**SE 2226 Software Quality Assurance and Testing**



**YAŞAR UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF SOFTWARE ENGINEERING**

**SE 2226 TERM PROJECT**

**Report Sheet**

**Group Members:**

**Muhamed CICAK 21070006208**

**Melis ÖZVERİ** **Y20070002011**

[**1.** **INTRODUCTION** 2](#_Toc135087319)

[**2.** **APPLICATION** 3](#_Toc135087320)

[**3.** **BLACK BOX TECHNIQUES** 6](#_Toc135087321)

[**3.1** **Equivalence Class Partitioning and Boundary Value Analysis (BVA)** 6](#_Toc135087322)

[**3.2** **Decision Tables** 9](#_Toc135087323)

[**4. UNIT TESTS & AUTOMATED TESTING** 10](#_Toc135087324)

[**4.1 Code Structure** 10](#_Toc135087325)

[**4.2 Appointment Reservation Test** 12](#_Toc135087326)

[**4.3 Adding a shift Test** 13](#_Toc135087327)

[**4.4 Adding a Working Exception Test** 15](#_Toc135087328)

[**4.5 Registering a User Test** 17](#_Toc135087329)

[**4.6 Running Tests** 18](#_Toc135087330)

[**5. CONCLUSION** 20](#_Toc135087331)

1. **INTRODUCTION**

It is aimed to choose an application and test this application in this project. The chosen application for this report is a website called zamaninda.com. 4 pages from this website will be tested. Details of this website will be discussed later on. Test cases will be generated using black box techniques such as Equivalence Class Partitioning, Boundary Value Analysis and Decision Table. Then, unit tests will be created according to test cases mentioned earlier. In addition to these, the unit tests will be automated as well. In addition to these, load testing will be applied as well.

1. **APPLICATION**

zamaninda.com is the subject of this report. Zamaninda.com aims to be a bridge between customers and businesses. Customers can book appointments with different businesses such as hairdressers, nail salons, and doctors. Business owners can set the schedules, see the appointments, set any exceptions with working hours on holidays, and so on. 4 pages of this website are the subject of this report. These can be listed as:

1. **Appointment Reservation Page:** Customers can reserve appointments from their preferred service provider. Their chosen appointment time needs to be between the appropriate shift hours and their appointment can be 20 minutes long. Equivalence Class Partitioning and Boundary Value Analysis methods will be used to create test cases for this page. The page for selection of period for appointment reservation and confirmation message screen can be found below:

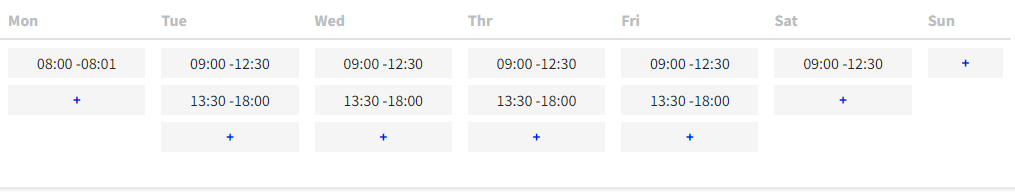


**Figure 1** Period choices for appointment reservation

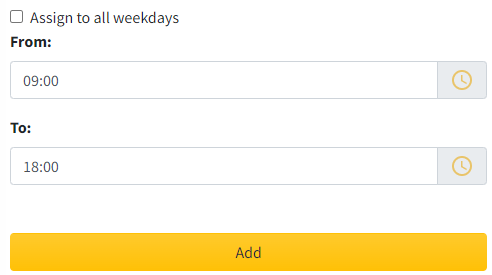


**Figure 2** Confirmation message saying that appointment is placed successfully

1. **Adding Shift Page:** Business owners can add shifts for their businesses through this page. The starting and the ending time of the shift must be between the appropriate working hours of the business, and it should not crash with another shift that exists for that day. Also, business owners can add shifts to the whole week. The chosen hours for the shift need to be free for the whole week. Equivalence Class Partitioning and Boundary Value Analysis methods will be used to create test cases for this page. Pages of Adding Shift can be found below:

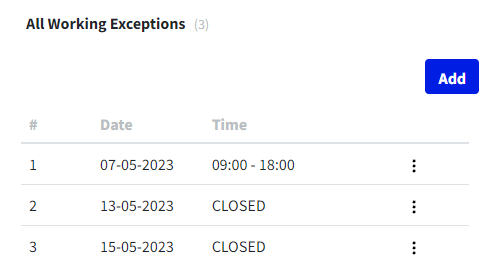


**Figure 3** Adding shift page in zamaninda.com

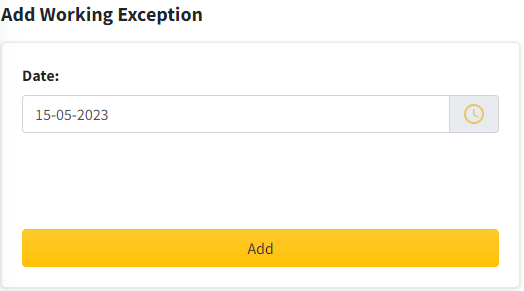


**Figure 4** Choosing the beginning and the ending hours of the shift. In addition to this, this shift can be added to the whole week if it is possible

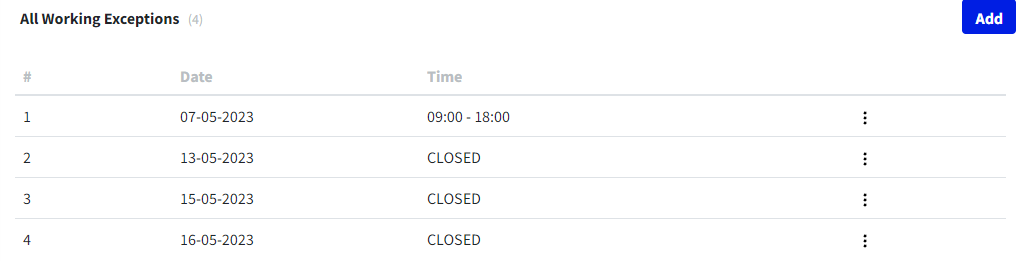
1. **Adding Work Exception Page:** Business owners can add work exception days for their businesses. These are the days when the business has the day off. The chosen day needs to be between tomorrow and 1 year in advance from today. In addition to this, the chosen day cannot be a work exception day already. Equivalence Class Partitioning and Boundary Value Analysis methods will be used to create test cases for this page. Pages of Adding Work Exception can be found below:



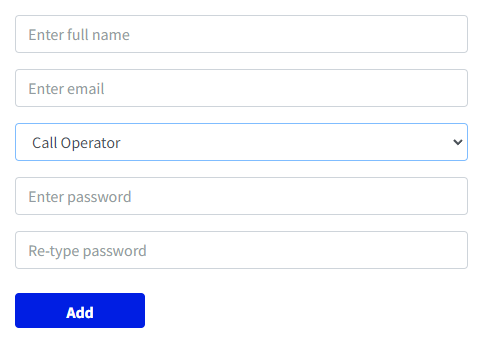
**Figure 5** Adding work exception page displays the existing work exception days. New exception days can be added through the “add” button



**Figure 6** The date can be chosen as work exception day. It can be confirmed through “add” button

**Figure 6** The updated list of work exceptions is displayed to the user

1. **User Registration:** Users can register to this system with their valid emails and valid passwords. The password must be longer than 8 characters and match the password confirmation. The Decision Table method will be used to create test cases for this page. The user registration page can be found below:



**Figure 7** The user registration page

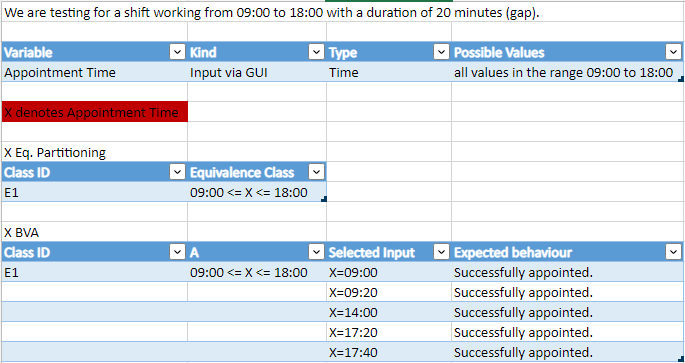
1. **BLACK BOX TECHNIQUES**

Equivalence Class Partitioning, Boundary Value Analysis, and Decision Table methods were used for generating test cases for the pages mentioned earlier in this report. Full versions of the techniques mentioned below can be found in the appendix at the end of this report.

