CSCI 5308

Advance Topics in Software Development

ASSIGNMENT – 1

Banner ID: B00948977

Git Assignment Link : https://git.cs.dal.ca/sukumaran/fork-from-github/-/tree/master?ref_type=heads (Master branch)

Commit Link: https://git.cs.dal.ca/sukumaran/fork-from-github/-/commit/f3ac4bdc5a12c66f5c1df5af371bb41948465aec

Table of contents

Task #1: Fork a Git repo with following constraints:

I have chosen an open-source project called Cosmo for the given assignment. Link: https://github.com/1and1/cosmo/commits/master

1. It must be a maven or gradle-based project: Cosmo is a Maven project

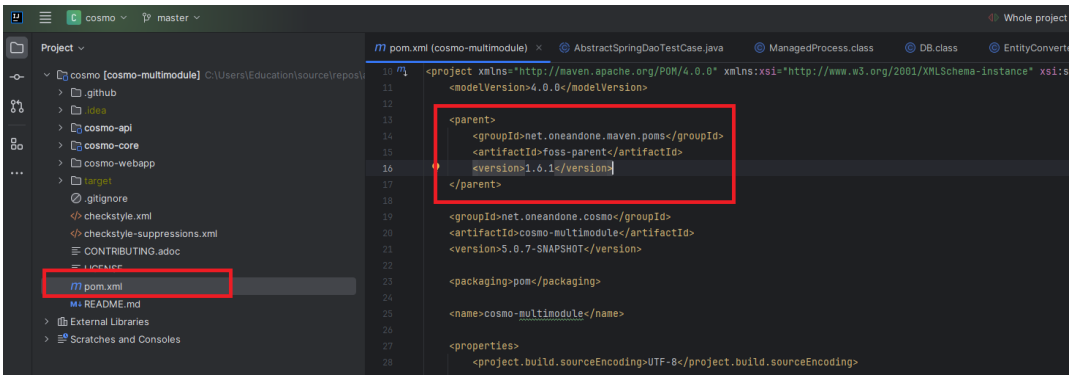


Figure 1: pom.xml screenshot

2. It must have at least 10,000 lines of code: Cosmo has 103,523 lines of code

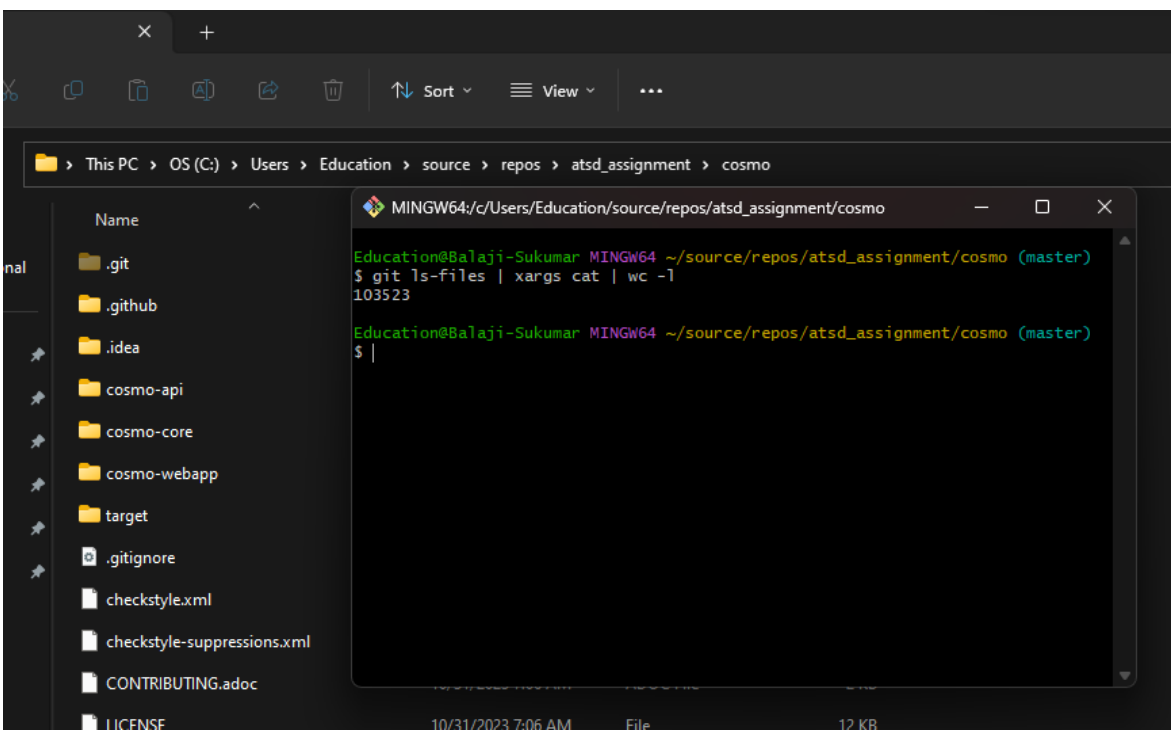


Figure 2: LOC in the repo

3. It must have at least 50 stars: The chosen repo has 75 Stars

