Lab Manual

CSE-455 Software Testing

Department of Computer Science
Islamabad Campus

Lab Contents:

Test case management in JIRA; Writing test cases in JUNIT, Automated Test cases in Cypress and Selenium

Graduate Attributes (Keep the ones related to your course)

S.#	Description
2	Apply knowledge of computing fundamentals, knowledge of a computing specialization, and mathematics, science, and domain knowledge appropriate for the computing specialization to the abstraction and conceptualization of computing models from defined problems and requirements.
4	Design and evaluate solutions for *complex* computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations.

Intended Learning Outcomes

Sr.#	Description	Blooms Taxonomy Learning Level	SO
CLO -4	Implement a program using basic programming constructs.	*Applying*	2,4

Lab Assessment Policy

Assignments	Lab Mid Term Exam	Lab Terminal Exam	Total
25	25	50	100

Note: Midterm and Final term exams must be computer based.

List of Labs

Lab #	Main Topic	Page #
Lab 01	Software Testing Tools, Errors, Faults and Failures	**4**
Lab 02	Write Test Cases for ATM System	**7**
Lab 03	Bug Tracking tool – JIRA (Issues, Scrum board, Springs, Backlog)	**10**
Lab 04	JIRA Kanban Board, Bug tracking and Reports in JIRA	**25**
Lab 05	JIRA Automation	**29**
Lab 06	Zephyr Scale in JIRA for Test Case Managment	**37**
Lab 07	JUNIT for writing unit tests (Assertion and Annotation methods)	**50**
Lab 08	Test Suits, Test case execution order, Parameterized Test	**73**
Lab 09	**Mid Term Exam**	
Lab 10	Selenium IDE, Commands (Login, search, logo,Page Title etc), Test Suites	**78**
Lab 11	Selenium WebDriver, FindElements and WebElements in WebDriver	**90**
Lab 12	Cypress installation and Basic Commands	**112**
Lab 13	Miscesllaneous features in Cypress, Data Driven Tests	**126**
Lab 14	API Mocking in Cypress, Cypress Studio, Visual Regression Testing, Cypress Dashboard	**137**
Lab 15	**Final Term Exam**	

Lab 01

Software Testing Tools, Errors, Faults and Failures

Objective:

The primary objectives of this lab are to:

- Understand the fundamentals of software testing tools.
- Understand fundamental concepts of Software Testing like Errors, Fault/Bug/Defect, Failure
- Develop the ability to identify and report software defects.
- Understand the importance of software quality assurance.

Activity Outcomes:

On completion of this lab student will be able

- Understanding of Error, Fault and Failure.
- Student will be able to find Bugs in any of the software products.

1) Useful Concepts

Introduction:

According to IEEE Definitions:

Fault: It is a condition that causes the software to fail to perform its required function.
Error : Refers to difference between Actual Output and Expected output.
Failure : It is the inability of a system or component to perform required function according to its specification.

- **Failure:** External behavior is incorrect
- **Fault:** Discrepancy in code that causes a failure.
- **Error:** Human mistake that caused fault

Note:

- **Error** is terminology of Developer.
- **Bug** is terminology of Tester

Some of Software Testing Tools

S#	Tool Name	Basic Description
1	Selenium	Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language

2	JUNIT	It's unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively.
3	Bugzilla	Bugzilla is a defect/bug tracking tool. Defect tracking systems allow developers and testers to track all the outstanding defects. Bugzilla can be linked to other testing tools like JIRA, QC or ALM, etc. Bugzilla is developed in Perl and runs on MYSQL server.
4	JIRA	JIRA is a project management tool used for issues and bugs tracking system. It is widely used as an issue-tracking tool for all types of testing. JIRA can also be used for test management.
5	PyTest	Pytest is a testing framework which allows us to write test codes using python. You can write code to test anything like database , API, even UI if you want. But pytest is mainly being used in industry to write tests for APIs.
6	Ranorex	Windows GUI test and automation framework for C++, Python and for the .Net languages
7	Jmeter	Java desktop application designed to load test functional behavior and measure performance
8	TestComplete	TestComplete is an automated testing tool used for functional testing, it was created by SmartBear Software. TestComplete enables testers to generate tests for Windows, Web, Android, and iOS applications. these tests can be recorded effectively. automated scripts can be composed from Python, C++Script, VBScript, Jscript, or JavaScript languages as well
9	Watir	It is an open-source (BSD) family of Ruby libraries for automating web browsers. It is simple and flexible. It supports multiple browsers on different platforms
10	Katalon Studio	It is a test automation framework such as Selenium and Appium by eliminating their technical complexities to allow testers to efficiently setup, create, run, report and manage their automated tests. It also offers a viable alternative to commercial test automation solutions that are unaffordable to many small and medium-sized teams
11	Soap Test	Automated tool for testing Web services. SOAP test facilitates server functional testing by automatically creating a test suite from a WSDL document that tests every operation associated with that document
12	Appium	Appium offers a totally fresh revolution in automation testing that promises effective, bug-free and high-quality apps, which saves a project time, cost and labor. Appium is an open source, crossplatform mobile testing tool that enables developers to write tests on various platforms such as iOS and android.
13	Quick Test Professional (QTP)	QTP (Quick Test Professional) a Windows based programming testing tool used to test applications on desktop or the web, it offers testing automation for regression and functional testing, developed by Hewlett Packard (HP).
14	MochaJS	MochaJS is the most popular testing framework that supports backend and frontend testing. MochaJS Support NodeJS debugger

15	Jasmine	Jasmine is a user-behavior mimicker that allows you to perform test cases similar to user behavior on your website. Jasmine is useful for a testing frontend for visibility, click clarity as well as the responsiveness of the UI in different resolutions. Jasmine allows to automate user behavior with customs delays and wait time to simulate actual user behavior.
16	JEST	JEST is one of the most popular frameworks that is maintained regularly by Facebook. It is a preferred framework for the React based applications as it requires zero configuration.
15	Karma	Karma is a productive testing environment that supports all the popular test description framework within itself. It provides your application the support to execute tests in different environments. It has wide support for executing tests on different devices and applications.

2) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	***10***	***Low***	***CLO-4***
2	***20***	***Medium***	***CLO-4***

Activity 1:

The example demonstrates the difference in Error, Fault and Failure.

```
pre: param is an integer.
post: returns the product of the param multiplied by 2.

1. int double (int param) {
2.   int result;
3.   result = param * param;
4.   return result;
5. }
```

Identification of Error, Fault and Failures

. A call to double(3) returns 9, but the post condition says it should return 6.

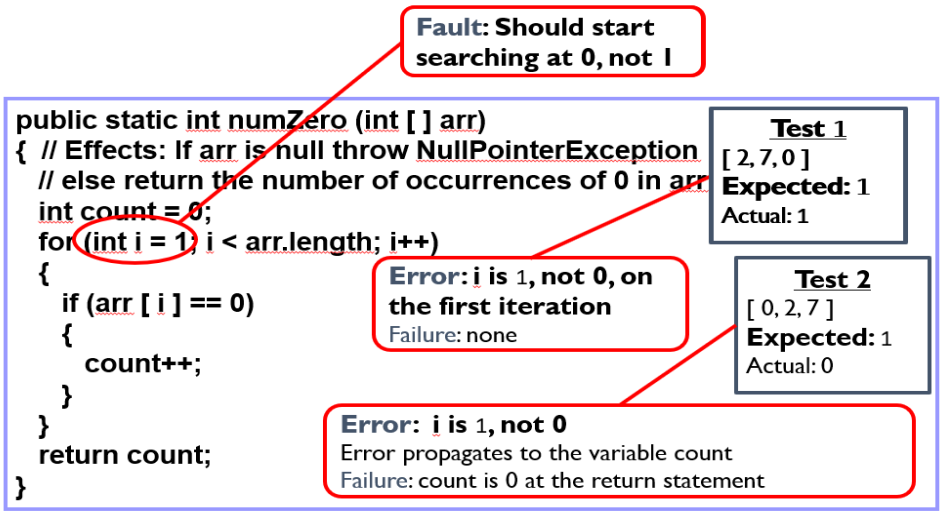
• Result 9 represents a **failure**.