* 1. **Equivalence Class Partitioning and Boundary Value Analysis (BVA)**

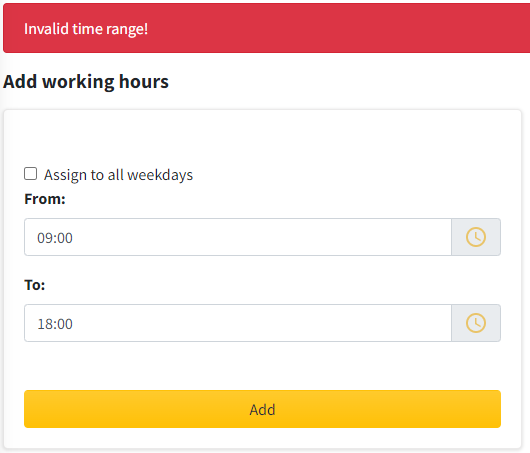
Both methods are used together to create test cases for all pages except the User Registration page. So, these methods were used for Appointment Reservation, Adding Shift, and Adding Working Exceptions pages.

For the Appointment Reservation page, the chosen time needs to be between 9:00 and 18:00. These are the default working hours defined by the system. The periods are also predefined as 20 minutes. After the period is chosen, the user receives a message that tells the user has successfully placed an appointment. The Equivalence Class Partitioning and BVA for this page can be found below:



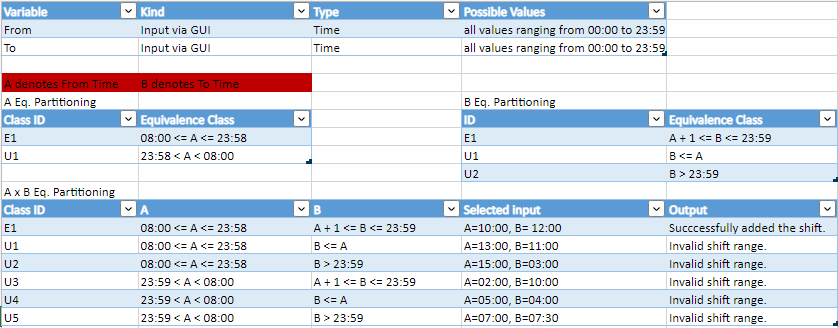
**Table 1** Equivalence Class Partitioning and BVA for Appointment Reservation page

For the Adding Shift page, there are 2 variables which are the starting time of the shift and the ending time of the shift. Starting time can be between 8:00 and 23:58. The ending time needs to be after the starting time and before 23:59. If conditions are met, then the shift is added successfully. Otherwise, an error message is given to the user. The error message given to the user when their entered from time and/or end time is not valid can be found below:

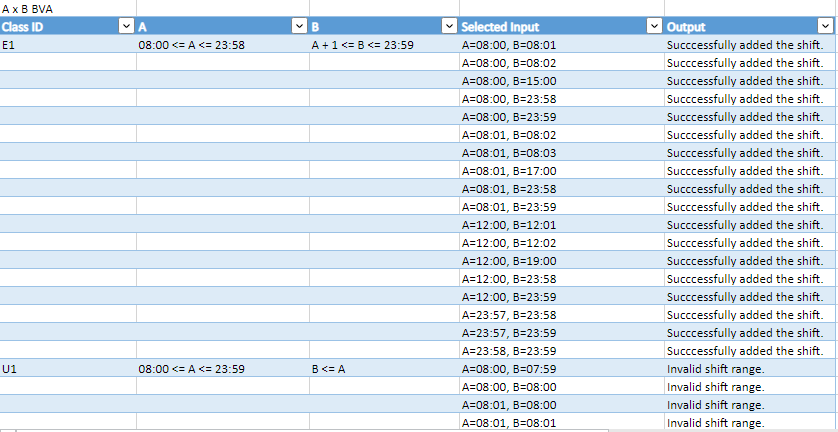


**Figure 8** Error message of the Adding Shift page. 9:00 and 18:00 are the default times given by the system

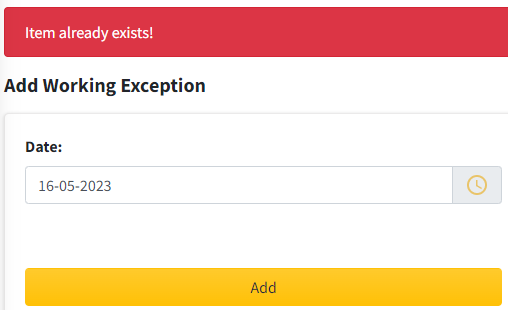
Equivalence Class Partitioning for this page can be found below:

**Table 2** Equivalence Class Partitioning for Adding Shift page

Some parts of the BVA for this page can be found below:

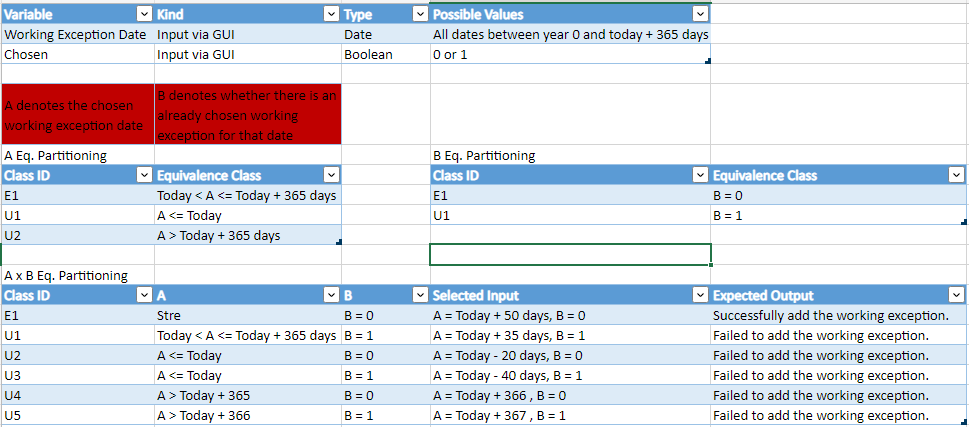
**Table 3** Beginning of the BVA for Adding Shift page