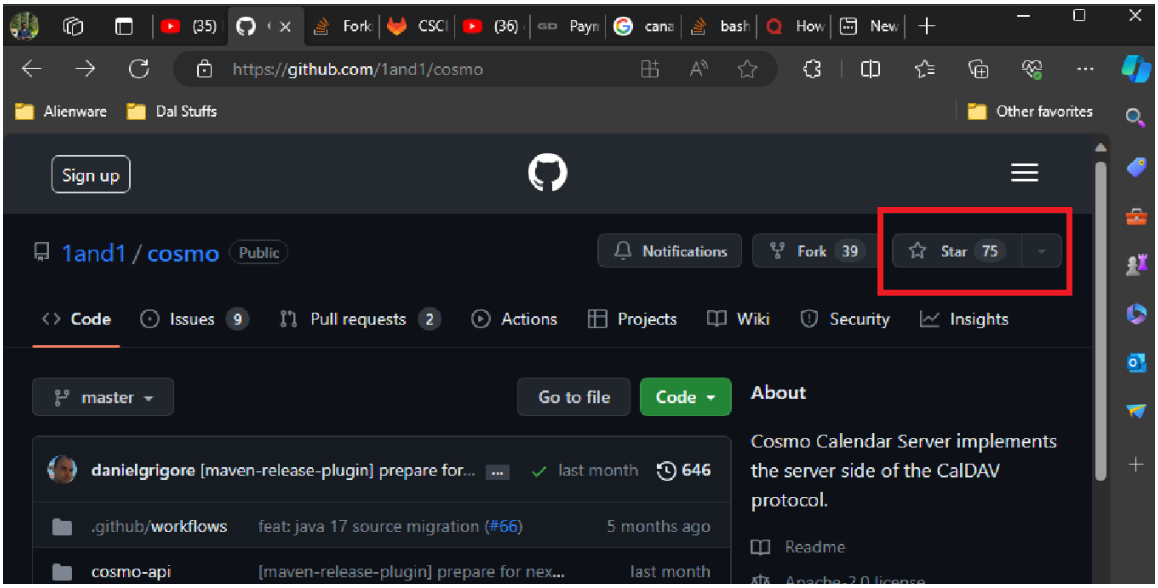


Figure 3: number of stars for the repo

4. It must have tests written using the JUnit framework: Cosmo has tests written using Junit library.

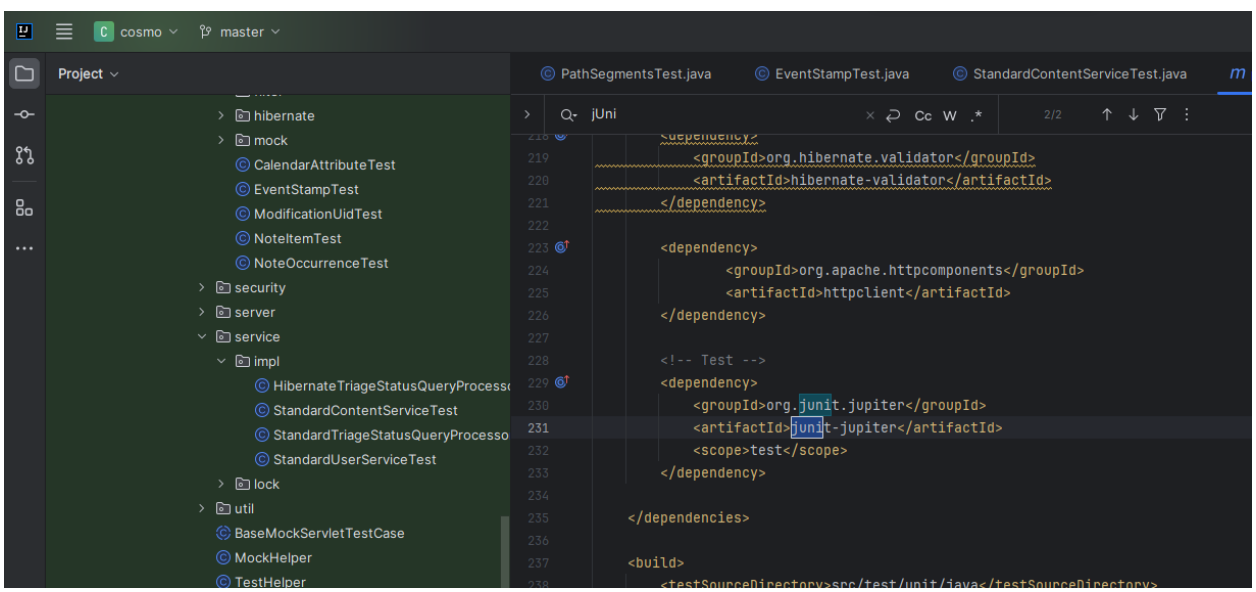


Figure 4: JUnit framework pom.xml

5. It must not be a tutorial or example repository:

   Cosmo application is not a tutorial or example repository. It's an open-source project.

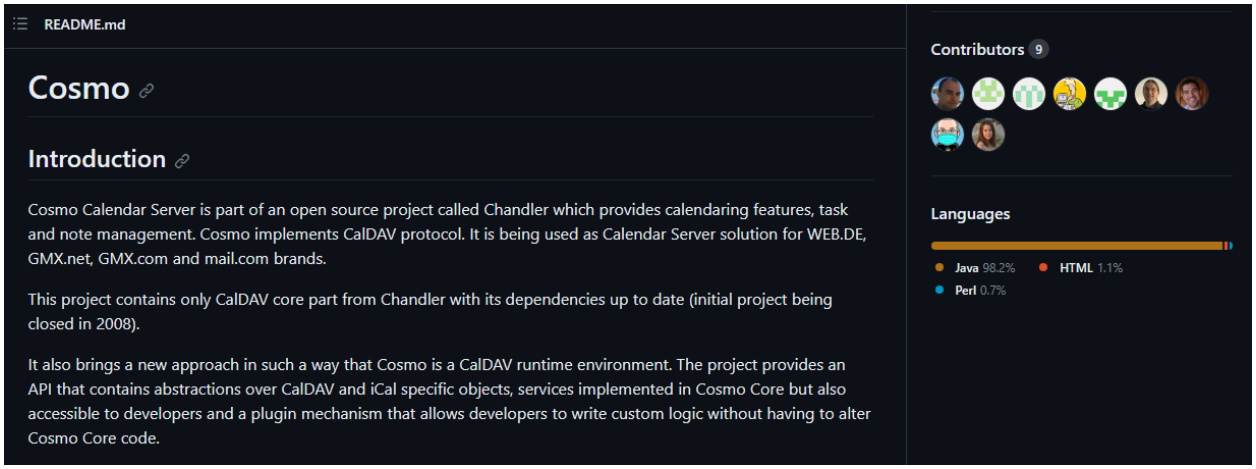


Figure 5: Repo README.md

6. It must be active (at least one commit in the past one year):

   The chosen repository is active and has commits made this year.

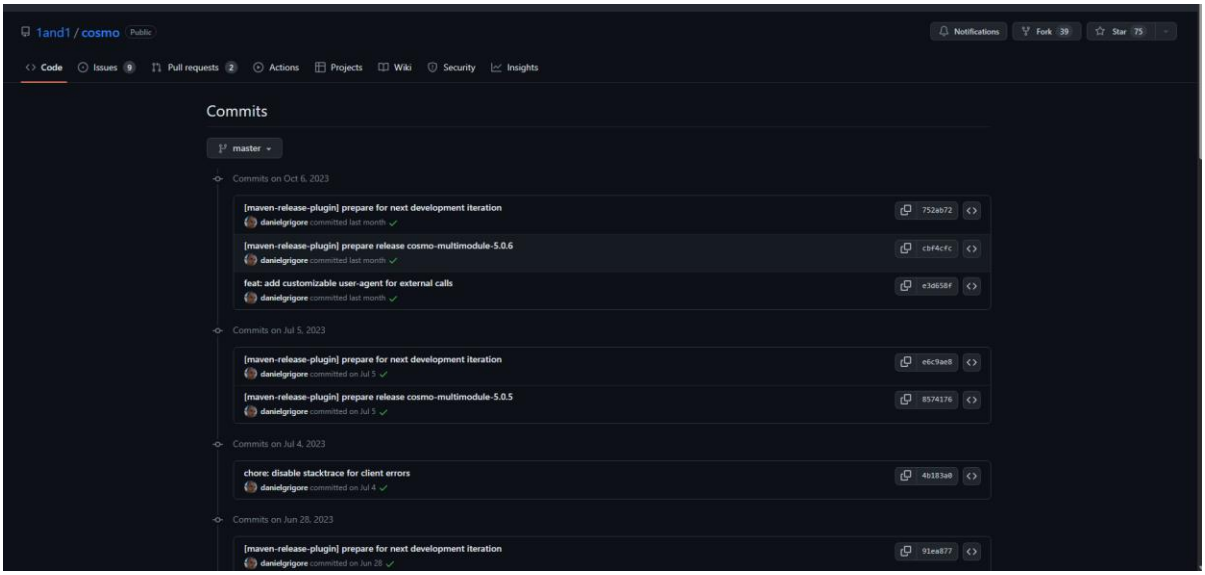


Figure 6: repo activity

<u>Task #2:</u> Provide quantitative measures of test implementation. Specifically, provide the total number of automated tests and code coverage (branch).

Application's java codebase has a total code coverage of 44% (line coverage).