• The failure is due to the **fault** at line 3, ("* param" is used instead of "* 2")

• The **error** is a typo, (someone typed "* param" instead of "* 2" by mistake).

Activity 2:

The example demonstrates the difference in Error, Fault and Failure.



3) Graded Lab Tasks

Lab Task 1

Evaluate software by observing it's execution. Write test cases by introducing some errors, faults and failures to explain your answer.

Consider the concept of a CourseResult. The CourseResult should have data members like the student name, course name and grade obtained in that course. This concept can be represented in a class as follows:

Public class CourseResult

{

Public String studentname;

Public String coursename;

Public String grade;

public void display()

{

System.out.println("Student Name is: — + studentname + "Course Name is: — + coursename + "Grade is: — + grade); }

}

Public class CourseResultRun

```
 {
public static void main(String[]args)
 {
CourseResult c1=new CourseResult (); c1.studentName= ―Ali‖; c1.courseName= ―OOP‖; c1.grade= ―A‖;
 c1.display();
 CourseResult c2=new CourseResult ();
 c2.studentName= ―Saba‖;
c2.courseName= ―ICP‖;
c2.grade= ―A+‖;
c2.display();
}  }
```

Lab Task 2

Design a class Point with two data members x-cord and y-cord. This class should have an arguments constructor, setters, getters and a display function. Now create another class ―Line‖, which contains two Points ―startPoint‖ and ―endPoint‖. It should have a function that finds the length of the line. Hint: formula is: sqrt((x2-x1) 2 + (y2-y1) 2) Create two line objects in runner and display the length of each line.

Write test cases to check the above functionality by introducing different errors, faults and failure Mention 3 test cases for failures and 3 test cases for Non failures as shown in above example.

Lab Task 3

Consider CUOnline system Identify, document, and report defects found during testing

Lab 02

Write Test cases for ATM System

Objective:

The purpose of this lab is to create an understanding of how to write test cases.

Activity Outcomes:

Student will be able to write test cases for any given system

1) Useful Concepts

Software Testing

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product:

• Meets the business and technical requirements that guided its design and development

• Works as expected

• Can be implemented with the same characteristic.

Let's break the definition of Software testing into the following parts:

- **Process**: Testing is a process rather than a single activity.
- **All Life Cycle Activities:** Testing is a process that's take place throughout the Software Development Life Cycle (SDLC). The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it's referred as "verifying the test basis via the test design".
- The test basis includes documents such as the requirements and design specifications.
- **Static Testing:** It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost-effective way of testing. For example: reviewing, walkthrough, inspection, etc.
- **Dynamic Testing:** In dynamic testing the software code is executed to demonstrate the result of running tests. It's done during validation process. For example: unit testing, integration testing, system testing, etc.
- **Planning:** We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.
- **Preparation:** We need to choose what testing we will do, by selecting test conditions and designing test cases.
- **Evaluation:** During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.
- **Software products and related work products:** Along with the testing of code the testing of requirement and design specifications and the related documents like operation, user and training material is equally important

Test Case Template:

Test Case ID	A Unique ID
Test Case Name	A Valid Name
Test Data	Required to perform test e.g. Password
Pre-Condition	A condition which is required to fulfill the task
Actions	**System Response**
Performed by the user	System response against each step
Expected Result	Result required
Actual Result	Resultant Output
Test Case Status	Pass/Fail
Testing Environment: e.g. Android platform 7.0 & 4.4.2 s	
Tested by: XYZ	
Date: 14th, May 2024	

1) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*10*	*Low*	*CLO-4*

Activity 1:

Write Test Cases for the ATM system (FR)

TC- 02: Sign in with invalid credentials

Test Case ID	A 1.2
Test Case Name	Enter Invalid Pin
Test Data	1111
Pre-Condition	• • User has inserted the credit card in the system.
Actions	**System Response**
1. User enter 4-digit pin number using keypad. 2. User presses enter	1. Screen displays input 2. System show message “Validating”
Expected Result	System will show: Invalid PIN
Actual Result	Application shows error message Invalid PIN
Test Case Status	Pass

Testing Environment: Linux

Tested by: XYZ

Date: May 2, 2024

2) Graded Lab Tasks

Lab Task 1:

Write all test cases for CUOnline System.

Lab 03

JIRA (SCRUM Board-Sprints)

Objective:

The purpose of this lab is to understand what is the use of JIRA tool. The focus of this lab is to familiarize students with JIRA's bug tracking capabilities, enabling them to effectively report, manage, and track bugs within a software testing project.

Activity Outcomes:

- Understanding of JIRA as bug tracking tool
- Student will be able to track bugs in any of the software products.

Instructor Note:

1) Useful Concepts

Introduction:

JIRA is a planning, tracking and managing tool built to cater to projects following the agile approach.

It allows:

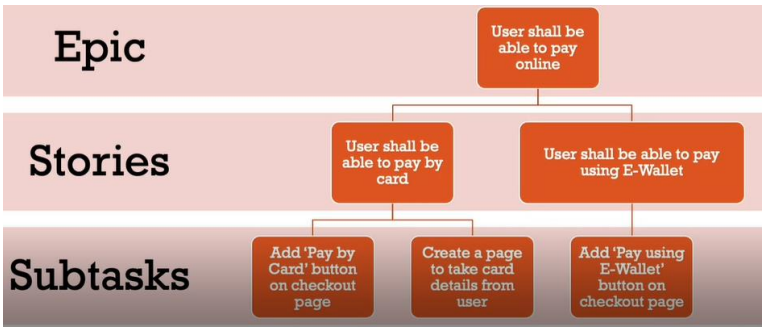
- creation of backlog
- creating sprints
- creating tasks
- updating status
- code integration

Epics, Stories and Child Issues

When it comes to dividing the project in parts, Jira has a concept which comprises of epics, stories and child issues (sub tasks).

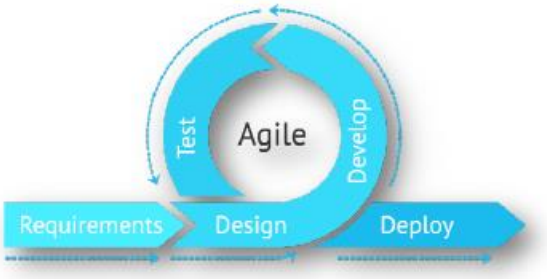
- **Epic –** An abstract idea of some goal to achieve.

- **Story** - Dividing epic into sub components for ease and better structuring.
- **Child Issue** – Lowest level task.



