicates that these classes need to be moved to the correct "...services..." package.

Example:

```
Class <org.trebol.mailing.impl.mailgun.MailgunMailingServiceImpl> does
not reside in a package '..services..' in
(MailgunMailingServiceImpl.java:0)
Class <org.trebol.payment.impl.webpayplus.WebpayplusPaymentServiceImpl>
does not reside in a package '..services..' in
(WebpayplusPaymentServiceImpl.java:0)
Class <org.trebol.security.UserDetailsServiceImpl> does not reside in a
package '..services..' in (UserDetailsServiceImpl.java:0)
```

5.2.3 Inheritance

```
  @ArchTest
  public static final ArchRule servicesShouldBeService =
    classes().that()
      .areAnnotatedWith(Service.class)
      .and()
      .haveSimpleNameNotContaining("Test")
      .should().haveSimpleNameEndingWith("ServiceImpl");
```

This test aims to ensure that all service classes in the project follow a consistent naming convention. This test passed without errors.

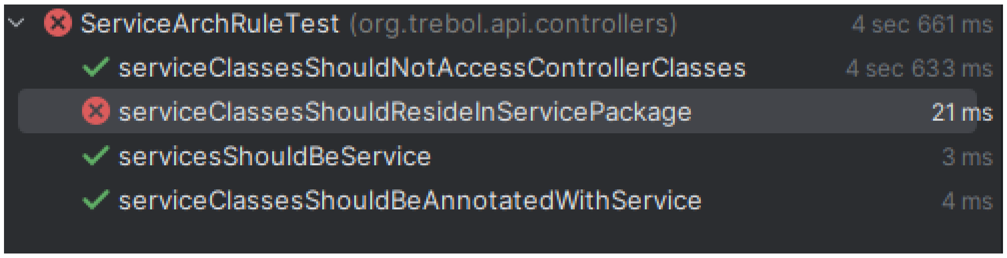
5.2.4 Annotation

```
  @ArchTest
  public static final ArchRule serviceClassesShouldBeAnnotatedWithService
=
    classes().that()
      .resideInAPackage("..services..")
      .and()
      .areNotInterfaces()
      .and()
      .haveSimpleNameNotContaining("Test")
      .and()
      .haveSimpleNameEndingWith("ServiceImpl")
      .should().beAnnotatedWith(Service.class);
```

This test checks whether the service classes in a Java project conform to naming conventions and annotations.

This test passed wihout errors.

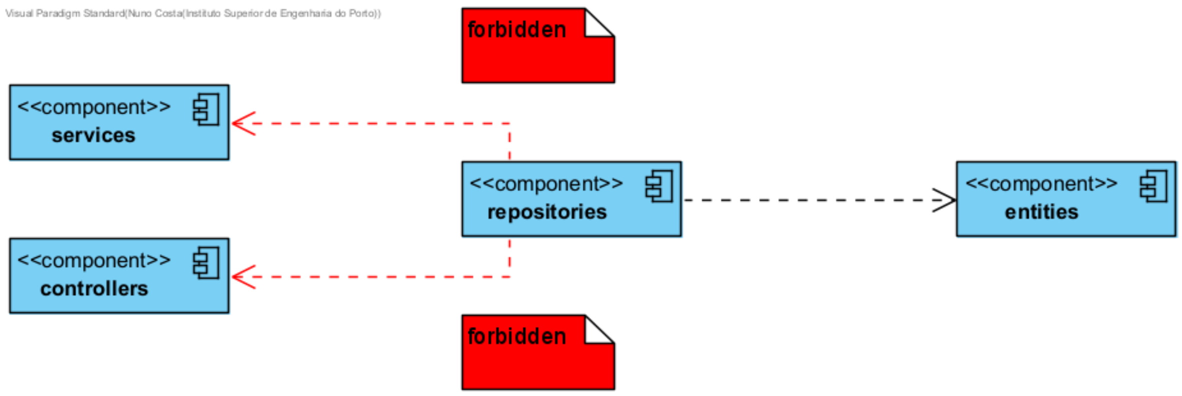
Results



```
✗ ServiceArchRuleTest (org.trebol.api.controllers)            4 sec 661 ms
    ✓ serviceClassesShouldNotAccessControllerClasses          4 sec 633 ms
    ✗ serviceClassesShouldResideInServicePackage                     21 ms
    ✓ servicesShouldBeService                                         3 ms
    ✓ serviceClassesShouldBeAnnotatedWithService                      4 ms
```

5.3 Repository

5.3.1 Package dependency


Visual Paradigm Standard(Nuno Costa(Instituto Superior de Engenharia do Porto))
forbidden
<<component>> services
<<component>> repositories
<<component>> entities
<<component>> controllers
forbidden