Class coverage: 69/100

Method coverage: 47/100

Line coverage: 44/100

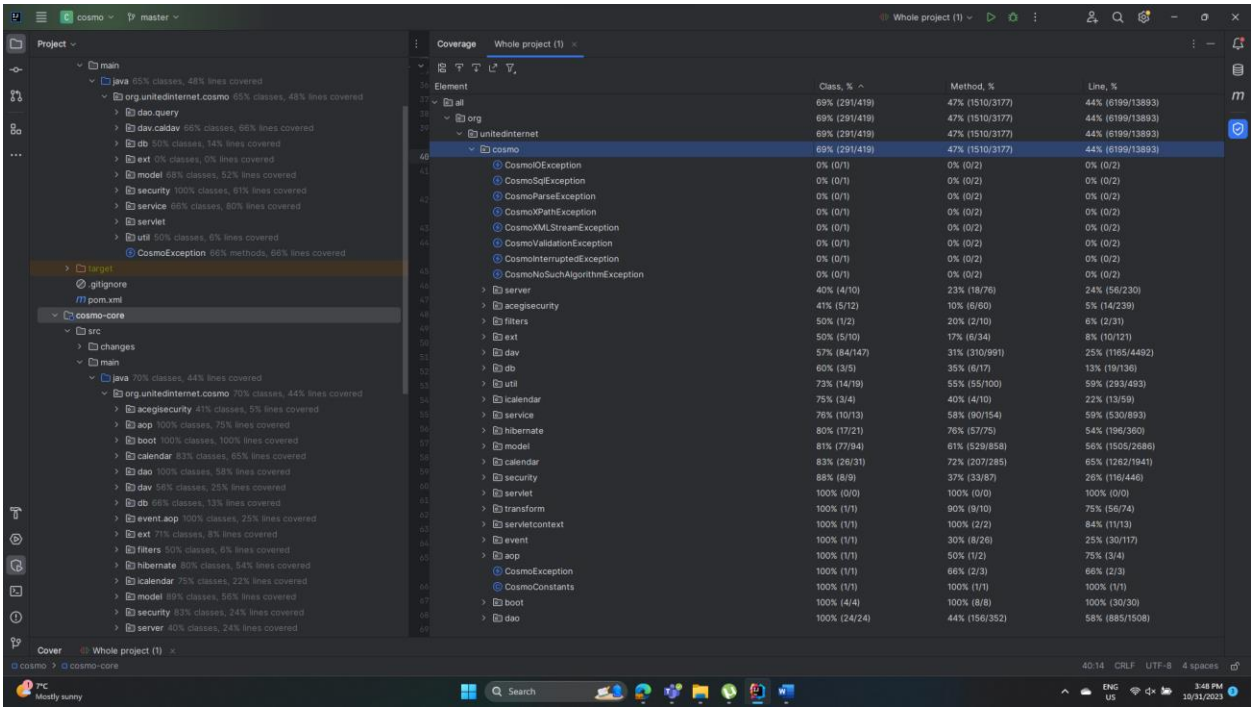


Figure 7: Code coverage

Class coverage:

By sorting based on class coverage,

we can see that, packages such as dao, boot, transform, serverlet, etc has 100% coverage.

Which means that at least a test class for all the class within these packages are defined.

Coverage | Whole project (1)

Element	Class, %	Method, %	Line, %
all	69% (291/419)	47% (1510/3177)	44% (6199/13893)
org	69% (291/419)	47% (1510/3177)	44% (6199/13893)
unitedinternet	69% (291/419)	47% (1510/3177)	44% (6199/13893)
cosmo	69% (291/419)	47% (1510/3177)	44% (6199/13893)
dao	100% (24/24)	44% (156/352)	58% (885/1508)
boot	100% (4/4)	100% (8/8)	100% (30/30)
transform	100% (1/1)	90% (9/10)	75% (56/74)
servletcontext	100% (1/1)	100% (2/2)	84% (11/13)
event	100% (1/1)	30% (8/26)	25% (30/117)
aop	100% (1/1)	50% (1/2)	75% (3/4)
CosmoException	100% (1/1)	66% (2/3)	66% (2/3)
CosmoConstants	100% (1/1)	100% (1/1)	100% (1/1)
servlet	100% (0/0)	100% (0/0)	100% (0/0)
security	88% (8/9)	37% (33/87)	26% (116/446)
calendar	83% (26/31)	72% (207/285)	65% (1262/1941)
model	81% (77/94)	61% (529/858)	56% (1505/2686)
hibernate	80% (17/21)	76% (57/75)	54% (196/360)
service	76% (10/13)	58% (90/154)	59% (530/893)
icalendar	75% (3/4)	40% (4/10)	22% (13/59)
util	73% (14/19)	55% (55/100)	59% (293/493)
db	60% (3/5)	35% (6/17)	13% (19/136)
dav	57% (84/147)	31% (310/991)	25% (1165/4492)
ext	50% (5/10)	17% (6/34)	8% (10/121)
filters	50% (1/2)	20% (2/10)	6% (2/31)
acegisecurity	41% (5/12)	10% (6/60)	5% (14/239)
server	40% (4/10)	23% (18/76)	24% (56/230)
CosmoIOException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoSqlException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoParseException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoXPathException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoXMLStreamException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoValidationException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoInterruptedException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoNoSuchAlgorithmException	0% (0/1)	0% (0/2)	0% (0/2)

Figure 8: class code coverage

For example, if we expand dao, all the subpackages within the dao has a test-class defined.

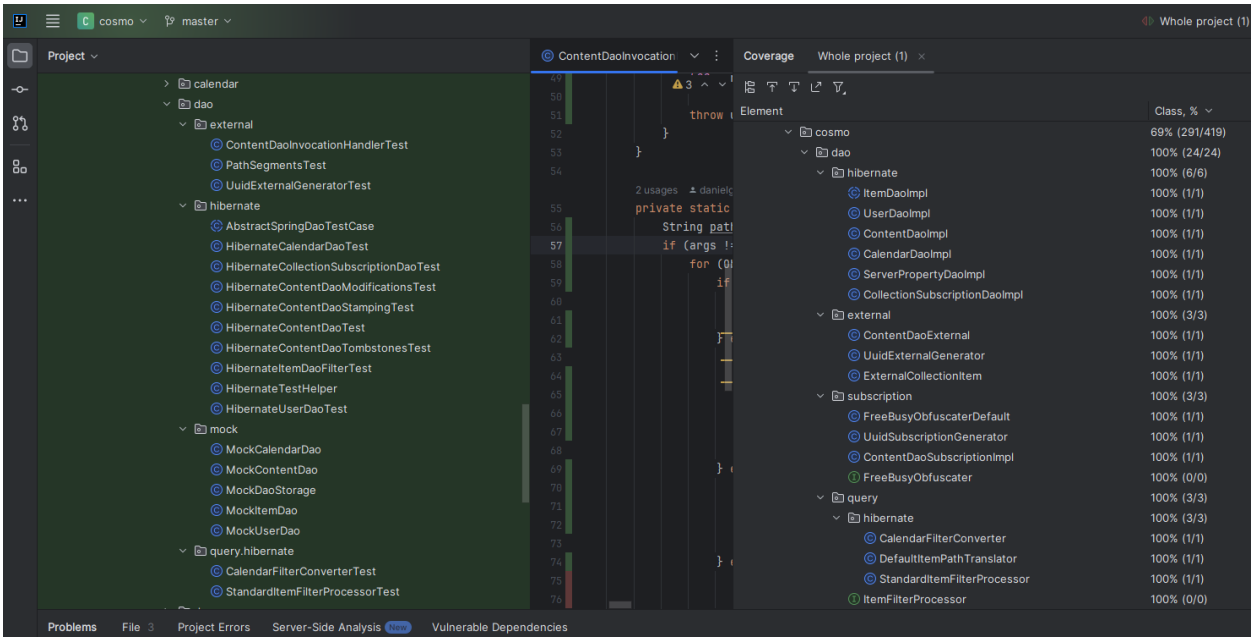


Figure 9: class code coverage details

Method coverage:

Few methods in a class have tests defined and few doesn't. For example: within UserDaoImpl has 100% class coverage but only 87% method coverage. Few methods within the class doesn't have a test defined.