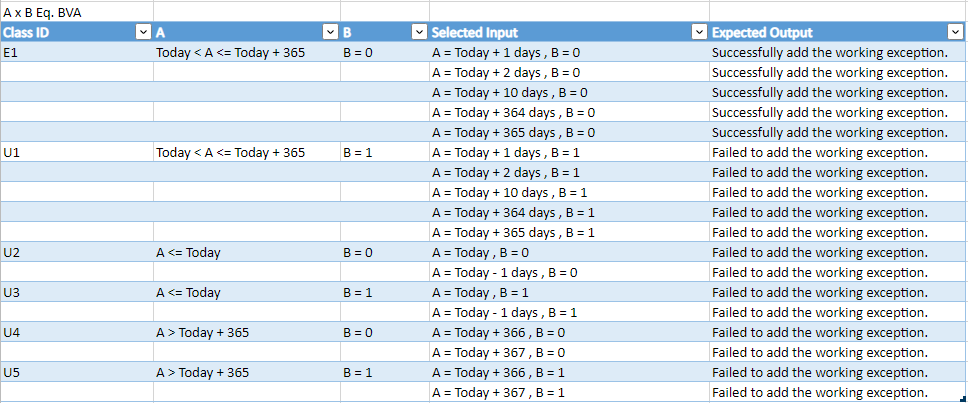
Lastly, the Adding Working Exception page has test generations with Equivalence Class Partitioning and BVA. There are 2 variables which are named Working Exception Date and Chosen. Working Exception Date denotes the date of the day chosen by the user and Chosen denotes if the Day is already a work exception day. If Day is between tomorrow and 1 year in advance (inclusive), and Day is not already a work exception day, then the work exception day is successfully added. Otherwise, an error message is given to the user and the day is not added as a work exception day. The error message given to the user when an already existing work exception day is tried to be added again can be found below:



**Figure 9** Error message of the Adding Work Exception page

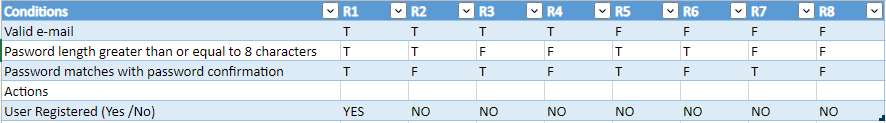
Equivalence Class Partitioning and BVA for this page can be found below:

**Table 4** Equivalence Class Partitioning for Adding Shift page

**Table 5** Equivalence Class Partitioning for Adding Shift page

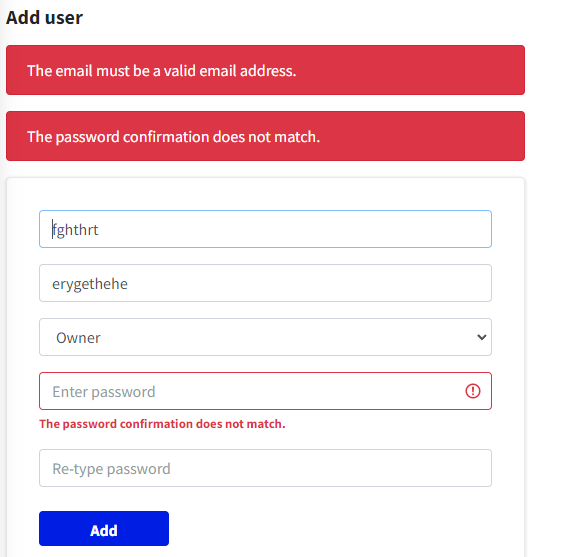
* 1. **Decision Tables**

The decision Table method was only used for the Registering User page to generate test cases. The decision table was used to generate tests for the Registering User page. A user needs a valid e-mail address and a valid password longer than 8 characters and matches the password confirmation to register to this website. The relationships between the conditions and their effect on the outcome can be seen in the decision table below:



**Table 6** Decision Table of Registering User page

The error messages that can be given by the Registering User page can be found below:



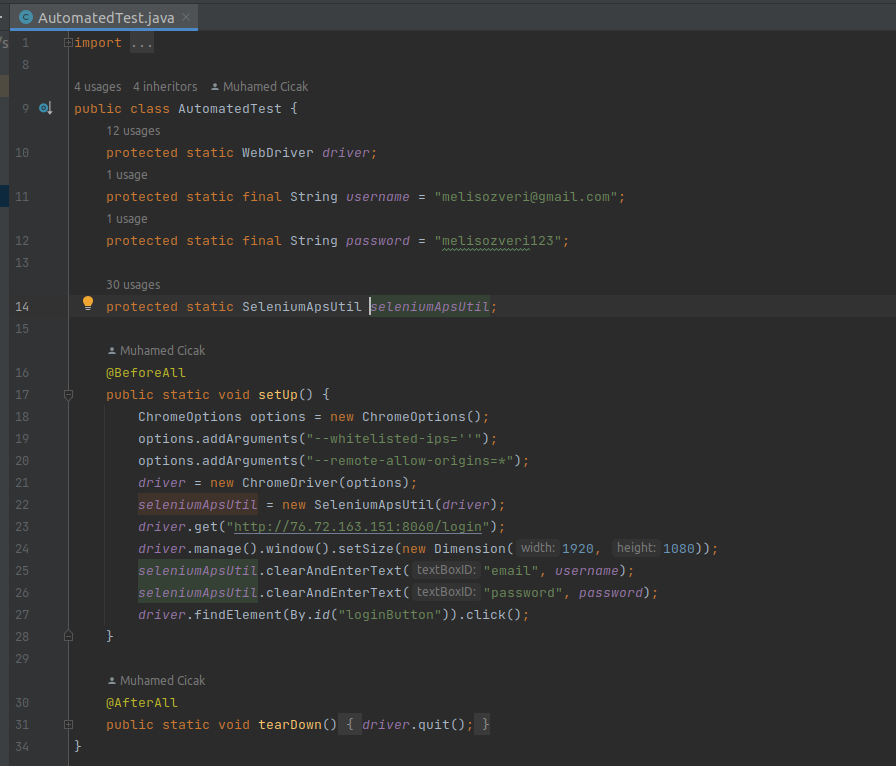
**Figure 10** Error messages of the Registering User Page. The first error is given when the email address is not valid. Second error message is given when the password confirmation does not match