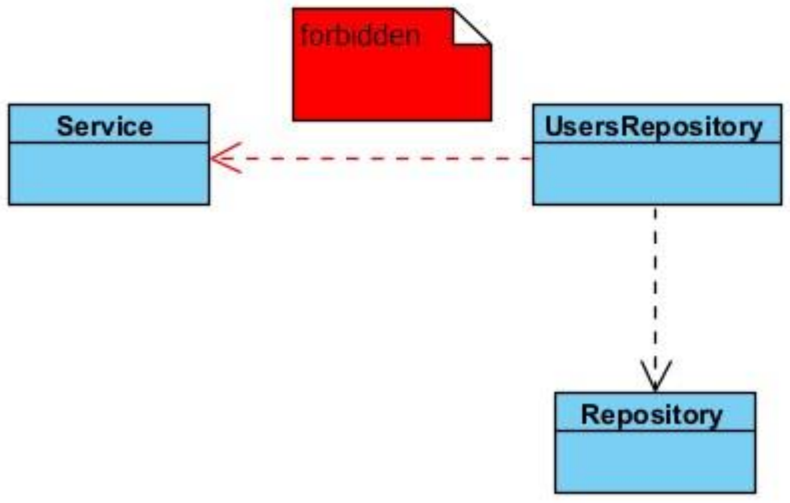
This fitness function checks that classes from the repository package are not depend on classes from the service or controller package. The goal is to ensure that the repository layer is separate from the other layers of the system and that coupling between the layers is minimized.

```
  @ArchTest
  public static final ArchRule
repositoryClassesShouldNotAccessServicesClasses =
    noClasses().that().resideInAnyPackage("..repositories..")
      .should().dependOnClassesThat().resideInAnyPackage( "..services..",
"controllers");
```

This test passed wihout errors.

5.3.2 Class dependency

This fitness function checks that all classes that have a name matching the default ".*Repository" only have dependent classes with the name "Repository". This is done to ensure that these classes do not have unnecessary or unrelated dependencies that could cause unnecessary coupling or confusion in the code.



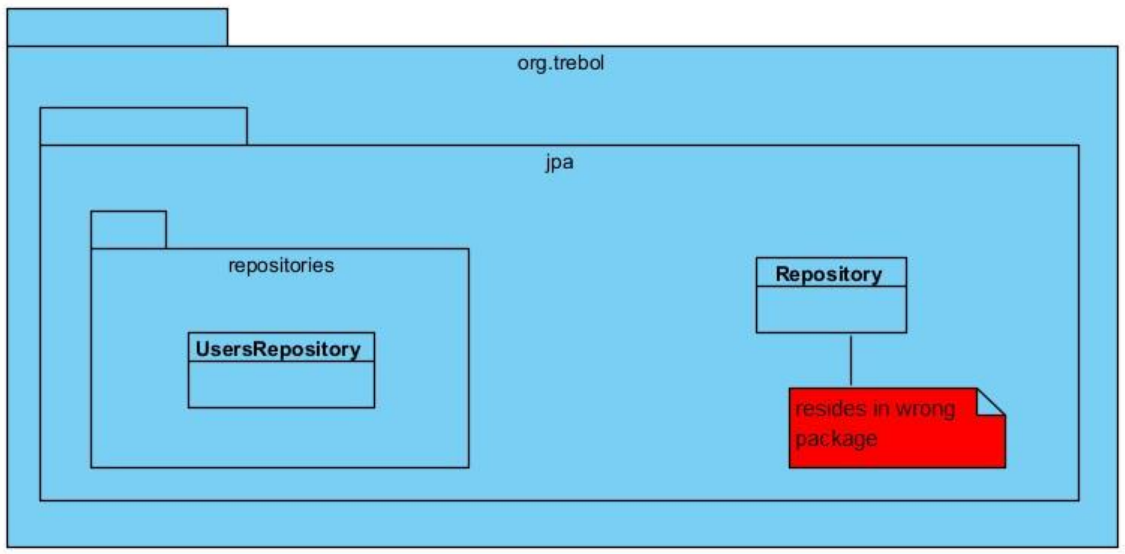
```
  @ArchTest
  public static final ArchRule
repositoryMatchingClassSimpleNameRepository =
    classes().that().haveNameMatching(".*Repository")

.should().onlyHaveDependentClassesThat().haveSimpleName("Repository");
```

This test failed because some classes have dependencies that do not have the required simple Repository name. This may be a design issue. Example:

```
Constructor
<org.trebol.api.services.impl.CompanyServiceImpl.<init>(org.trebol.jpa.re
positories.ParamsRepository)> has parameter of type
<org.trebol.jpa.repositories.ParamsRepository> in
(CompanyServiceImpl.java:0)
```

5.3.3 Class and package containment

The rule defined in this fitness function is that all classes whose names end with “Repository” must be located in the “..repositories” package.