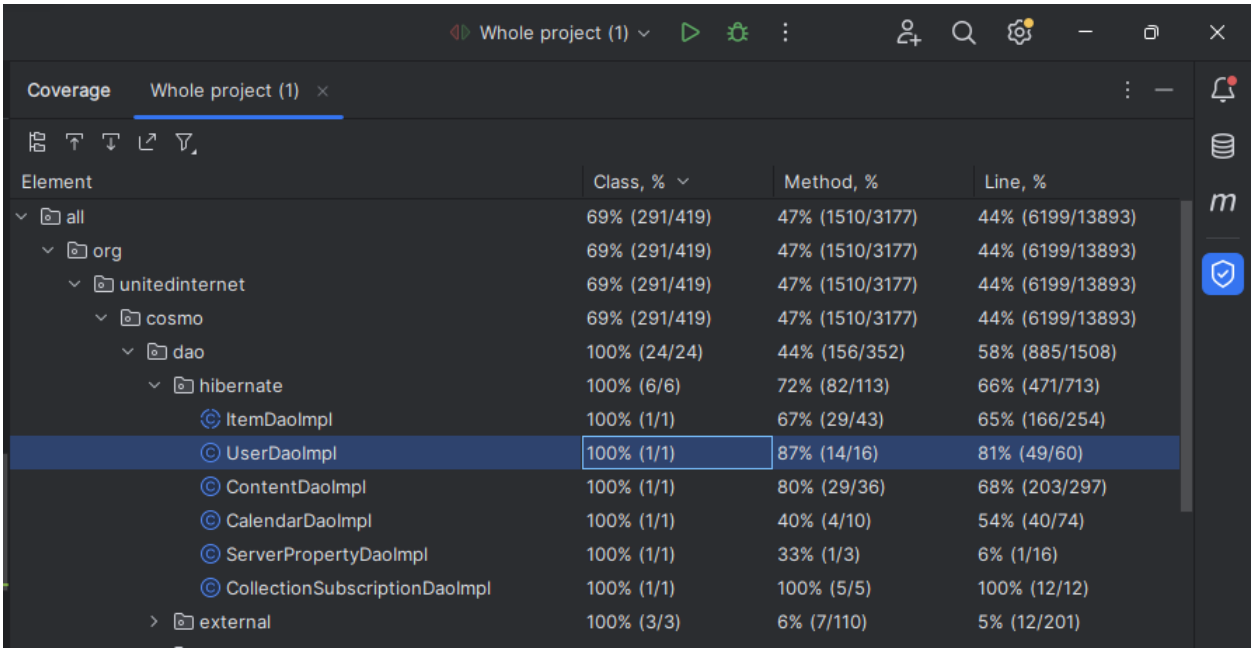


Figure 10: method code coverage

Here, getUserByEmail method within the UserDaoImpl class doesn't have a test defined.

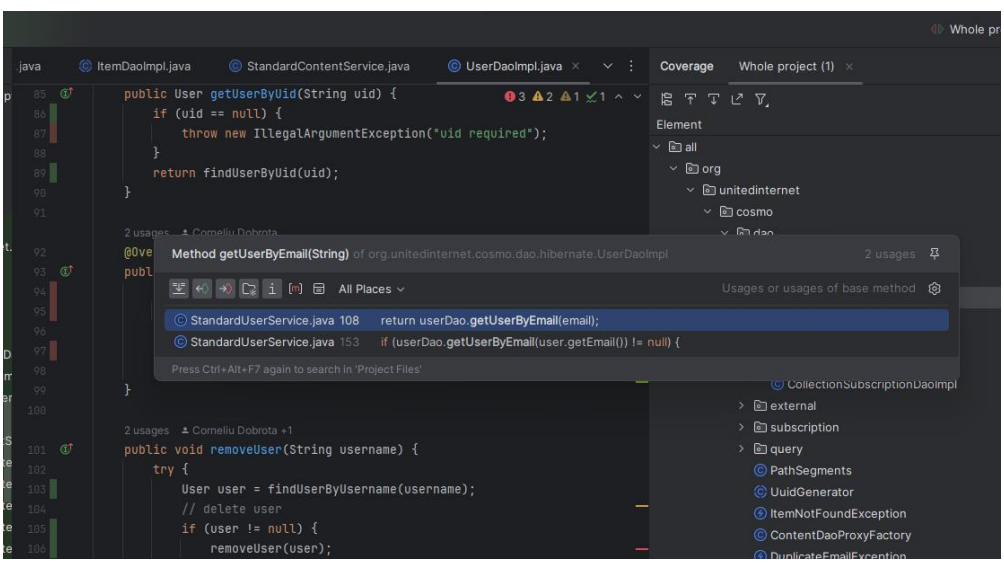


Figure 11: method code coverage detail

Line coverage:

Here although the test was written for method removeUser. Line number 108 till 110 is not covered in test.

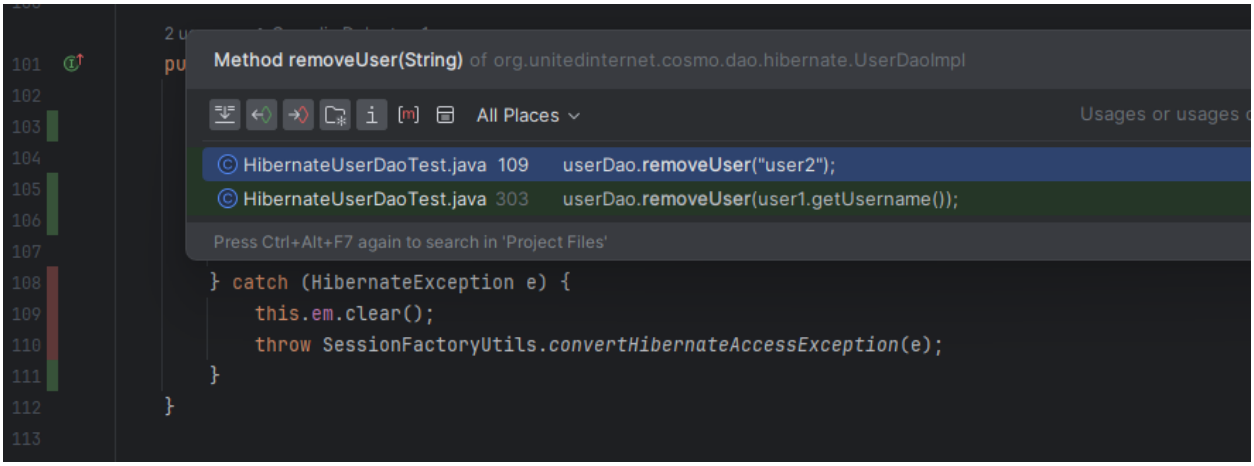


Figure 12: Line code coverage

Task #3:

Critique the test implementation. Provide at least three strong and weak aspects of the test implementation. If you do not find any strong (or weak) aspect, you may have six weak (or strong) aspects in your answer.

Weak aspect 1: Junit test not defined for all java packages within the project

The Project Cosmo application has API, core, and web application packages written in Java. However, the JUnit tests were written only for the API and core, with no unit test cases defined for the web application.

When I attempt to run the JUnit tests just for the Cosmo web application or the entire Cosmo application, it either does not display the run option or indicates that the JUnit library is not defined, respectively.