Agile process methodology:

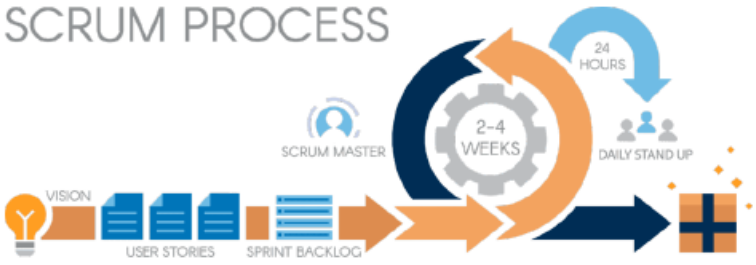
The Agile process methodology is a time-boxed, iterative approach to software development, to deliver the product incrementally instead of all at once. One of its core values being collaboration and constant communication, whether it is between developers or between developers and customers.



Scrum Framework:

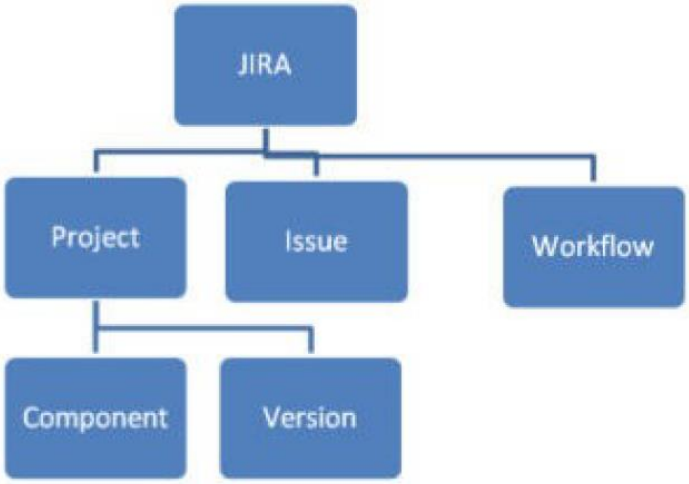
Scrum is a framework adopting the agile process, it's main components include the

- Sprint
- Sprint backlog
- Daily Scrum
- Sprint perspective
- Scrum roles, product owner, scrum master.



Basics about JIRA:

JIRA in its entirety is based on 3 concepts.



1. **Issue:** Every task, bug, enhancement request; basically anything to be created and tracked via JIRA is considered an Issue.
2. **Project:** a collection of issues
3. **Workflow:** A workflow is simply the series of steps an issue goes through starting from creation to completion. Say the issue first gets created, goes to being worked on and when complete gets closed. The work flow in this case is:

Say the issue first gets created, goes to being worked on and when complete gets closed. The work flow in this case is:

JIRA Sprints:

A sprint — also known as an iteration — is a short period in which the development team implements and delivers a discrete and potentially shippable application increment, e.g. a working milestone version. It is recommended using a fixed two-week duration for each sprint for beginners. It's long enough to get something accomplished, but not so long that the team isn't getting regular feedback. Sprints do not apply to Kanban projects.

- Sprints only apply to Scrum boards.
- We must have ranking enabled on your board to use sprints. See Enabling ranking.
- In general, sprint actions require the Manage Sprints permission

Creating a sprint

You can create a sprint for your current iteration, or multiple future sprints if you want to plan several iterations ahead.

1. Go to the Backlog of your Scrum project.
2. Click Create sprint at the top of the backlog.

Once you have created a sprint, you can add issues to it.

Adding issues to a sprint

Three ways to add an issue to a sprint:

-Add existing issues to a sprint

Use this when the issue already exists, and you want to add the issue to an active or future sprint.

-In the Backlog, drag and drop the issues onto the relevant sprint.

-If you want to add multiple issues you can:

- Select the issues (use Shift+Click or Ctrl+Click), right-click, then select the relevant sprint
- Drag the sprint footer down to include issues from the backlog.

Create then add an issue to an active sprint

Use this when the issue doesn't exist, and you want to quickly add it to an active sprint.

After creating an issue in the Active sprints, click the Add to <sprint name> link in the confirmation dialog that displays. <Sprint name> will be the name of the sprint that you are currently viewing on the board.

Note, if you do not have the 'Edit Issues' and 'Schedule Issues' permissions for all projects included by the board's filter, the issue will be added to the backlog instead of the sprint.

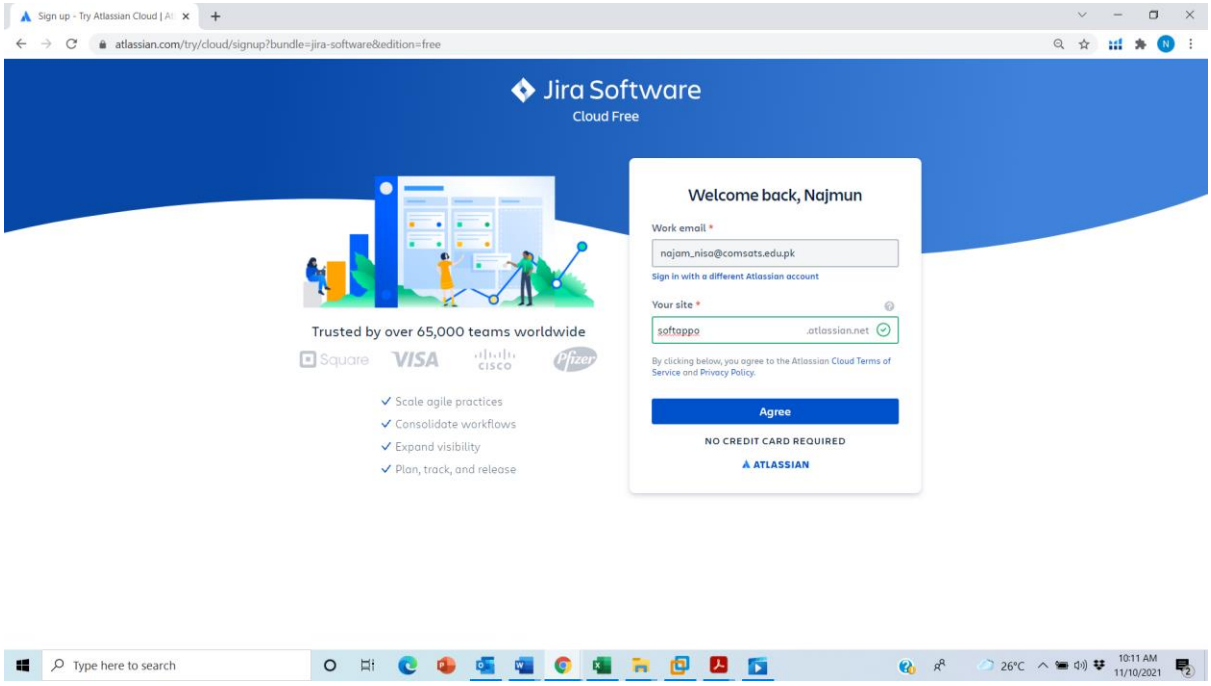
Edit the Sprint field for an issue
Use this when editing an issue, and you know the name of the sprint (active and future sprints only).
Create or edit an issue and enter the sprint name in the Sprint field. If the sprint field doesn't display on the Create Issue or Edit Issue dialog, choose Configure fields, then select the Sprint field.