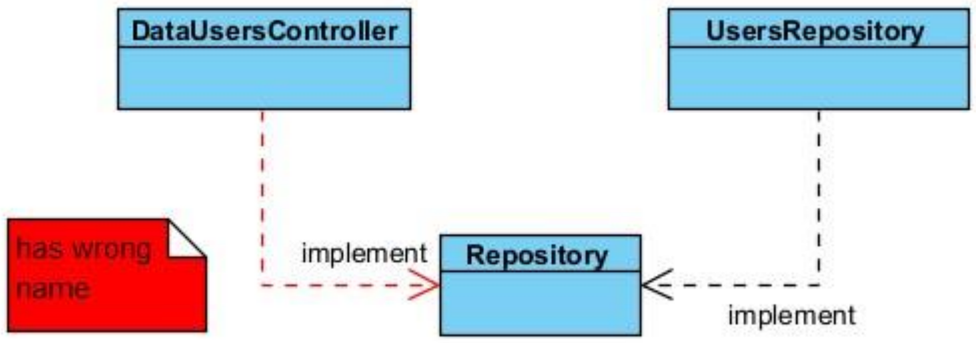


```
  @ArchTest
  public static final ArchRule
repositoryEndingNameShouldResideInRepository =
    classes().that().haveSimpleNameEndingWith("Repository")
      .should().resideInAPackage("..repositories");
```

This test has not passed because of a class that isn't in the expected package. According to the error message, the class is "org.trebol.jpa.Repository", which does not reside in the expected package.

5.3.4 Inheritance

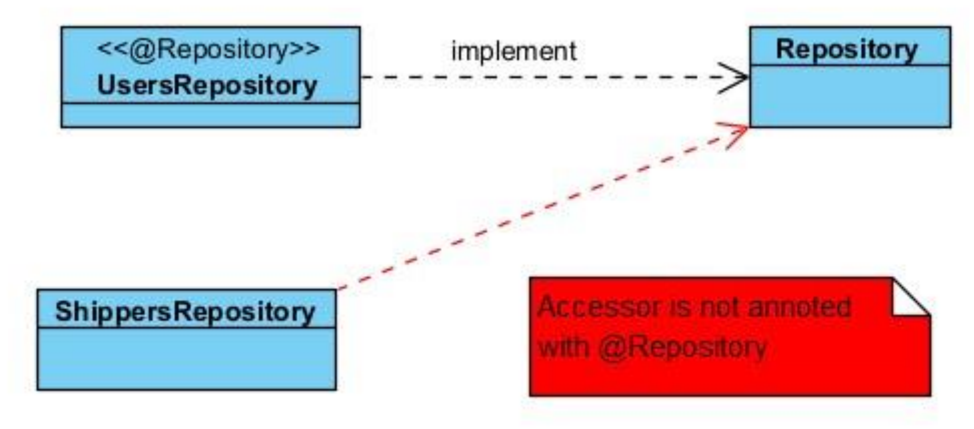
This fitness function checks that all classes that implement the JpaRepository interface have names that end with "Repository".



```
  @ArchTest
  public static final ArchRule
repositoryClassesShouldImplementJpaRepository =
    classes().that().implement(Repository.class)
      .should().haveSimpleNameEndingWith("Repository");
```

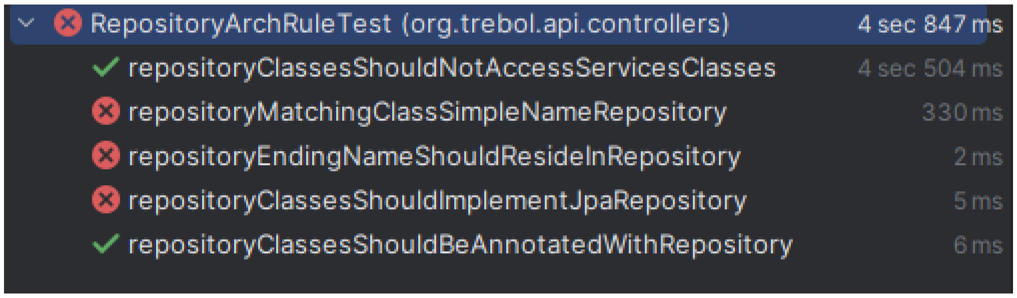
5.3.5 Annotation

This fitness function checks that all classes that are assignable to Repository are annotated with @Repository.



```
  @ArchTest
  public static final ArchRule
repositoryClassesShouldBeAnnotatedWithRepository =
    classes().that()
      .areAssignableTo(Repository.class)
      .should().beAnnotatedWith(Repository.class);
```