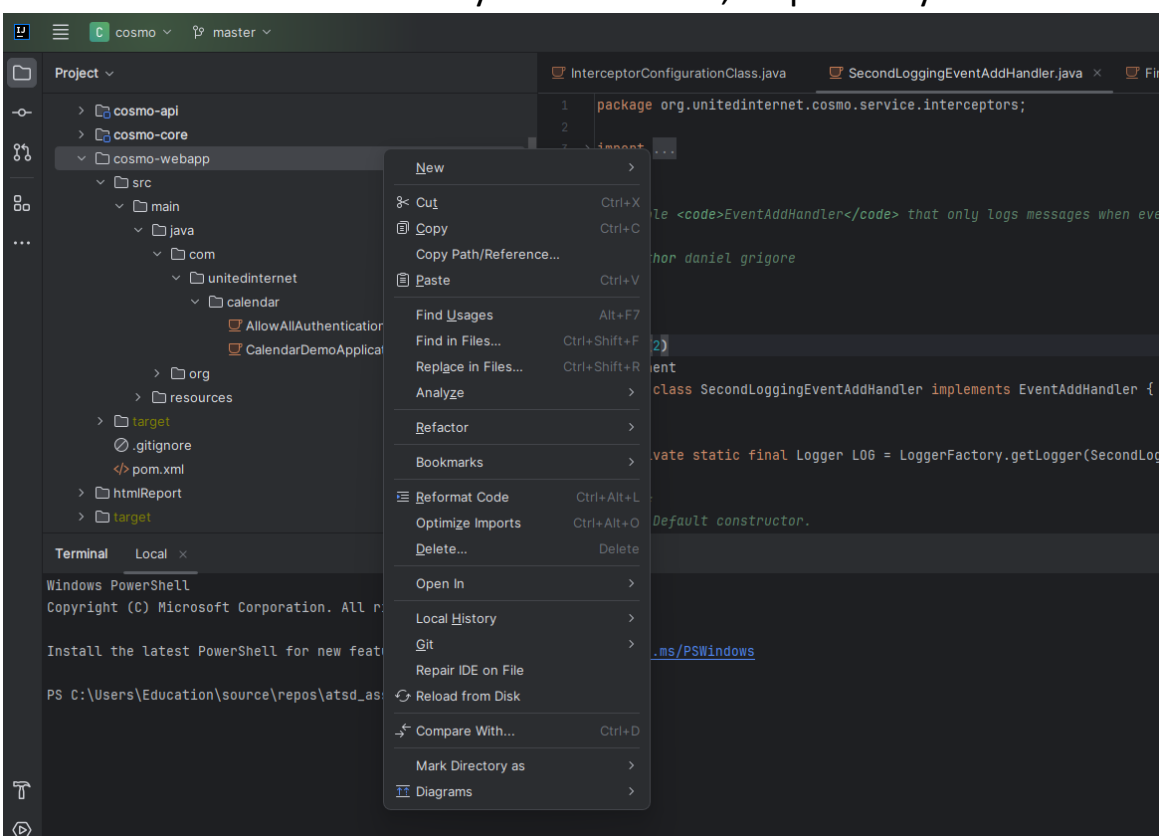


Figure 13: Junit not defined for all the packages

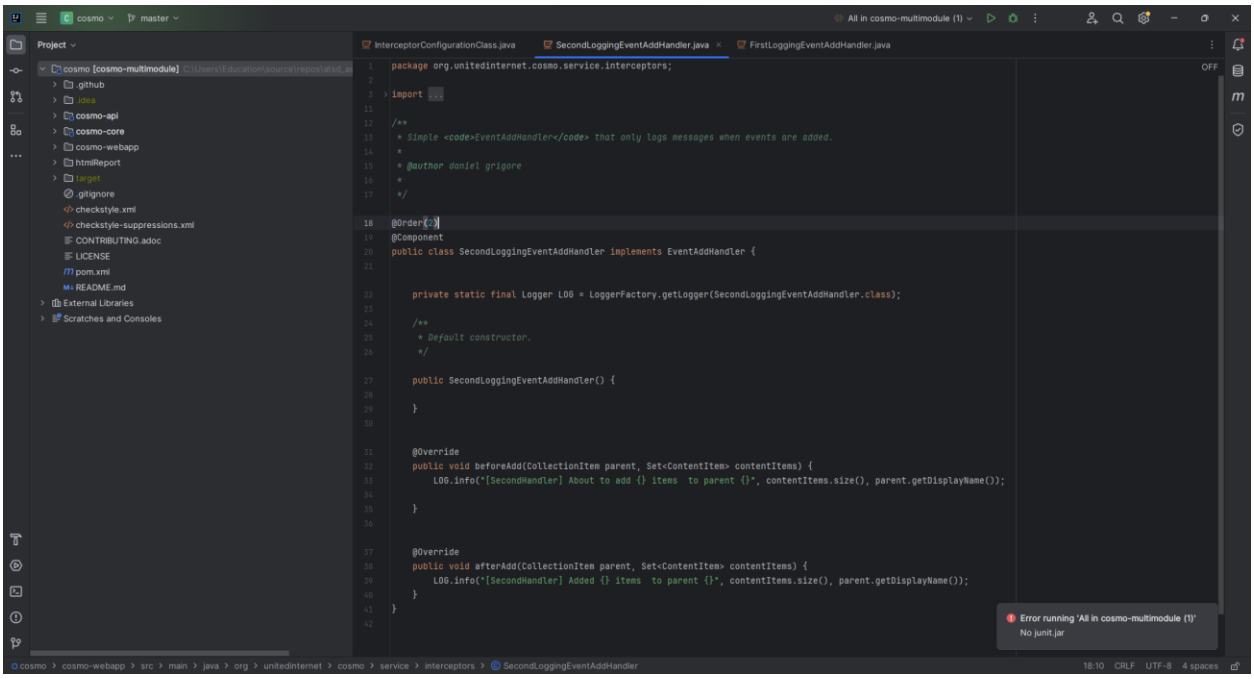


Figure 14: NO JUnit for all the java packages

<u>Weak aspect 2:</u> Junit testcase is dependent on external application.

StandardItemFilterProcessorTest test class has dependencies with a 3rd party application (mySql), It is failing to run the test because when I attempt to run the test it is trying to create a miniature version of maria DB to run the test and open it in port: 33060. If this port is in use, the test build won't run successfully. In order to run the test I have to close the mysql instance from task manager every time.