2) Solved Lab Activites

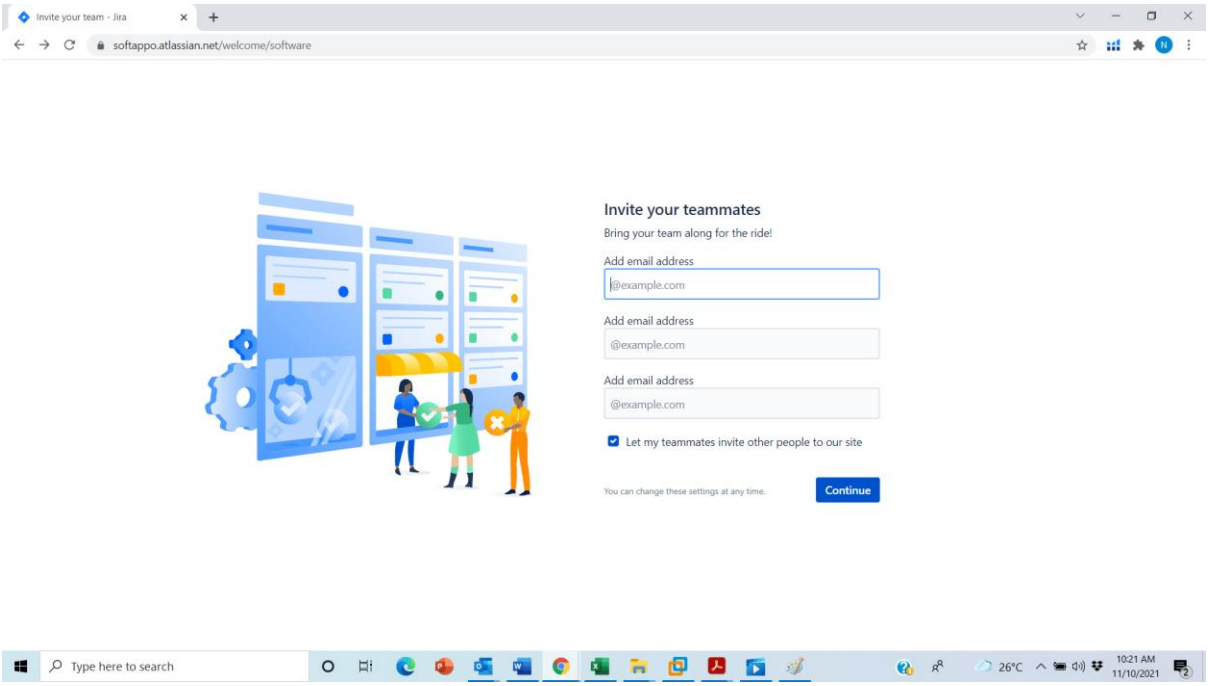
Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*10*	*Low*	*CLO-4*
2	*10*	*Low*	*CLO-4*
3	*15*	*Medium*	*CLO-4*
4	*15*	*Medium*	*CLO-4*

Setting up JIRA

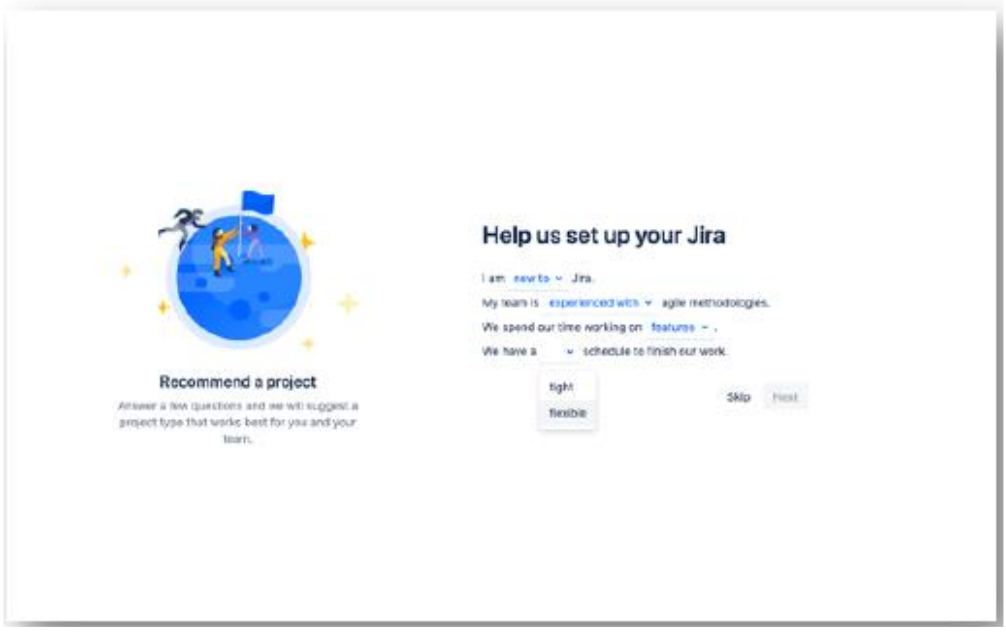
You start off by entering your work email and creating your site name. This site name can be the name of your organization (e.g., your software house) or your project name.



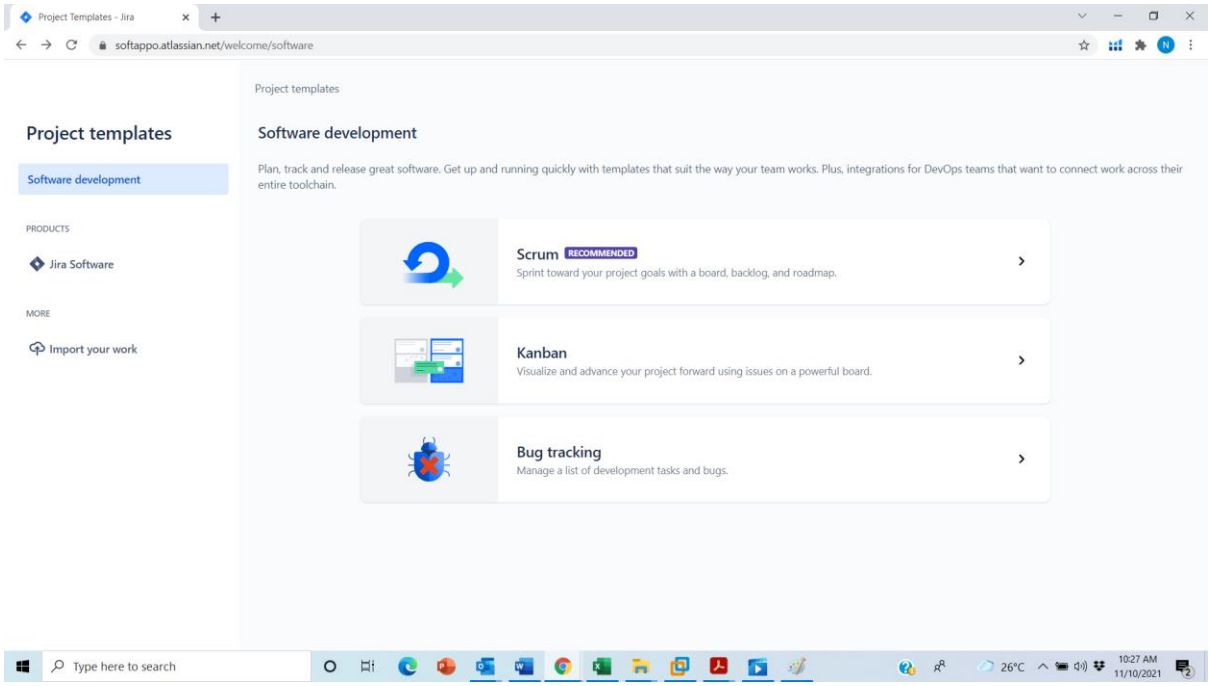
Then you can invite up to 3 members to make your team(You can also add team members later) If you don't want to do this at this stage, you have the option of doing this later after your project has been set up.