# **4. UNIT TESTS & AUTOMATED TESTING**

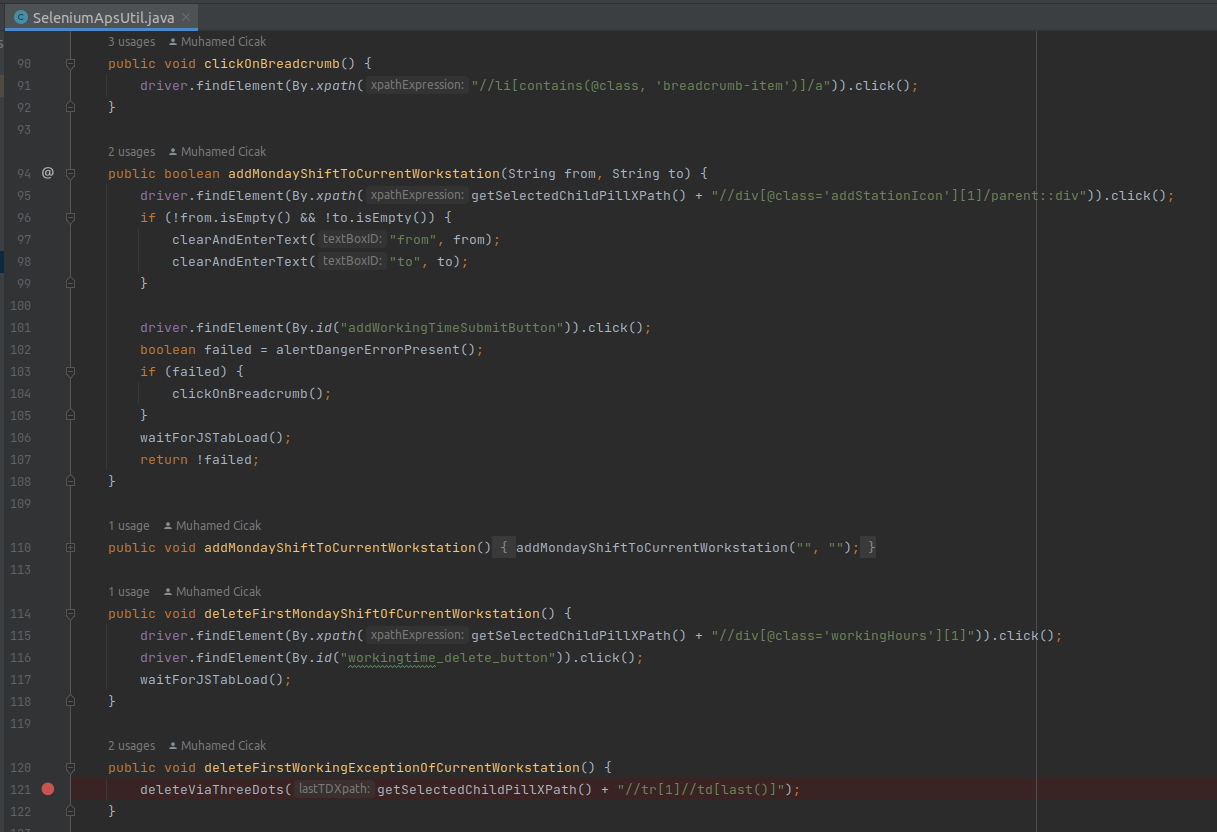
Manual testing is time consuming and prone to human error. Since a tester must manually enter the input while cross-checking the formal methods provided, s/he loses time and has a high chance of making a mistake. Manual testing proves especially difficult in case of any software changes, as the tests must be repeatedly performed. To solve these issues, we propose automated unit tests. Having thus formally written down the tests using the BVA method and the decision table method, we can easily transform them into unit tests, using JUnit, and programmatically interact with the website using Selenium. We will explain how these transformations take place test by test, in order.

**4.1 Code Structure**

Before we dive into the source code of the automated unit tests, I would like to briefly explain the code structure. The code is compact and designed to be reusable. In total, we have 7 java class files. One is for the Main, 4 are for the tests (1 class for each test), and the remaining two are called AutomatedTest and SeleniumApsUtil. The AutomatedTest class is a class every test inherits from. It is the backbone of the upcoming unit tests. Its data members are comprised of an instance of the Selenium WebDriver, the username/password for the application and an instance of the SeleniumApsUtil class (which we will talk about later).



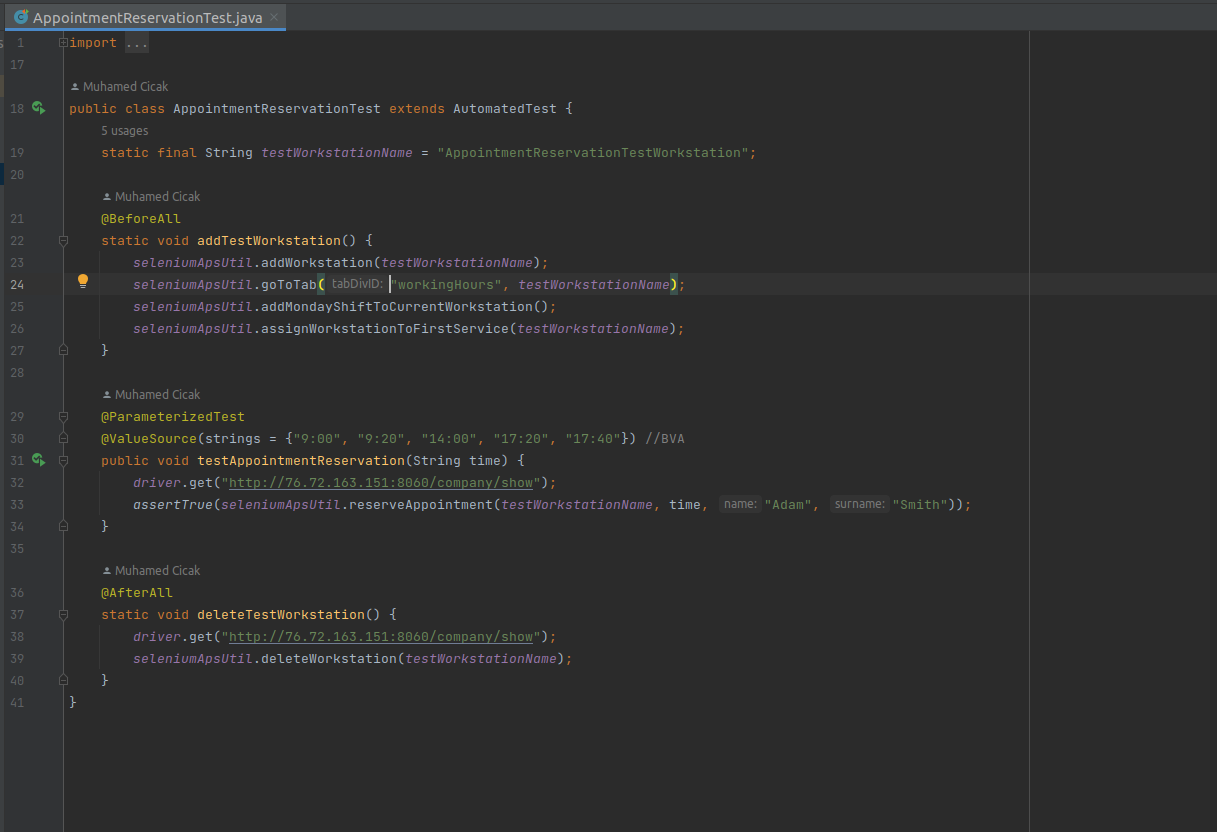
The AutomatedTest class sets up the Selenium WebDriver with needed configurations before any test cases are executed (using JUnit’s @BeforeAll annotation) and it logs into the application (every automated test case will need to be logged in to the application). It also quits the Selenium WebDriver after all the unit tests have finished. The SeleniumApsUtil class, on the other hand, is an isolated class that is a collection of reusable functions that repeat themselves throughout the interactions with this application. It also abstracts out a lot of Selenium-specific details and keeps the tests cleaner. It is 213 lines long, so I will just put a small part of it here to give you a general idea about it, you can see the whole code in the drive link we put at the end of this report.