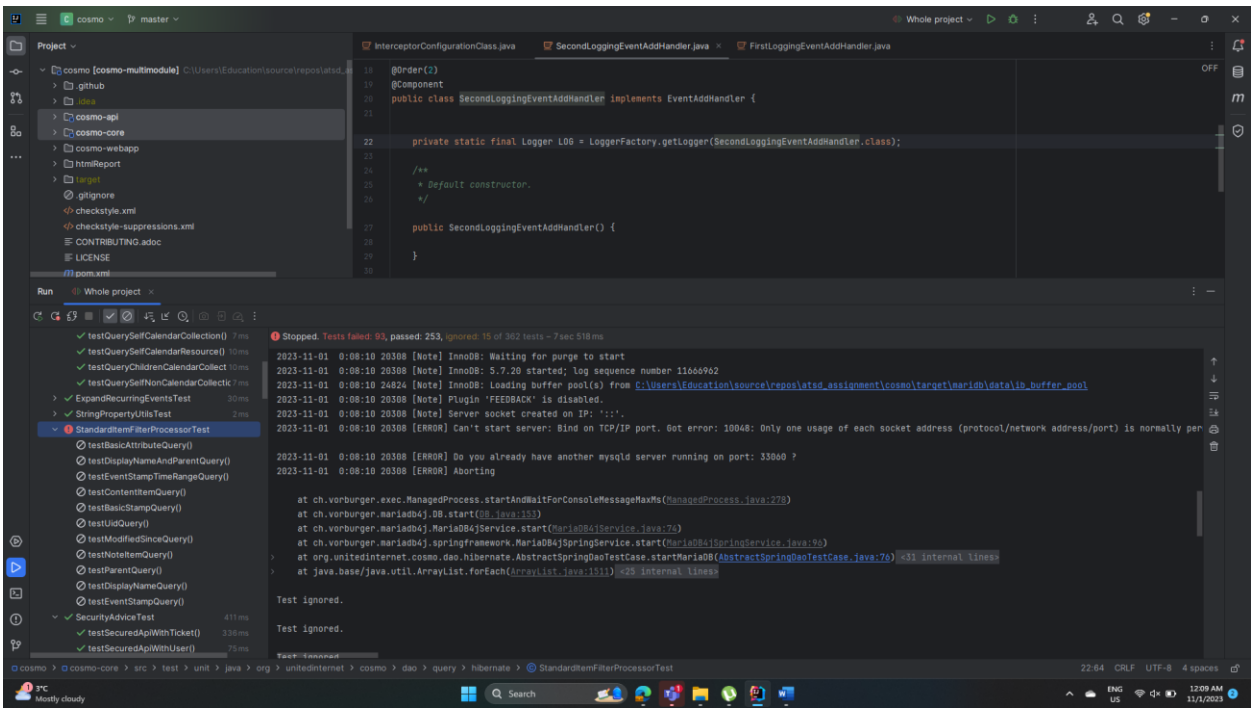


Figure 15: JUnit test dependent on external application

Weak aspect 3:

In JUnit 4, we use the @Test annotation with an expected exception class type, as shown in the following example: @Test(expected = IndexOutOfBoundsException.class). In JUnit 5, we can use assertThrows to assert an exception.

However, in the Cosmo application, the developer has handled the exception using a try-catch block, which is not the correct way to handle it. When we are unit testing, our goal is to find errors and situations where exceptions are raised.

```
SpaceUsageReportTest.addLineItemsTest

CollectionPathTest.java   ServerUtils.java   StatusSnapshot.java   SpaceUsageReport.java   HibHomeCollectionItem.java   SpaceUsageRep

1   /.../
16  package org.unitedinternet.cosmo.server;
17
18  import ...
23
24  /**
25   * Test case for <code>CollectionPath</code>.
26   */
    Corneliu Dobrota +1
27  public class CollectionPathTest {
28
29      /**
30       * Tests absolute url path.
31       * @throws Exception - if something is wrong this exception is thrown.
32       */
        Corneliu Dobrota +1
33      @Test
34      public void testAbsoluteUrlPath() throws Exception {
35          String badUrlPath = "http://dead.beef/";
36          try {
37              CollectionPath.parse(badUrlPath);
38              fail("absolute urlPath parsed successfully");
39          } catch (IllegalArgumentException e) {}
40      }
41
42      /**
43       * Tests successful parse.
44       * @throws Exception - if something is wrong this exception is thrown.
45       */
        Corneliu Dobrota +1
46      @Test
47      public void testSuccessfulParse() throws Exception {
48          String urlPath = "/collection/deadbeef";
49          CollectionPath cp = CollectionPath.parse(urlPath);
50          assertNotNull(cp, message: "path did not parse successfully");
51      }
```

Figure 16: Improper way of asserting exception

Strong aspect 1:

The developer has maintained a proper testing structure, even though he didn't comment on the AAA steps. It is properly structured in various places, and he has used a global arrange step, which serves as a base for all the tests within the test class.

For example, in the ICalendarUtilsTest file below, he has defined a variable TimeZoneRegistry and arranged the value for the object so that it will be used in all the tests below.

Act and assert steps are also properly structured one after the other.

```
32  public class ICalendarUtilsTest {
        9 usages
33      private static final TimeZoneRegistry TIMEZONE_REGISTRY =
34          TimeZoneRegistryFactory.getInstance().createRegistry();
35
36      /**
37       * Tests normalize UTC dateTime to date.
38       * @throws Exception - if something is wrong this exception is thrown.
39       */
        Corneliu Doorota +1
40      @Test
41      public void testNormalizeUTCDateTimeToDate() throws Exception {
42          TimeZone tz = TIMEZONE_REGISTRY.getTimeZone( s: "America/Chicago");
43
44          DateTime dt = new DateTime( value: "20070201T070000Z");
45
46          assertEquals( expected: "20070201", ICalendarUtils.normalizeUTCDateTimeToDate(dt, tz).toString());
47
48          tz = TIMEZONE_REGISTRY.getTimeZone( s: "America/Los_Angeles");
49          assertEquals( expected: "20070131", ICalendarUtils.normalizeUTCDateTimeToDate(dt, tz).toString());
50
51          tz = TIMEZONE_REGISTRY.getTimeZone( s: "Australia/Sydney");
52          assertEquals( expected: "20070201", ICalendarUtils.normalizeUTCDateTimeToDate(dt, tz).toString());
53      }
54
55      /**
56       * Tests compare dates.
57       * @throws Exception - if something is wrong this exception is thrown.
58       */
        Corneliu Doorota +1
59      @Test
60      public void testCompareDates() throws Exception {
61          TimeZone tz = TIMEZONE_REGISTRY.getTimeZone( s: "America/Chicago");
62
63          DateTime dt = new DateTime( value: "20070201T070000Z");
64          Date toTest = new Date( value: "20070201");
65
66          assertEquals( expected: -1, ICalendarUtils.compareDates(toTest, dt, tz));
```

Figure 17: AAA Steps

Strong aspect 2:

Every test method and test class have Javadoc comments explaining its type and functionalities. The naming convention for the tests is also properly followed across the codebase. Every test class is suffixed with the keyword "Test," and every test method starts with the keyword "test" followed by specific test details. The test method and class names themselves follow Java naming conventions, i.e., camelCase for methods and each word's first letter capitalized for classes.

```
CalendarQueryFilterTest.java   ICalendarUtilsTest.java   InstanceListTest.java   RecurrenceExpanderTest.java

/.../
package org.unitedinternet.cosmo.calendar;

import ...

/**
 * Test RecurrenceExpander.
 *
 */
Corneliu Dobrota +2
public class RecurrenceExpanderTest {

    1 usage
    private static final TimeZoneRegistry TIMEZONE_REGISTRY =
        TimeZoneRegistryFactory.getInstance().createRegistry();

    /**
     * Tests recurrence expander all day.
     * @throws Exception - if something is wrong this exception is thrown.
     */
    Corneliu Dobrota +1
    @Test
    public void testRecurrenceExpanderAllDay() throws Exception {
        RecurrenceExpander expander = new RecurrenceExpander();

        Calendar calendar = getCalendar( name: "allday_recurring1.ics");

        Date[] range = expander.calculateRecurrenceRange(calendar);

        assertEquals( expected: "20070101", range[0].toString());
        assertEquals( expected: "20070120", range[1].toString());

        calendar = getCalendar( name: "allday_recurring2.ics");

        range = expander.calculateRecurrenceRange(calendar);

        assertEquals( expected: "20070101", range[0].toString());
        assertNull(range[1]);
    }
```

Figure 18: Proper comments

Strong aspect 3:

In the application's test suite, each test is designed to run independently of the others, ensuring that the results of one test do not affect the results of another. This independent structure is crucial for identifying specific issues and ensuring the robustness of the application.