Results

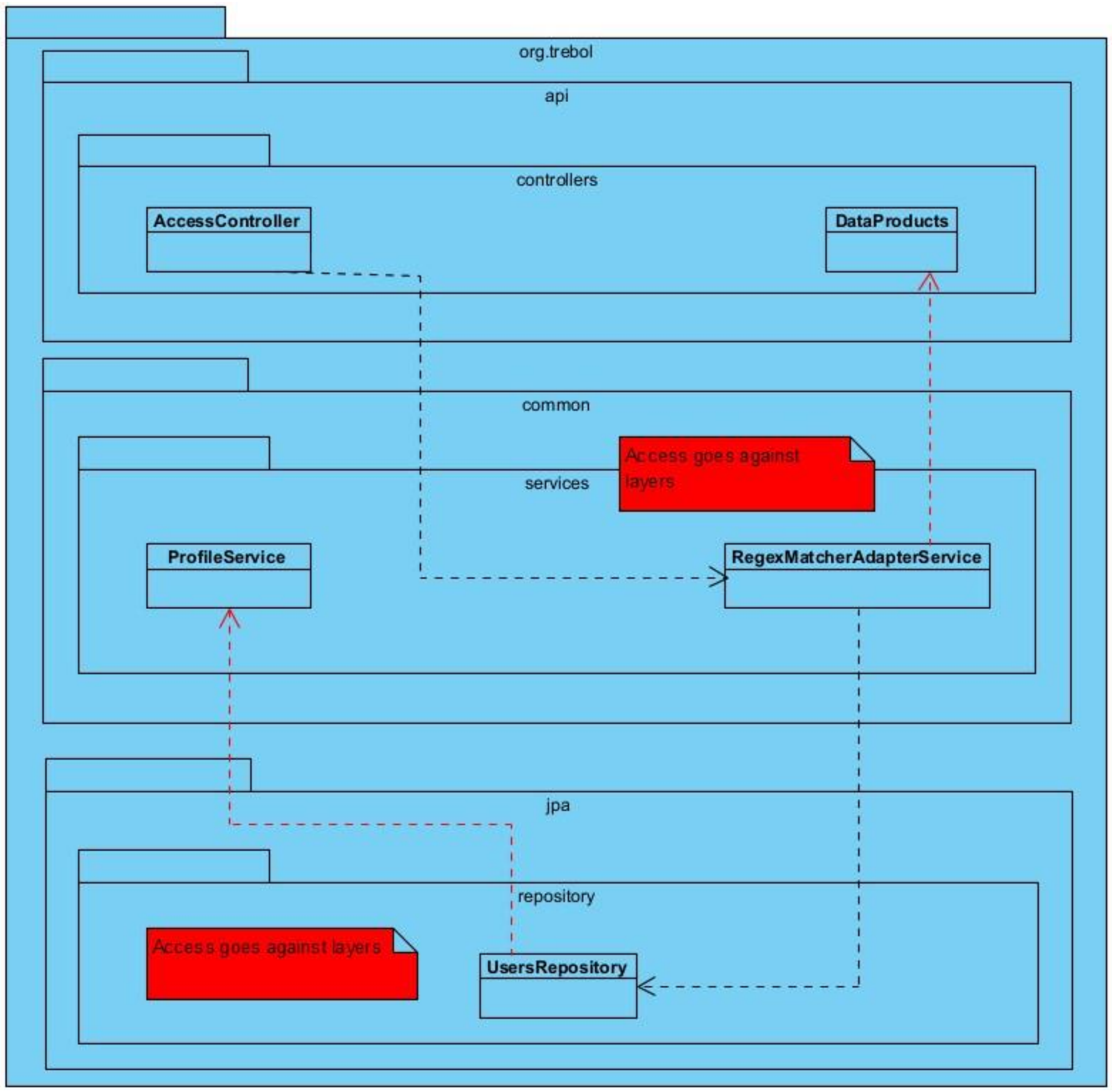


```
⌄ ✗ RepositoryArchRuleTest (org.trebol.api.controllers)      4 sec 847 ms
    ✓ repositoryClassesShouldNotAccessServicesClasses        4 sec 504 ms
    ✗ repositoryMatchingClassSimpleNameRepository                 330 ms
    ✗ repositoryEndingNameShouldResideInRepository                  2 ms
    ✗ repositoryClassesShouldImplementJpaRepository                 5 ms
    ✓ repositoryClassesShouldBeAnnotatedWithRepository              6 ms
```