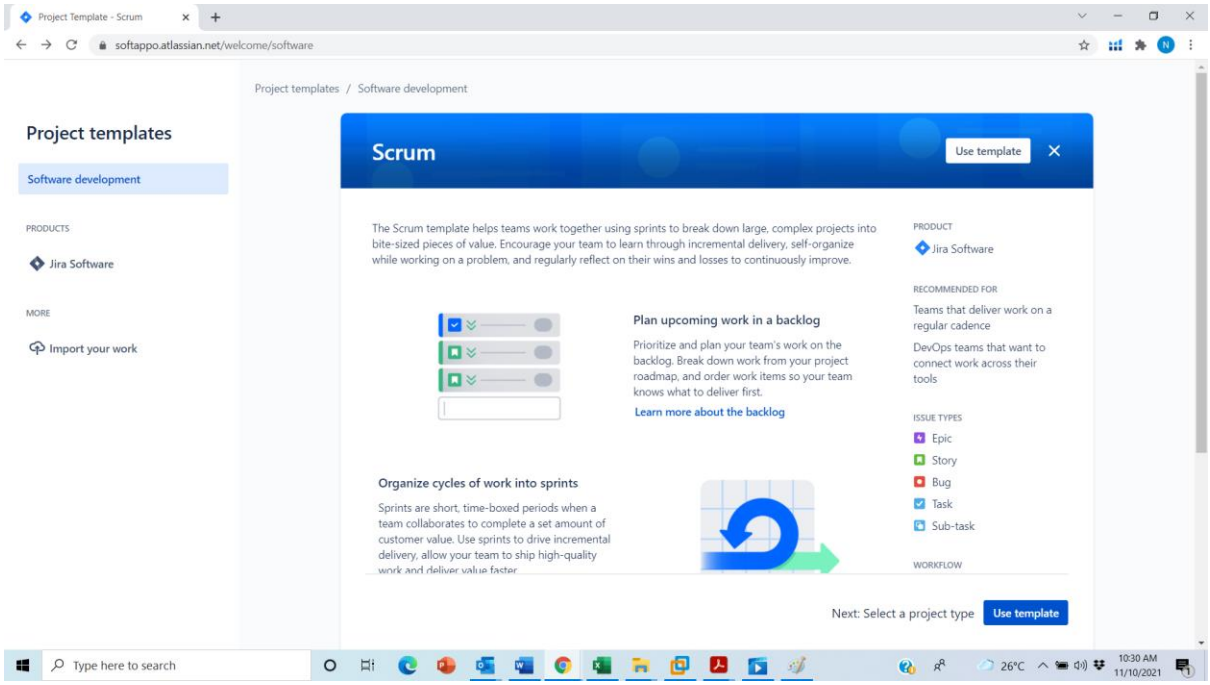
You are then asked a few questions so they can suggest a project type that works best for you and your team.



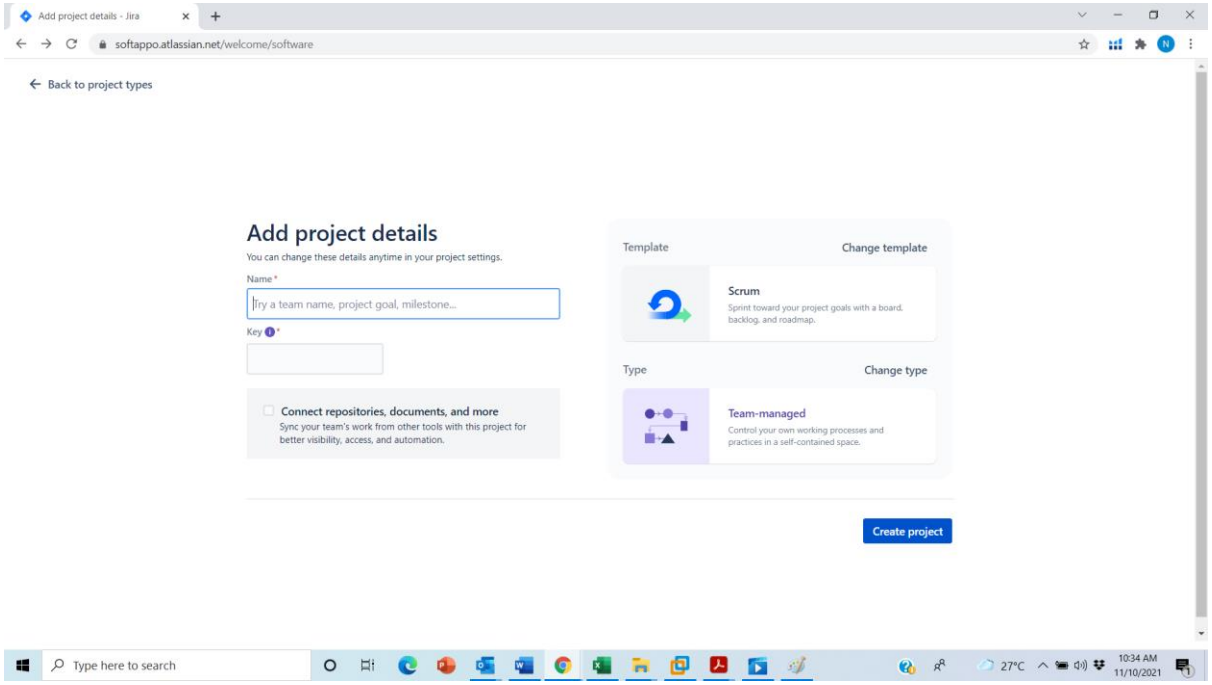
Next, some templates are suggested.



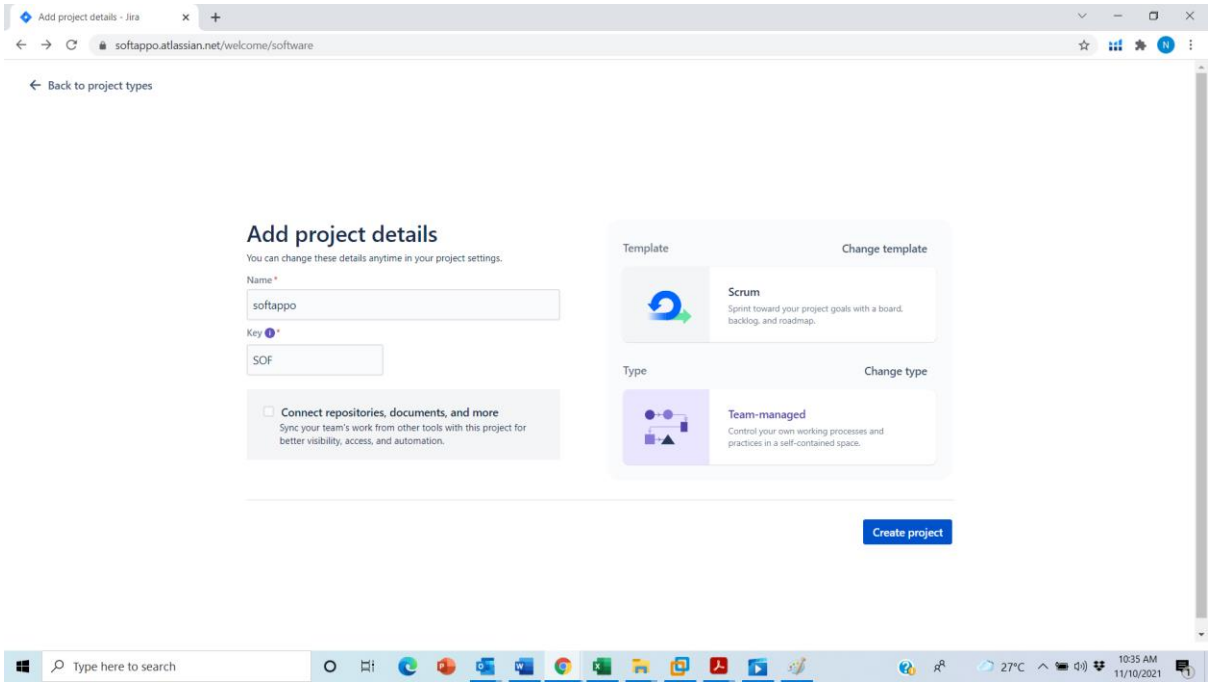
We will use scrum template, the Scrum template helps teams work together using sprints to break down large, complex projects into bite-sized pieces of value