Among the various test classes in the application, there is one specific test class defined for StandardItemFilterProcessor. Unfortunately, this particular test class is currently failing, and the root cause of this failure has been identified as a dependency on MySQL. This dependency issue is preventing the StandardItemFilterProcessor test class from executing successfully.

However, it is important to note that not all test classes are affected by this MySQL dependency issue. Other test files, such as ExpandRecurringEventsTest and StringPropertyUtilsTest, are executing as expected and are not experiencing any failures related to this dependency problem. This indicates that the issue is isolated to the StandardItemFilterProcessor test class and does not impact the entire test suite.

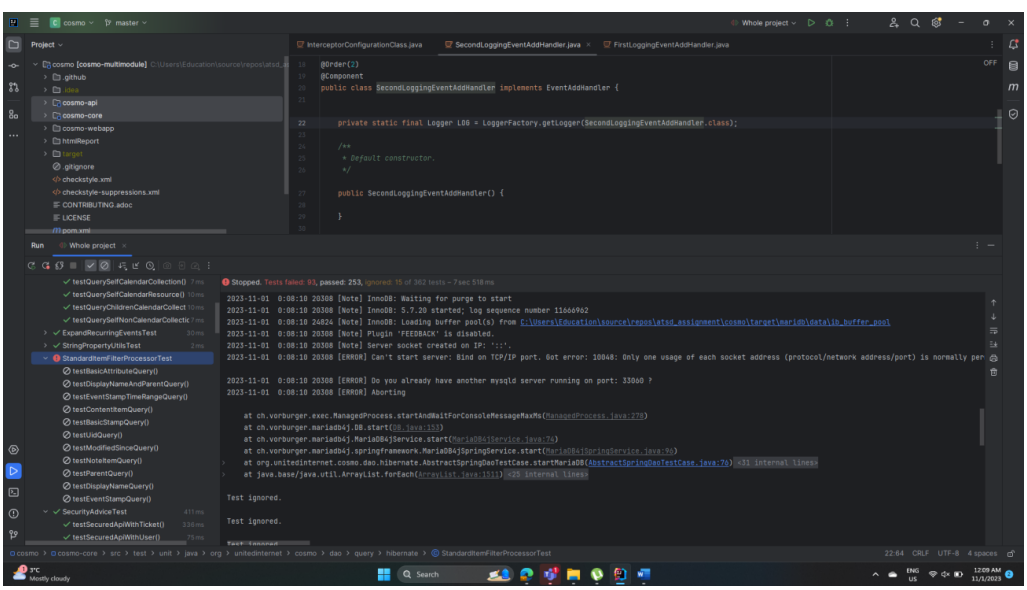
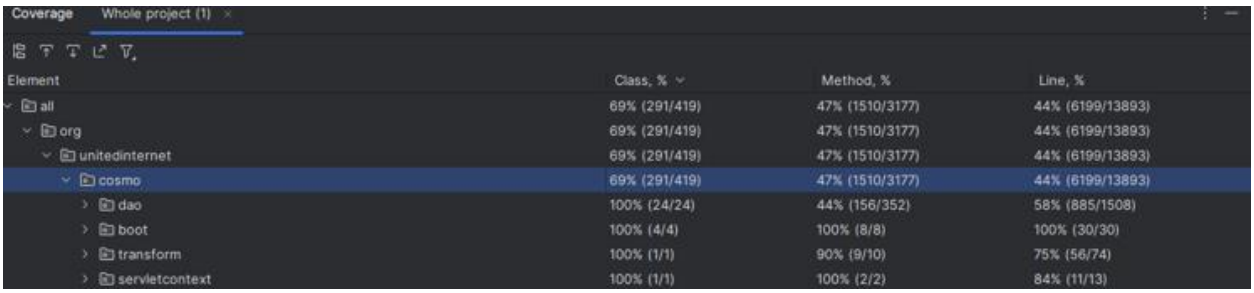


Figure 19: Independent test cases

Task #4:

Implement at least three new tests for the repository. It could be for new source code elements (new class or method) or for existing code. The newly added tests must not fail due to compilation issues; however, it is fine if they identify a new bug in the project.

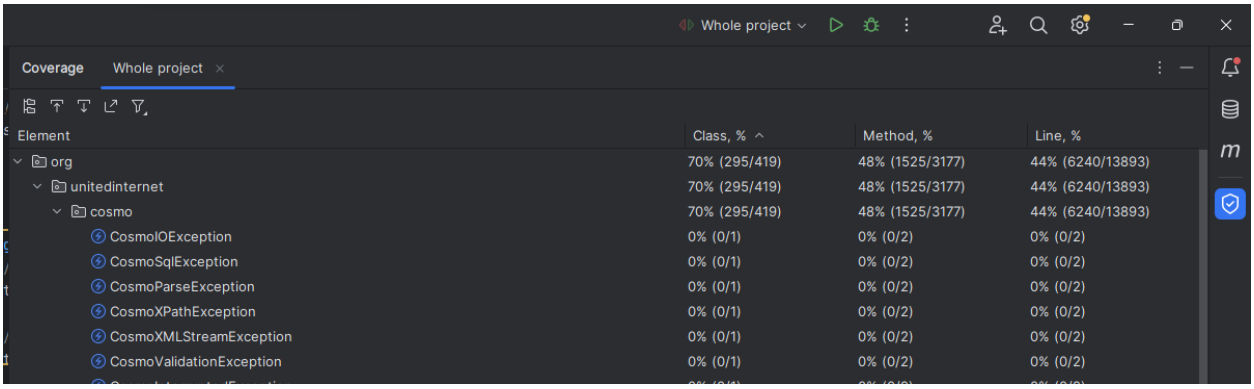
Before including my test cases, the class code coverage was 69% and method coverage 47%.



Element	Class, %	Method, %	Line, %
all	69% (291/419)	47% (1510/3177)	44% (6199/13893)
org	69% (291/419)	47% (1510/3177)	44% (6199/13893)
unitedinternet	69% (291/419)	47% (1510/3177)	44% (6199/13893)
cosmo	69% (291/419)	47% (1510/3177)	44% (6199/13893)
dao	100% (24/24)	44% (156/352)	58% (885/1508)
boot	100% (4/4)	100% (8/8)	100% (30/30)
transform	100% (1/1)	90% (9/10)	75% (56/74)
servletcontext	100% (1/1)	100% (2/2)	84% (11/13)

Figure 20: code coverage increase

After including my test cases, the class code coverage increased to 70% and method coverage 48%.