5.4 Application

5.4.1 Layer

The purpose of this test is to verify the layered architecture pattern. The control layer cannot be accessed by any layer, the service layer can only be accessed by the control layer, and the repository layer can only be accessed by the service layer.



```
@ArchTest
public static final ArchRule layeredArchitecture =
  layeredArchitecture().consideringOnlyDependenciesInLayers()
      .layer("Controller").definedBy("..controllers..")
      .layer("Service").definedBy("..services..")
      .layer("Repository").definedBy("..repositories..")
      .whereLayer("Controller").mayNotBeAccessedByAnyLayer()
      .whereLayer("Service").mayOnlyBeAccessedByLayers("Controller")
      .whereLayer("Repository").mayOnlyBeAccessedByLayers("Service");
```

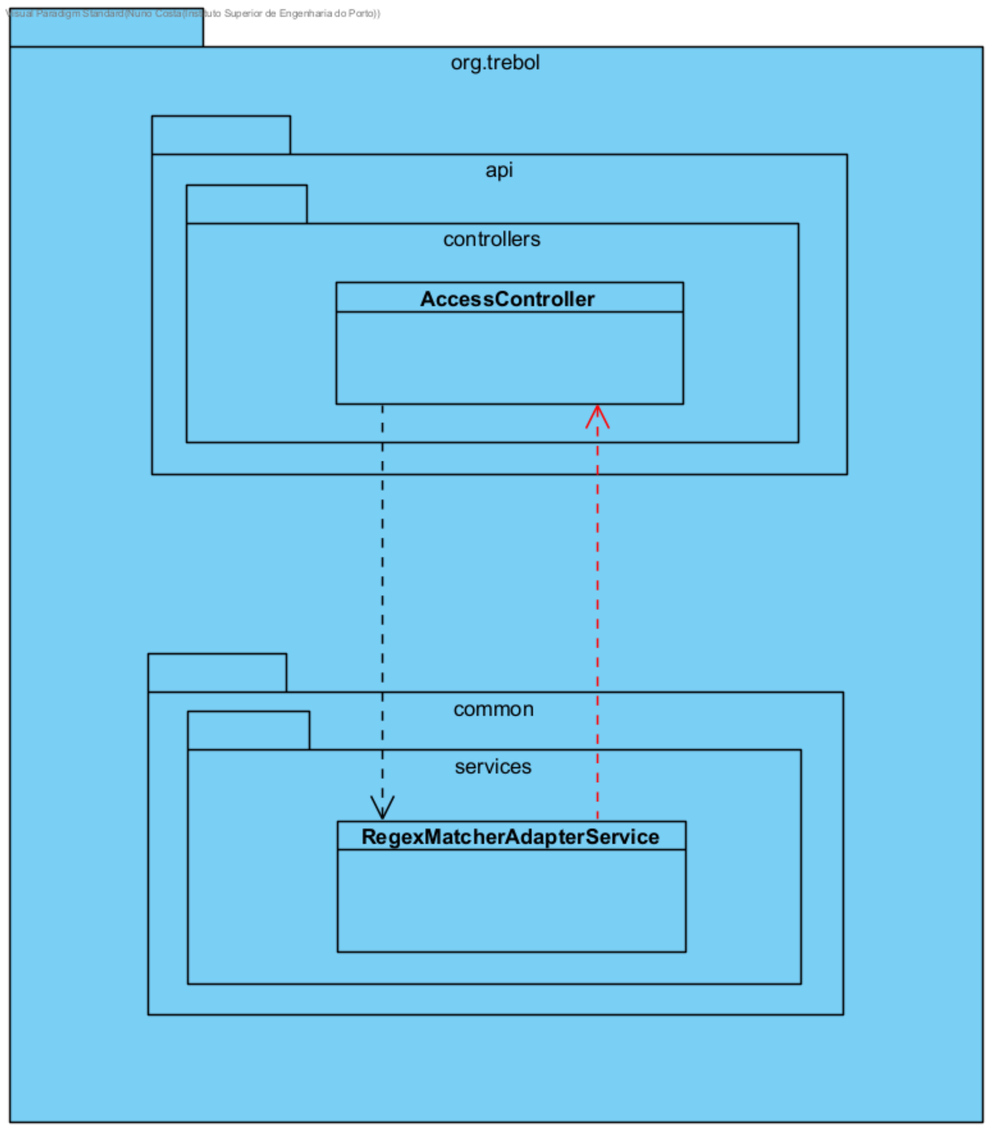
This test fails due to the use of ProductListItemsRepository and ProductListsRepository by the DataProductListContentsController class, which belongs to the control layer. This violation occurred 32 times in the system.