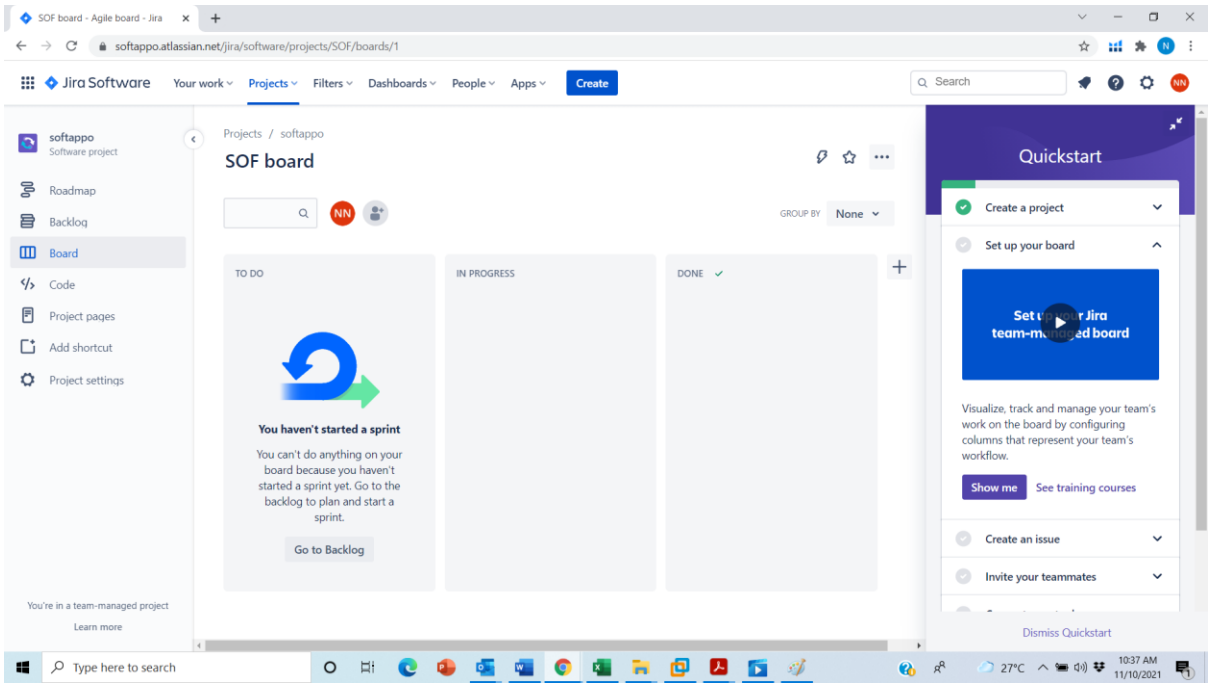
Lastly, you are asked for the project name with an option to change the chosen template if you wish to.



Add project details



Srum board will appear where we can manage your project as shown below:



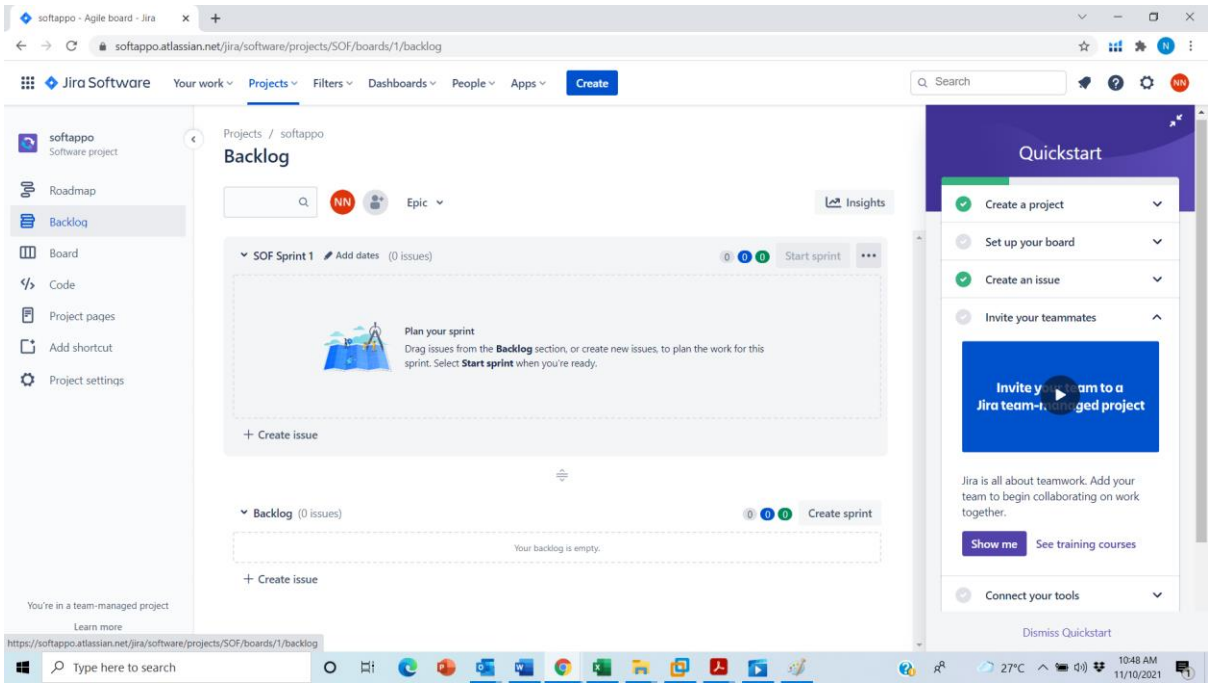
Lab Activity 1:

Backlog

Backlog is where you create epics and stories. This is where you create all the abstract level tasks and divide them into stories and further divide those stories into child issues. This collection of tasks can then be added to sprint showing how many tasks the sprint will cover.