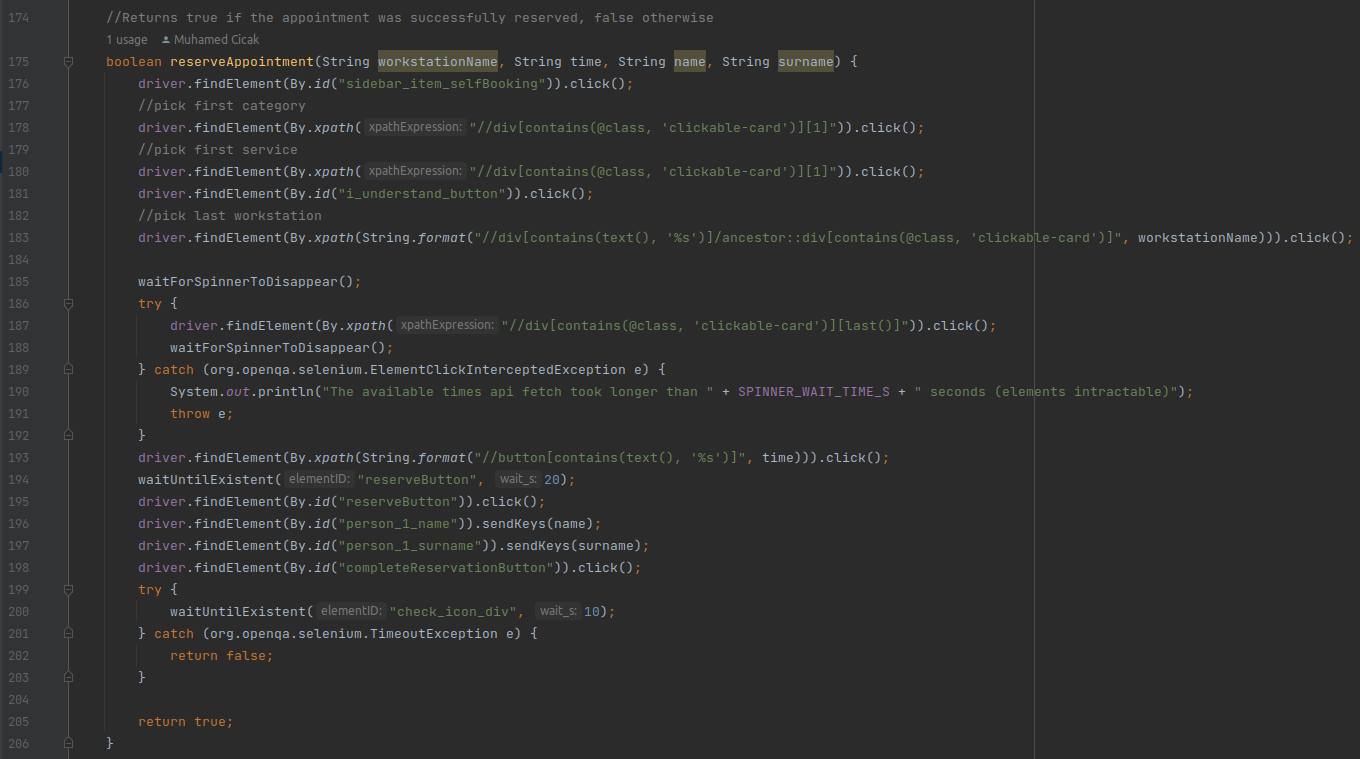
These functions are used throughout the tests and other functions inside the SeleniumApsUtil class. For instance, the clickOnBreadcrumb function clicks on the breadcrumb of the current page (so it assumes you are already on a page that has a breadcrumb). Other functions are self-explanatory by their names.

**4.2 Appointment Reservation Test**

This test’s unit test part was simple to implement due to it having small inputs to test, although the underlying selenium functions are long because of the number of screens one needs to traverse to reserve an appointment.



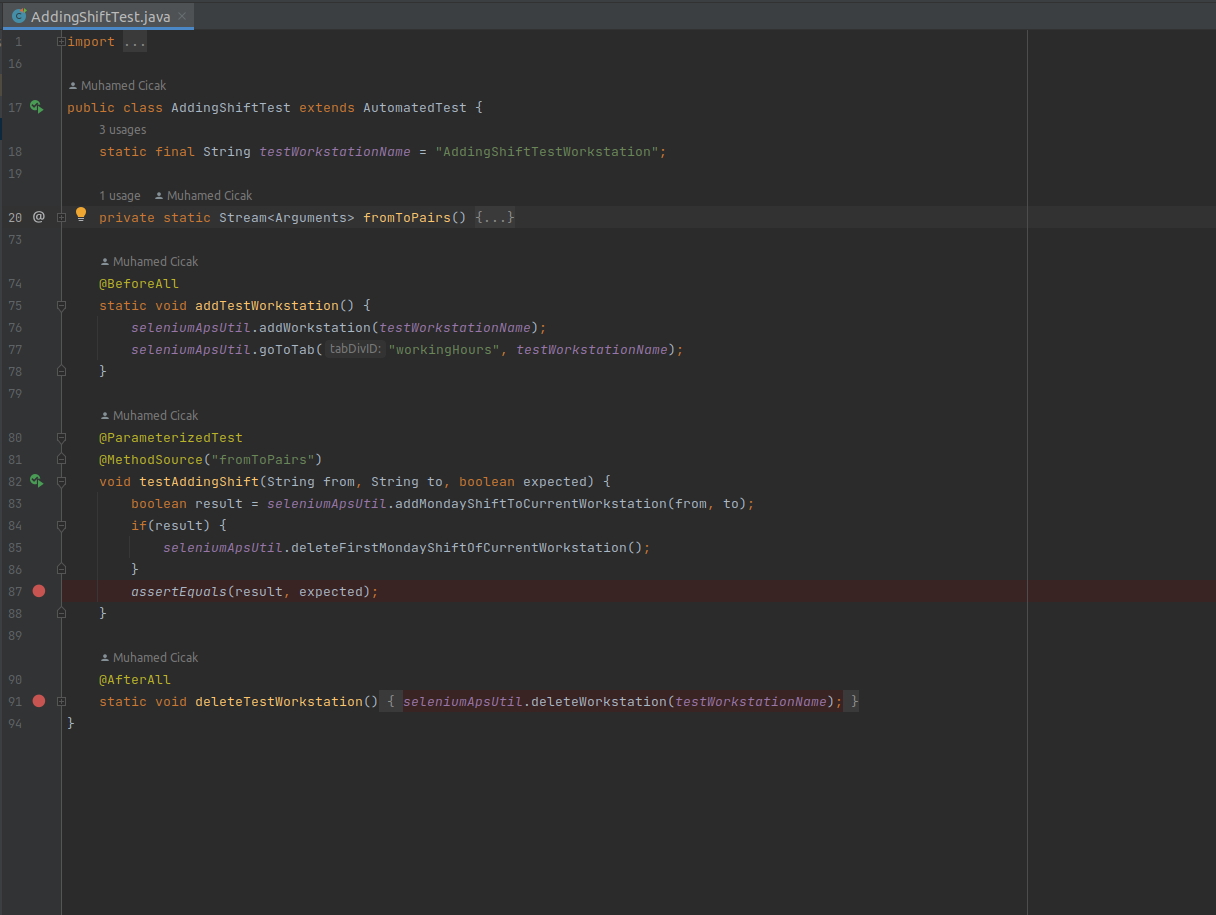
Above, you can see that we have three parts of this unit test: @BeforeAll, the unit test itself and @AfterAll part. The reason we need @BeforeAll and @AfterAll in this unit test is because we need to create a workstation for this particular test on which we will take the reservation, and in the @AfterAll part we delete this workstation (and the reservations together with it) so that the unit test does not change the state of the application. We will observe this pattern in AddingShiftTest and AddingWorkingExceptionTest as well. We also add a Monday standard shift and assign a service to the workstation in the @BeforeAll part, so that we can reserve an appointment on that workstation. The actual test process takes place in the testAppointmentReservation function. It is a @ParametrizedTest that has a @ValueSource of strings of inputs that we defined in our BVA process. Each string represents the test input reservation time. Below, we observe the seleniumApsUtil.reserveAppointment function



Here, we go step by step through the reservation process. One thing I would like to point out here is the use of waitForSpinnerToDissappear and waitUntilExistent functions. These functions are mandatory because there are animations and AJAX calls on the website that are done via JS and that are not loaded instantly. Had these functions not been there, the WebDriver would throw one of org.openqa.selenium.ElementNotInteractableException/ org.openqa.selenium.NoSuchElementException exceptions.