Element	Class, %	Method, %	Line, %
org	70% (295/419)	48% (1525/3177)	44% (6240/13893)
unitedinternet	70% (295/419)	48% (1525/3177)	44% (6240/13893)
cosmo	70% (295/419)	48% (1525/3177)	44% (6240/13893)
CosmoIOException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoSqlException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoParseException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoXPathException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoXMLStreamException	0% (0/1)	0% (0/2)	0% (0/2)
CosmoValidationException	0% (0/1)	0% (0/2)	0% (0/2)

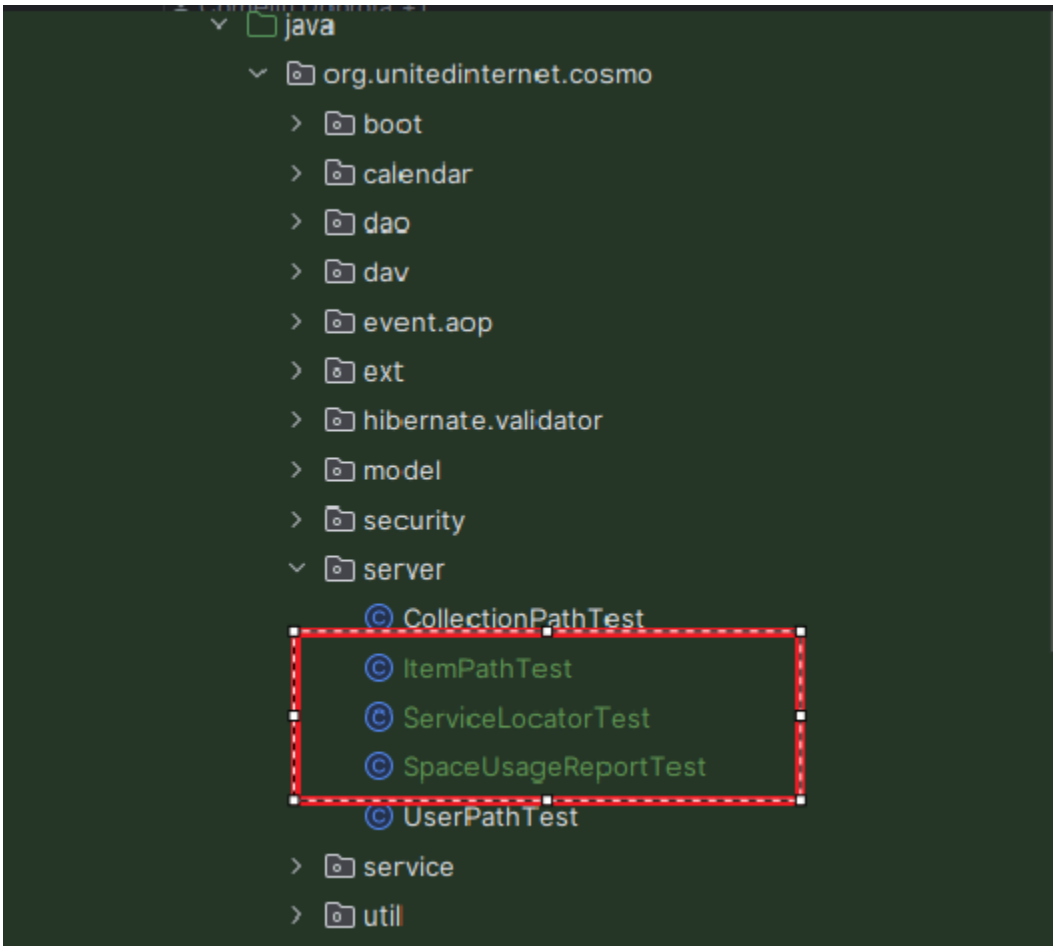
Figure 21: method code coverage increase

I have created 3 test classes,

ServiceLocatorTest contains 1 testcase.

SpaceUsageReportTest contains 1 testcase.

ItemPath contains 6 testcases.



Following are my testcases:

In ServiceLocatorTest.java,

```
ItemPathTest.java   ServiceLocatorTest.java ×   SpaceUsageReportTest.java   CalendarQueryFilterTest.java   ICale

class ServiceLocatorTest {
    2 usages
    ServiceLocator serviceLocator;
    3 usages
    ServiceLocatorFactory factory;
    2 usages
    String appMountUrl;
    3 usages
    User testUser;

    /**
     * Constructor for ServiceLocatorTest class.
     * contains code to arrange objects for test
     */
    new *
    public ServiceLocatorTest(){
        this.appMountUrl = "/appMountUrlTest";
        this.factory= Mockito.mock(ServiceLocatorFactory.class);
        Mockito.when(this.factory.getDavPrefix()).thenReturn( t: "/davPrefix");
        this.serviceLocator = new ServiceLocator(this.appMountUrl,factory);
        this.testUser=Mockito.mock(User.class);
    }

    /**
     * get DAV Calendar home url test
     */
    new *
    @Test
    void testGetDavCalendarHomeUrl() {
        //Arrange
        Mockito.when(this.testUser.getUsername()).thenReturn( t: "testUserName");

        //Act
        String actualResult = this.serviceLocator.getDavCalendarHomeUrl(this.testUser);

        //Assert
        assertEquals( expected: "/appMountUrlTest/davPrefix/testUserName/",actualResult);
    }
```

In SpaceUsageReportTest.java,

```java
package org.unitedinternet.cosmo.server;

import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
import org.unitedinternet.cosmo.model.HomeCollectionItem;
import org.unitedinternet.cosmo.model.User;
import org.unitedinternet.cosmo.model.hibernate.HibHomeCollectionItem;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Space usage report test class
 */
class SpaceUsageReportTest {

    /**
     * test add line items.
     */
    @Test
    void testAddLineItems() {
        //Arrange
        HomeCollectionItem homeCollectionItem=new HibHomeCollectionItem();
        User testUser = Mockito.mock(User.class);
        Mockito.when(testUser.getUsername()).thenReturn( t: "testUserName");

        //Act
        SpaceUsageReport SpaceUsageReport= new SpaceUsageReport(testUser,homeCollectionItem);

        //Assert
        assertEquals( expected: 1, SpaceUsageReport.getLineItems().size());
    }

}
```

In ItemPathTest.java,

```java
package org.unitedinternet.cosmo.server;

import ...

/**
 * ItemPath class test file
 */
class ItemPathTest {

    /**
     * get item path object - throws IllegalStateException
     */
    @Test
    void testGetItemPathNotAnItemPath() {
        //Arrange
        String url = "/test";

        //Act
        Exception exception = assertThrows(IllegalStateException.class, () -> {
            ItemPath itemPath = new ItemPath(url);
        });

        String expectedMessage = "urlPath is not an item path";
        String actualMessage = exception.getMessage();

        //Assert
        assertTrue(actualMessage.contains(expectedMessage));
    }
```

```java
    /**
     * Get item path object throws IllegalArgumentException
     */
    @Test
    void testGetItemPathMustStartWithSlash() {
        //Arrange
        String url = "test";

        //Act
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            ItemPath itemPath = new ItemPath(url);
        });

        String expectedMessage = "urlPath must start with /";
        String actualMessage = exception.getMessage();

        //Assert
        assertTrue(actualMessage.contains(expectedMessage));
    }
```

```java
    /**
     * Get Item Path object with correct input
     */
    new *
    @Test
    void getItemPath() {
        //Arrange
        String url = "/item/testUid/testPathInfo";

        //Act
        ItemPath itemPath = new ItemPath(url);


        //Assert
        assertEquals( expected: "/testPathInfo", itemPath.getPathInfo());
        assertEquals( expected: "testUid", itemPath.getUid());
        assertEquals(url, itemPath.getUrlPath());
    }

    /**
     * Parse url test - return null Item Path
     */
    new *
    @Test
    void parseDoNotAllowPathInfo() {
        //Arrange
        String url = "/item/testUid/testPathInfo";

        //Act
        var itemPath = ItemPath.parse(url);

        //Assert
        assertNull(itemPath);
    }
```

```java
    /**
     * Parse url test - return Item Path Object
     */
    new *
    @Test
    void parseAllowPathInfo() {
        //Arrange
        String url = "/item/testUid/testPathInfo";

        //Act
        var itemPath = ItemPath.parse(url,  allowPathInfo: true);

        //Assert
        assertNotNull(itemPath);
    }
```

All the new testcases does not have any compilation error and executed successfully.

Main code was building without any compilation error and code is running successfully at port: 8080