Example:

```
Field
<org.trebol.api.controllers.DataProductListContentsController.listItemsRe
pository> has type
<org.trebol.jpa.repositories.ProductListItemsRepository> in
(DataProductListContentsController.java:0)
Field
<org.trebol.api.controllers.DataProductListContentsController.listsReposi
tory> has type <org.trebol.jpa.repositories.ProductListsRepository> in
(DataProductListContentsController.java:0)
Field
<org.trebol.api.controllers.DataProductListContentsControllerTest.listIte
msRepositoryMock> has type
<org.trebol.jpa.repositories.ProductListItemsRepository> in
(DataProductListContentsControllerTest.java:0)
Field
<org.trebol.api.controllers.DataProductListContentsControllerTest.listsRe
positoryMock> has type
<org.trebol.jpa.repositories.ProductListsRepository> in
(DataProductListContentsControllerTest.java:0)
```

5.4.2 Cycle

This fitness function checks that there are no cycles in the dependencies between packages that match the pattern ..org.trebol.(*)... This checks that there are no circular dependencies between different packages.


Visual Paradigm Standard(Nuno Costa(Instituto Superior de Engenharia do Porto))
org.trebol
api
controllers
AccessController
common
services
RegexMatcherAdapterService

```
  @ArchTest
  public static final ArchRule noCyclesInPackageDependencies =
    slices()
       .matching("..org.trebol.(*)..").should().beFreeOfCycles();
```

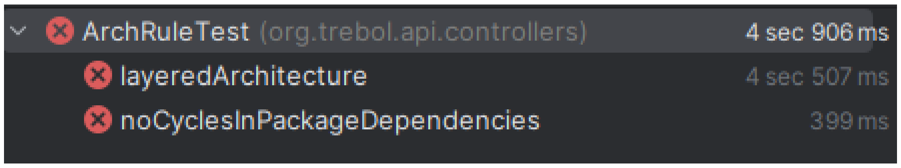
This test failed because there is a specific dependency loop between the slices org.trebol.api and org.trebol.common. This rule was violated 92 times.

Example:

```
Constructor
<org.trebol.api.controllers.AccessController.<init>(org.trebol.security.s
ervices.AuthorizationHeaderParserService,
org.springframework.security.core.userdetails.UserDetailsService,
org.trebol.security.services.AuthorizedApiService,
org.trebol.common.services.RegexMatcherAdapterService)> has parameter of
type <org.trebol.common.services.RegexMatcherAdapterService> in
```

```
(AccessController.java:0)
Field <org.trebol.api.controllers.AccessController.regexMatcherService>
has type <org.trebol.common.services.RegexMatcherAdapterService> in
(AccessController.java:0)
```

Results

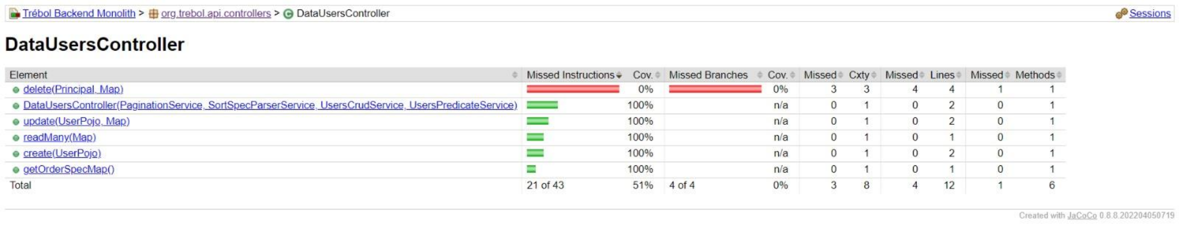


```
ArchRuleTest (org.trebol.api.controllers)      4 sec 906 ms
    layeredArchitecture                        4 sec 507 ms
    noCyclesInPackageDependencies                   399 ms
```

6. Test Examination

6.1 Test completeness

To measure metrics related to test examination, it was used the Jacoco library and the following report was generated with respect to the DataUsersController class. JaCoCo is a free code coverage library for Java, which has been created by the EclEmma team based on the lessons learned from using and integration existing libraries for many years.