**4.3 Adding a shift Test**

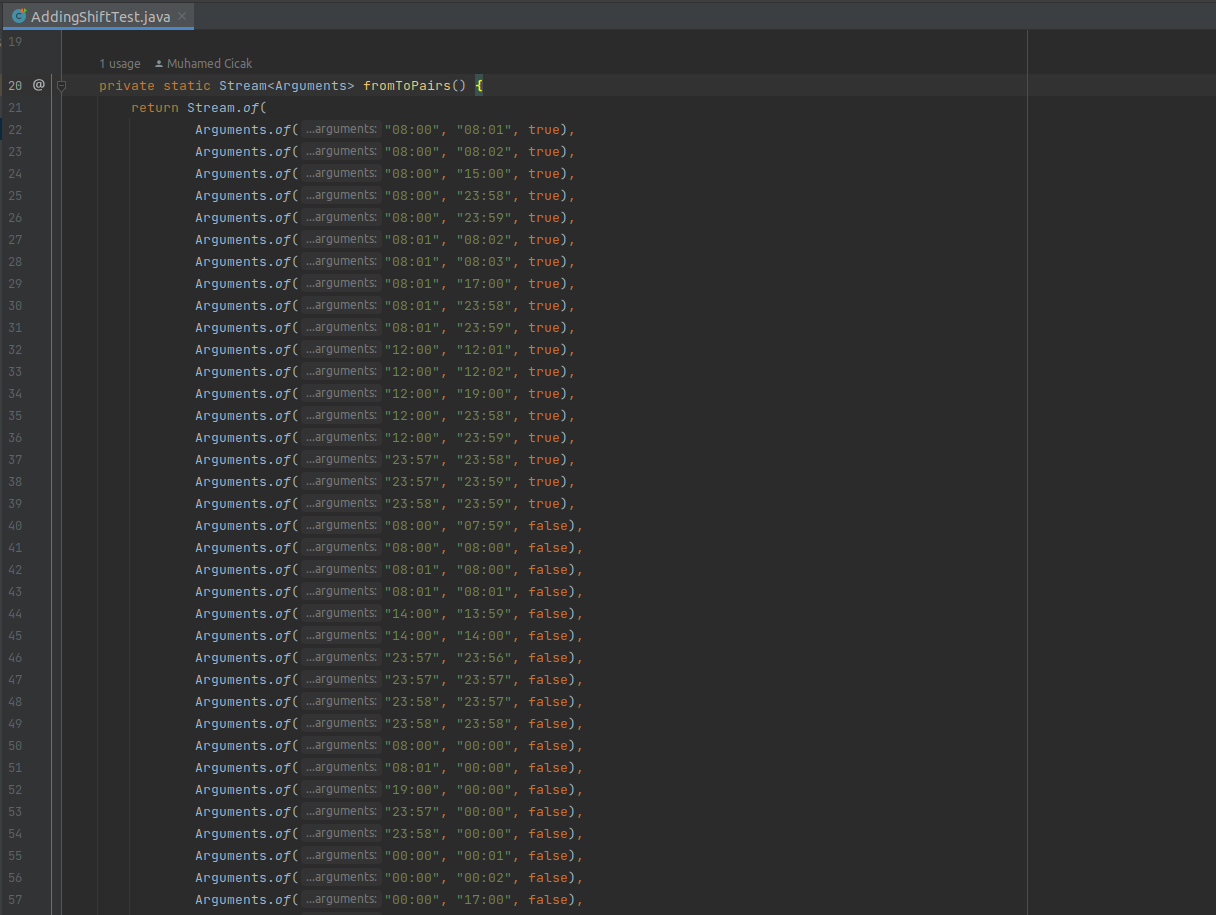
This test is easily the slowest test because of the number of inputs it must test. The BVA is vast (there are 49 test cases in total), so the performance is naturally affected. Although the number of inputs may seem daunting, since this is also a @ParametrizedTest its implementation is much simpler than that of, for example. AppointmentReservationTest. The underlying Selenium functions do not have to interact with as many inputs. In the same fashion as can be seen in the previous test, we here have three parts. The @BeforeAll and @AfterAll parts are the same, the only difference is that the workstation is saved under a different name (thus, it is a different workstation, this is so that the tests can run in parallel). It is also worth noting that after the new workstation is added in the @BeforeAll part, we navigate to the workingHours tab of that workstation. The actual unit test is inside the testAddingShift function. It is a @ParameterizedTest, but it does not use the @ValueSource annotation, instead, it uses the @MethodSource annotation where it is passed Arguments from a Stream of Arguments returned by the method passed in the @MethodSource annotation. This is because this unit test requires a bit more complex input structure, and @ValueSource annotation was not used so that we can spare ourselves from painful parsing. The function accepts three arguments (two Strings and a Boolean): from, to, and expected output. The from/to arguments are the inputs for the shift range that should be added, and the expected output argument is true if the add operation is expected to succeed, and false otherwise. Below is the code of the AddingShiftTest class.



The actual unit test function testAddingShift makes a call to a function we’ve seen before: seleniumApsUtil.addMondayShiftToCurrentWorkstation

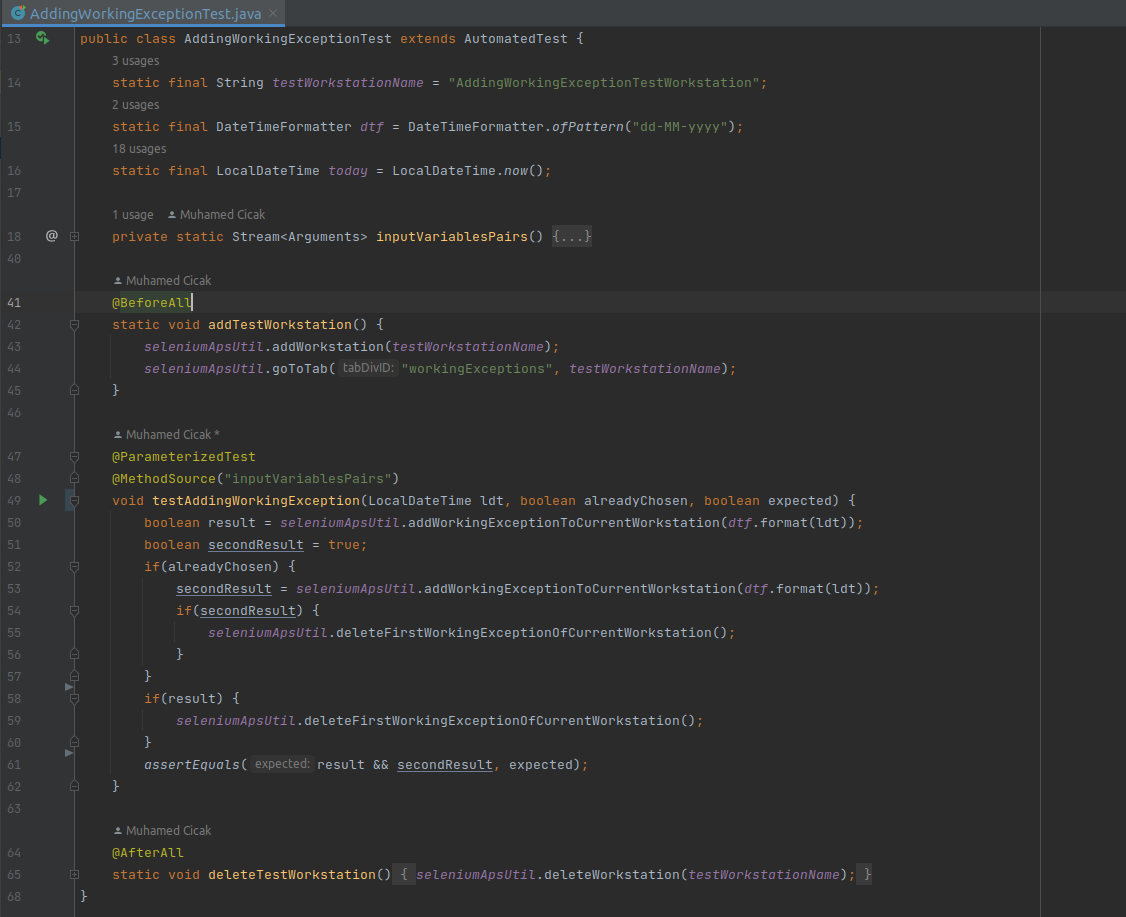
If the add process succeeds, the function returns true, otherwise false. We also delete the added shift if the add process succeeds, so that the next test input can run without a problem, and so that at the end of the test, the test will have left the application in the same state as it was before the test was started.