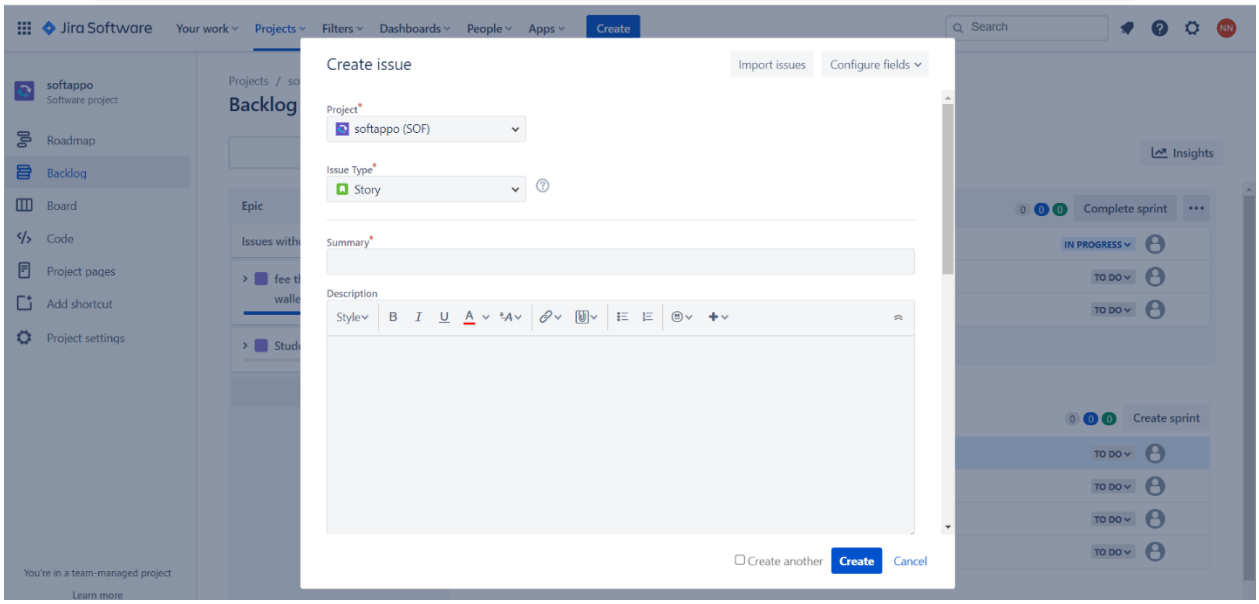
Epic

Here, I've created an epic in the epic panel called 'Fee payment through credit card or cash'. This is an abstract idea or an abstract goal that my team wishes to achieve. So if u haven't started your sprint u can't do any thing on your board for this go on the backlog and start a sprint.



Lab Activity 2:
Creating an issue:

You can also create issues and can link it with other issues from the Create button on the top.



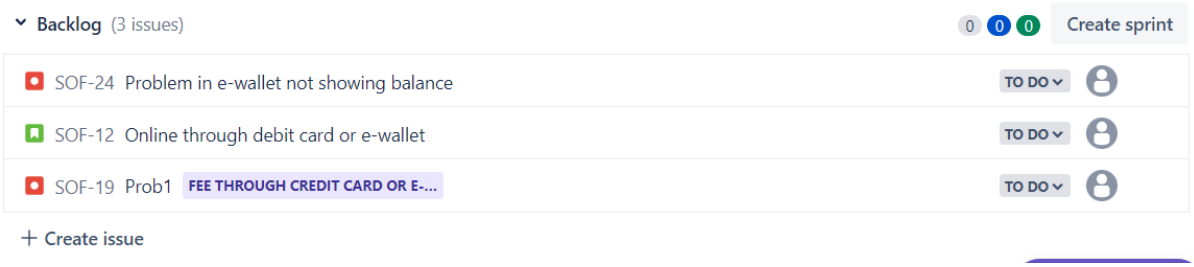
Child Issues:

We can also create child issues in the created issues.

Lab Activity 3:

Sprint:

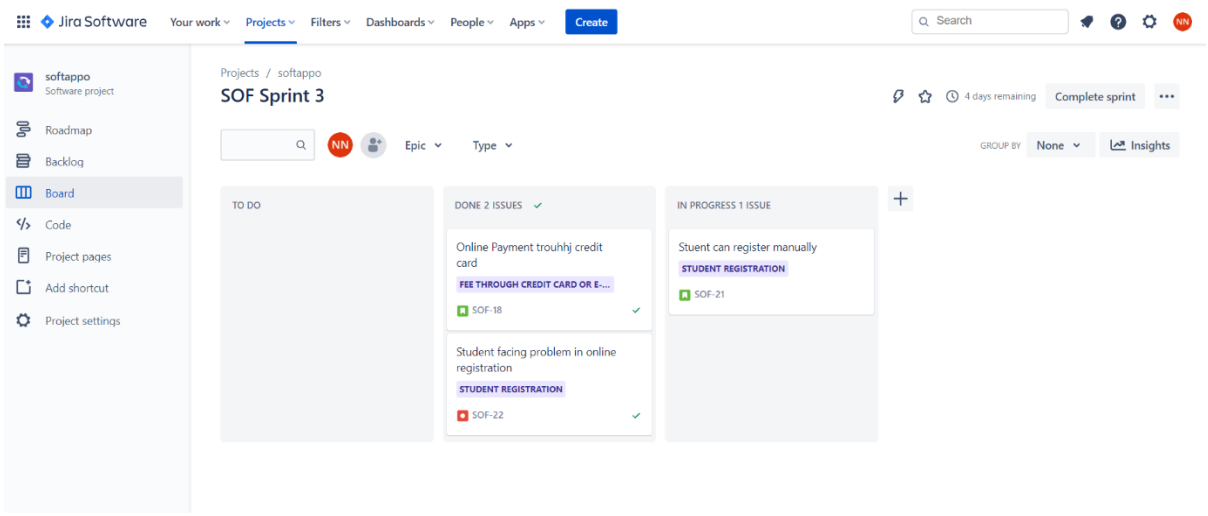
To start a sprint, we will add tasks to it from the backlog. For the backlog given in the screenshot below, I'll add some tasks to a sprint and then start it.



Adding tasks or issues to sprint and start spring.

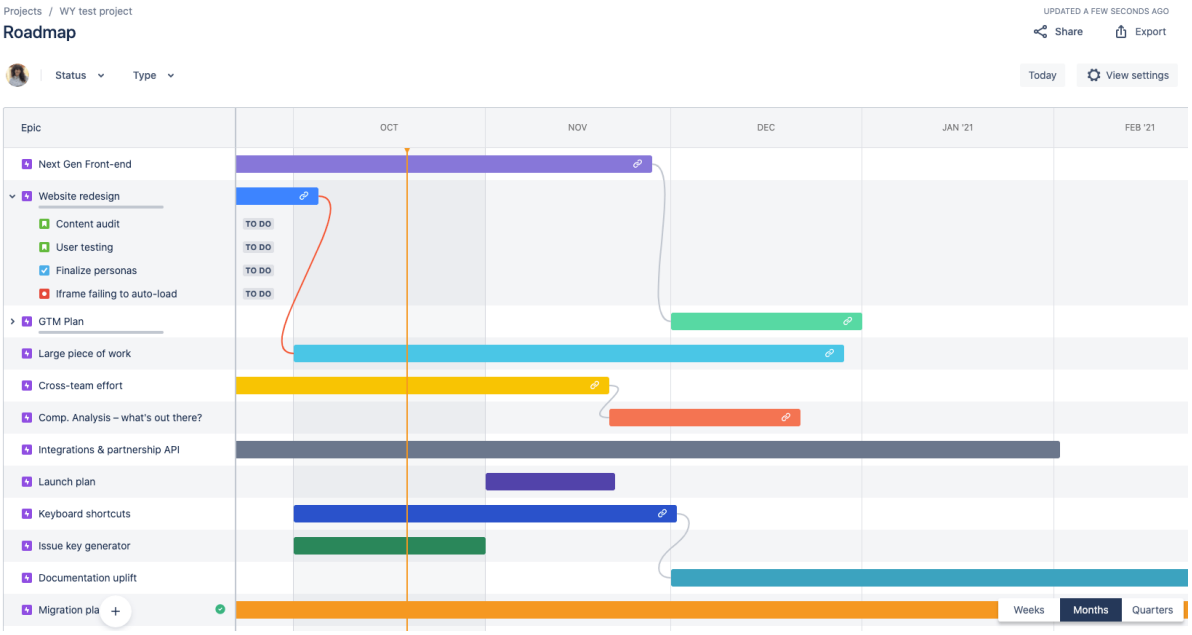
Board

The Board is where you see your ongoing sprint progress. It is empty when there is no ongoing sprint. By default, it has the 'To Do', 'In Progress' and 'Done' columns but more columns can be added if one wishes to.