Trébol Backend Monolith > org.trebol.api.controllers > DataUsersController   Sessions

DataUsersController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
delete(Principal, Map)	🟥	0%	🟥	0%	3	3	4	4	1	1
DataUsersController(PaginationService, SortSpecParserService, UsersCrudService, UsersPredicateService)	🟩	100%		n/a	0	1	0	2	0	1
update(UserPojo, Map)	🟩	100%		n/a	0	1	0	2	0	1
readMany(Map)	🟩	100%		n/a	0	1	0	1	0	1
create(UserPojo)	🟩	100%		n/a	0	1	0	2	0	1
getOrderSpecMap()	🟩	100%		n/a	0	1	0	1	0	1
Total	21 of 43	51%	4 of 4	0%	3	8	4	12	1	6

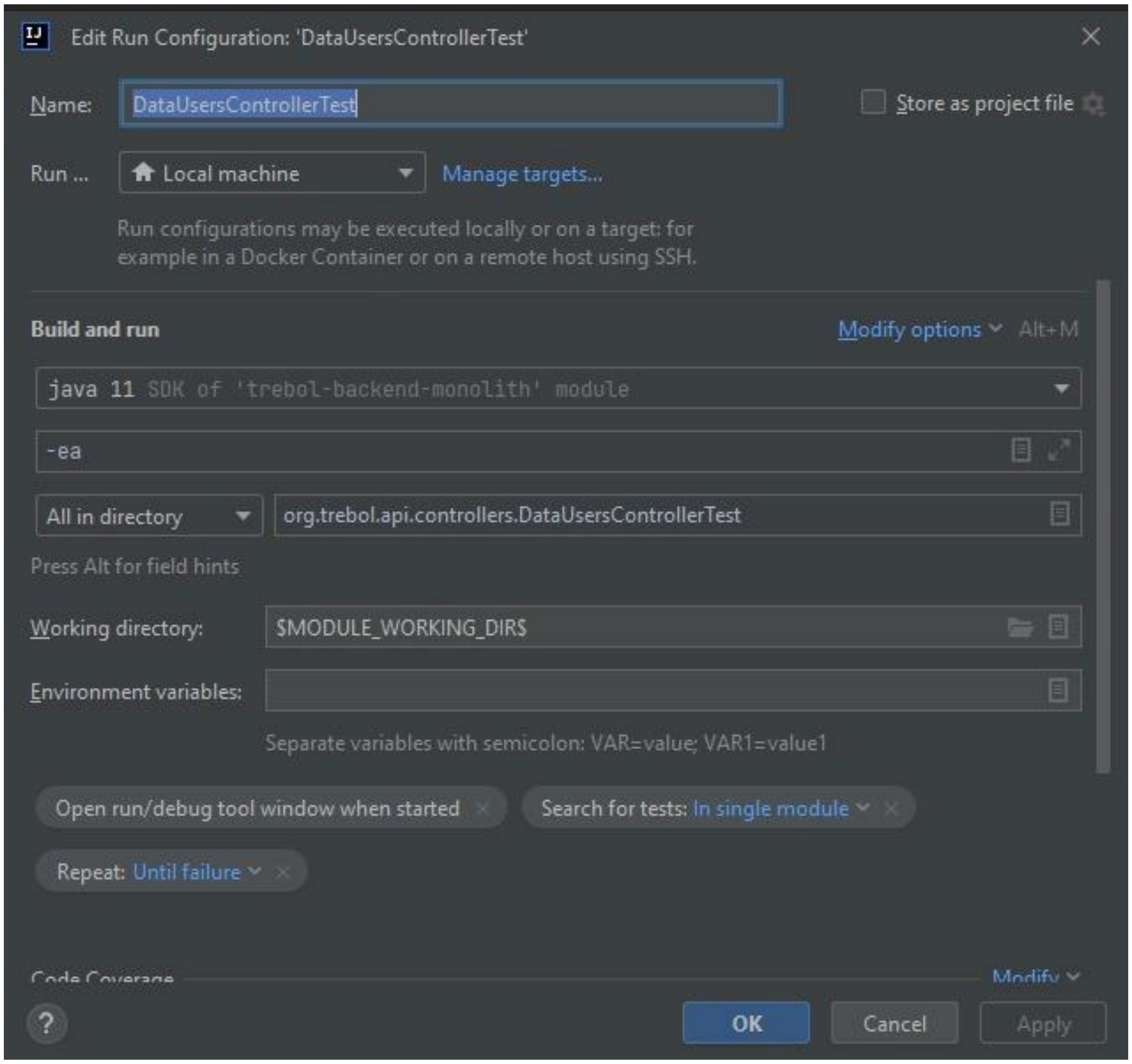
Created with JaCoCo 0.8.8.202204050719

In the image it is possible to see that the tests developed don't have a code coverage of 100%. This is the result of missed instructions and branches regarding the method delete.