The aforementioned fromToPairs function is compacted for readability, below is a part of it (it is lengthy).

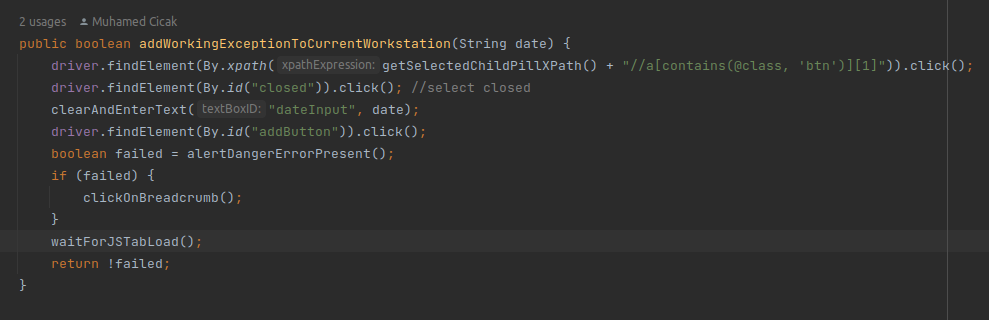


**4.4 Adding a Working Exception Test**

This test has the most challenging logic of the unit testing part, as we have to handle the “already chosen” case by adding an additional working exception. This unit test also has three parts, just like the last two. I will not go over the @BeforeAll and @AfterAll annotations again, they are the same as before. The important part to note in the @BeforeAll is that it navigates to the workingExceptions tab of the added workstation for the unit test. The part that does differ here is the data members of the unit test class. Alongside testWorkstationName we also have DateTimeFormatter dtf and LocalDateTime today member variables here. We will use them later inside the unit test.



The part we are concerned with is the testAddingWorkingException function. Here too, it uses a @MethodSource and like in the example before, the method passed is compacted as it is lengthy. The test function accepts three arguments: LocalDateTime ldt, boolean alreadyChosen and boolean expected. These are extracted from the BVA. The ldt represents the test input date variable, the alreadyChosen boolean whether there already exists a working exception on the given date and boolean expected represents whether this adding operation is expected to succeed or fail. The testfunction starts off by calling seleniumApsUtil.addWorkingExceptionToCurrentWorkstation and passing in dtf.format(ldt), which is the string representation of the passed in LocalDateTime, in the “dd-MM-yyyy" format. The seleniumApsUtil.addWorkingExceptionToCurrentWorkstation function returns true if the adding operation succeeds, and false if it fails. After that, we check whether “alreadyChosen” variable is true, if it is, we must call seleniumApsUtil.addWorkingExceptionToCurrentWorkstation again with the same parameter, emulating a conflict with an existing working exception. We also check if this adding operation succeeded, and if it did, we remove the added working exception. We also do this for the first added working exception to make sure the unit test is deterministic. We finish off by making an assertion for the unit test based on both added working exceptions: if either one of them fails, the unit test fails, otherwise it passes. Below, we examine the seleniumApsUtil.addWorkingExceptionToCurrentWorkstation function.



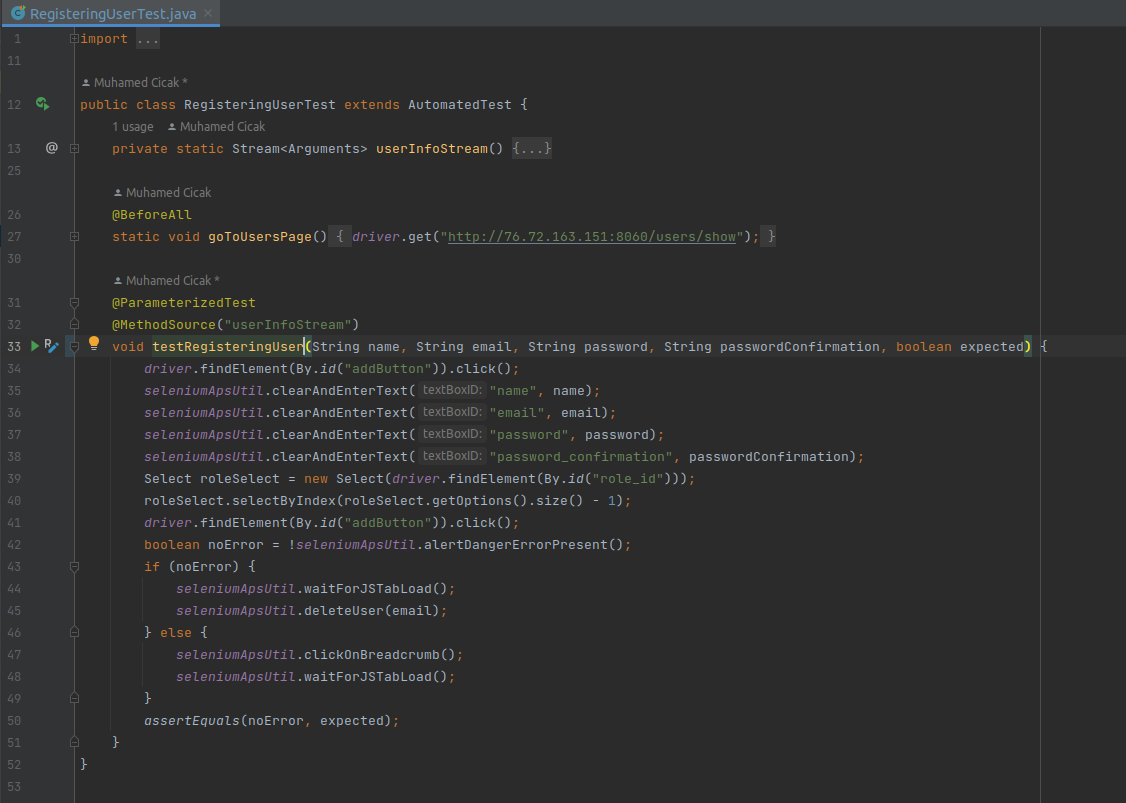
Here we add a working exception and check whether there are any errors. If there are we go back to the working exceptions list via the breadcrumbs. If there are no errors, we will be redirected to the list anyway, but in both cases, we must call waitForJSTabLoad(), this is to ensure the js script of the website switched to our desired tab before we proceed with other actions.

Here is also the expanded view of the inputVariablesPairs function:



**4.5 Registering a User Test**

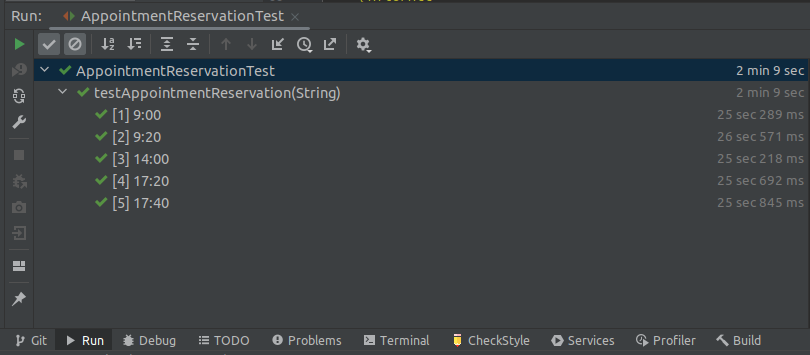
Registering a user is the simplest unit test in every sense. Its test input cases are few and the GUI interaction is quite simple. Also, the test has no preconditions or postconditions, so we have no @BeforeAll and @AfterAll annotations. It does, nevertheless, use @MethodSource as it has 5 arguments to be passed in to.



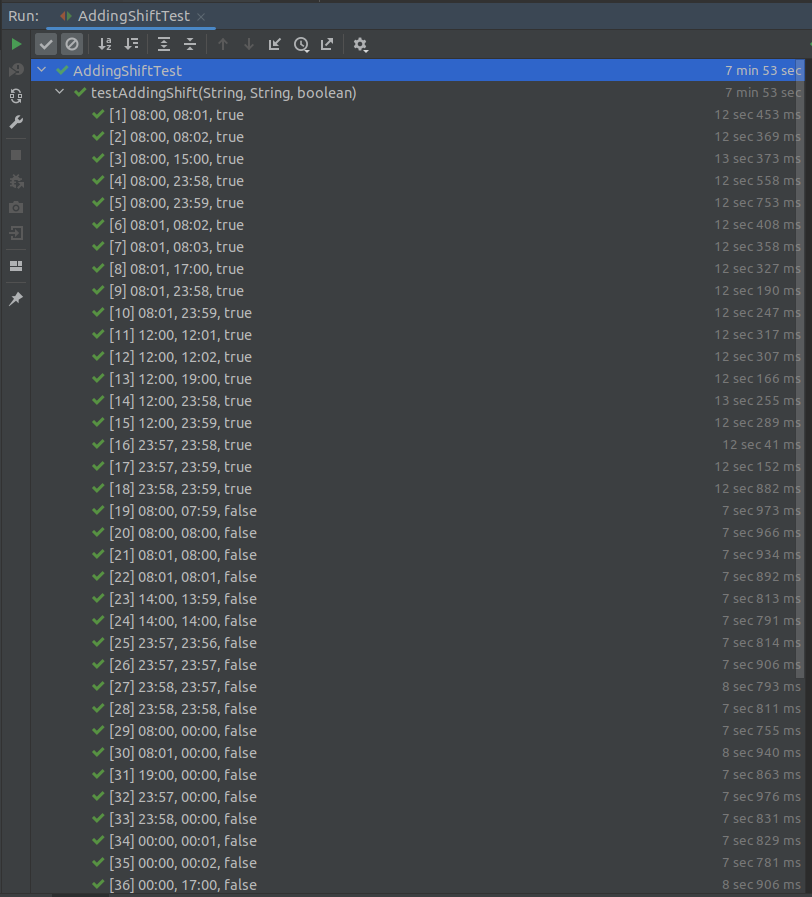
Here, in testRegisteringUser, we fill in the necessary fields and then try to “add” (register) the user. If no error occurs, we wait for the js tab to load after the redirect, and then delete the added user. If an error occurs, we go back to the user list and wait for the js tab to load. Then the noError boolean is compared with the expected boolean with assertEquals, if they are the same, the test passes, otherwise it fails.

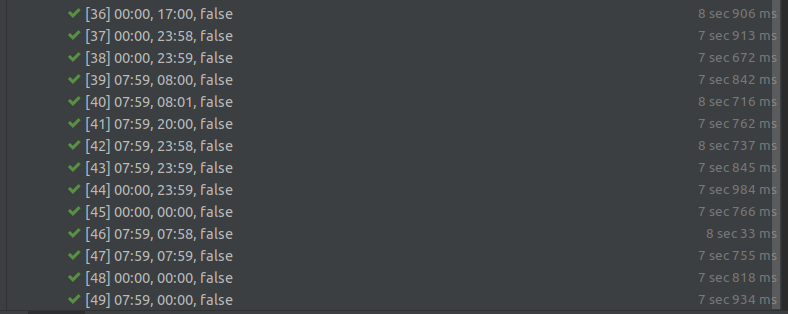
**4.6 Running Tests**

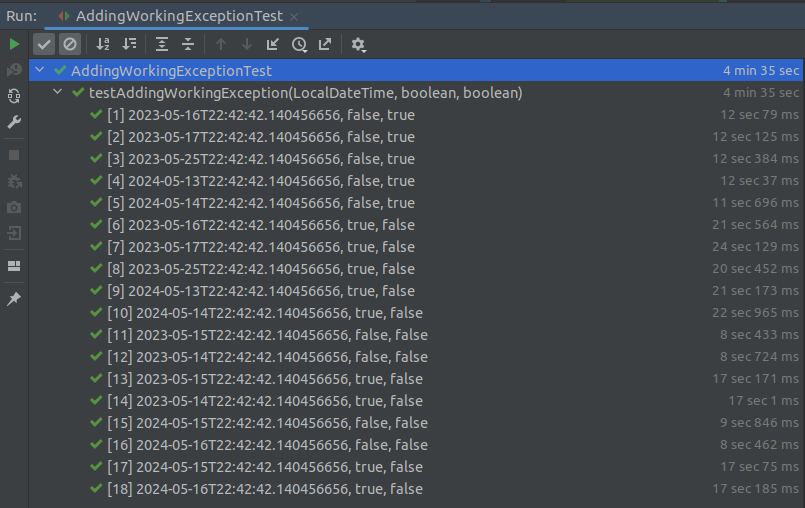
Below we will put all of the unit tests’ results and you can see that all of them pass as expected. The beauty of test automation and the deterministic property of these automated tests is that you can run time as many times, in parallel, without full reliability. This is very useful as part of a continuous integration development, as these tests are suitable to be run after each new build/version of the app.

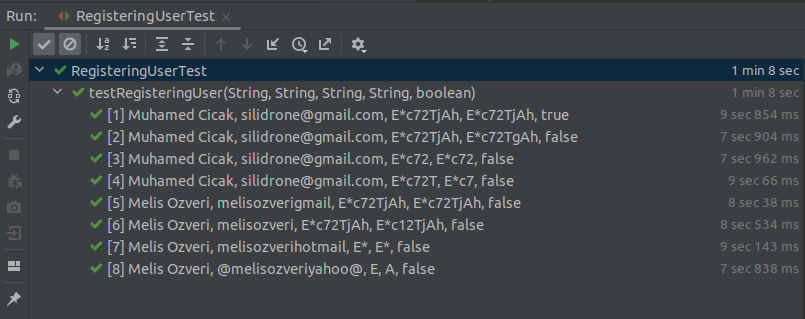


We had to break AddingShiftTest into two images, as it couldn’t fit a single one, the upcoming two images below belong to the AddingShiftTest









**5. CONCLUSION**

This report has achieved its purpose by successfully testing zamaninda.com. Test cases were created with Equivalence Class Partitioning, Boundary Value Analysis and Decision Table methods. Then, test cases were created. On top of this, test cases were automated. Lastly, the results of these testing procedures were displayed.