6.2 Flaky Tests

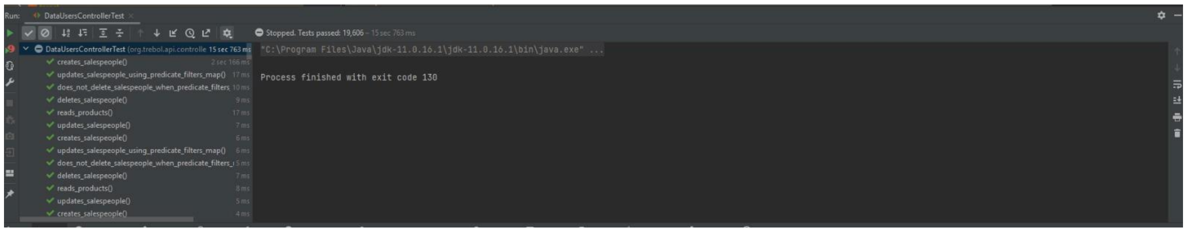
A flaky test is a type of software test that provides inconsistent results with each execution, causing the test to fail or pass unexpectedly even if the code or test remains unchanged. The unpredictable behavior of flaky tests can pose significant challenges for developers trying to diagnose and fix issues, and can potentially impact the end-user experience. [7]

To infer about the existence of tests in the class DataUsersControllerTest, its run configuration was changed so that it stops when any of its tests fail.



```
Edit Run Configuration: 'DataUsersControllerTest'

Name: DataUsersControllerTest                    Store as project file
Run ... Local machine    Manage targets...
Run configurations may be executed locally or on a target: for
example in a Docker Container or on a remote host using SSH.

Build and run                                    Modify options  Alt+M
java 11 SDK of 'trebol-backend-monolith' module
-ea
All in directory    org.trebol.api.controllers.DataUsersControllerTest
Press Alt for field hints
Working directory:       $MODULE_WORKING_DIR$
Environment variables:
                         Separate variables with semicolon: VAR=value; VAR1=value1
Open run/debug tool window when started    Search for tests: In single module
Repeat: Until failure

Code Coverage                                    Modify
?                                        OK    Cancel    Apply
```

The tests ran for 15 seconds and all of them passed, making a total of 19606 successfully completed tests in this time period.



```
Run: DataUsersControllerTest
Stopped. Tests passed: 19,606 - 15 sec 763 ms
DataUsersControllerTest (org.trebol.api.controlle 15 sec 763 ms
  creates_salespeople()                                  2 sec 166 ms
  updates_salespeople_using_predicate_filters_map()      17 ms
  does_not_delete_salespeople_when_predicate_filters     10 ms
  deletes_salespeople()                                  9 ms
  reads_products()                                       17 ms
  updates_salespeople()                                  7 ms
  creates_salespeople()                                  6 ms
  updates_salespeople_using_predicate_filters_map()      6 ms
  does_not_delete_salespeople_when_predicate_filters_i   5 ms
  deletes_salespeople()                                  7 ms
  reads_products()                                       8 ms
  updates_salespeople()                                  5 ms
  creates_salespeople()                                  4 ms

"C:\Program Files\Java\jdk-11.0.16.1\jdk-11.0.16.1\bin\java.exe" ...

Process finished with exit code 130
```

This way, the class DataUsersControllerTest does not have flaky tests.

7. Conclusions

In what concerns the maintainability, considering that the number of components in the system is less than 500, it can be concluded that the propagation cost values obtained are of no concern given the size of the system. DataUsersController seems to be a easy component to understand, it is not complex and a modification will not risk any danger to the system.

Three different types of test were used to evaluate the performance of this class when it receives HTTP requests. Some problems were identified regarding a request.

In the security analysis were found some bugs and by examining the tests of this class it was possible to conclude that it has a code coverage of 51% and there is any flaky tests.

8. References

[1] MetricsTree plugin for IntelliJ IDEA. https://plugins.jetbrains.com/plugin/13959-metricstree

[2] Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994, August). Using metrics to evaluate software system maintainability. Computer, 27(8), 44-49. DOI: 10.1109/2.291294

[3] Apache Software Foundation. https://jmeter.apache.org/

[4] SpotBugs - IntelliJ IDEs Plugin. https://plugins.jetbrains.com/plugin/14014-spotbugs

[5] Gafert, P. (2022). ArchUnit: Automated architecture testing. https://www.archunit.org/

[6] Yin, K. (2022) ArchUnit: Test Software Architecture Easily. https://betterprogramming.pub/archunit-test-software-architecture-easily-a874c54bda1e

[7] Flaky Tests https://httpain.com/blog/debugging-flaky-tests-in-intellij-idea/