Lab Activity 4:
Road map:

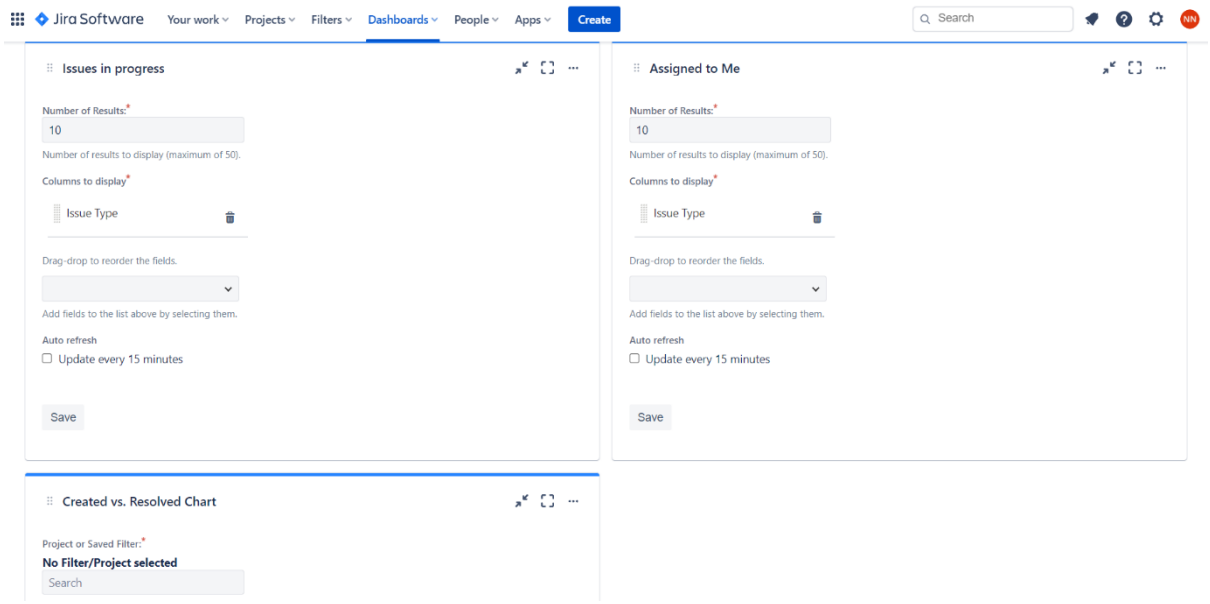
Roadmaps in Jira Software are team-level roadmaps useful for planning large pieces of work several months in advance at the Epic level within a single project. Simple planning and dependency management features help your teams visualize and manage work better



together.

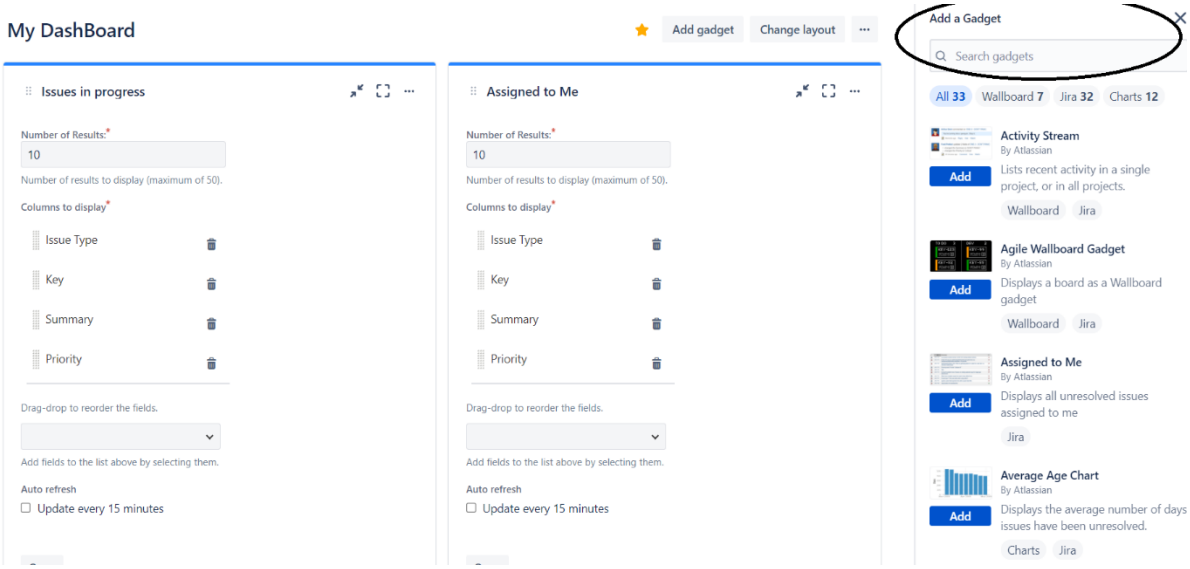
Dashboard
Your dashboard is the main display you see when you log in to Jira. You can create multiple dashboards from different projects, or multiple dashboards for one massive overview of all the work you're involved with.

You can create a personal dashboard and add gadgets to keep track of assignments and issues you're working on. Dashboards are designed to display gadgets that help you organize your projects, assignments, and achievements in different charts.

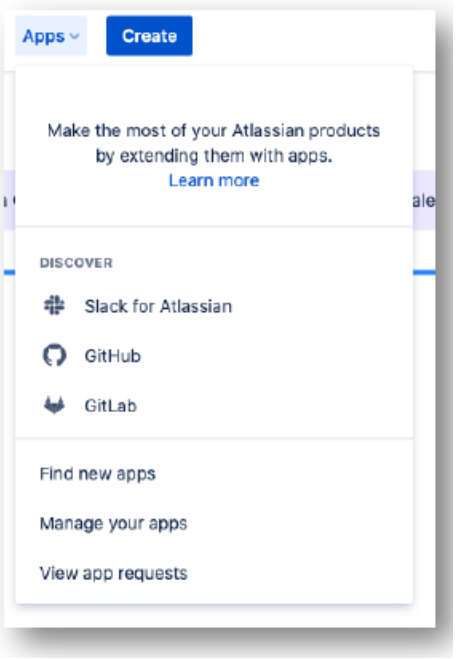


Add Gadgets:

You can add different gadgets to it.



Third Party Software Integration
Many third-party apps can be integrated within your Jira project.



GitHub

One important app integration is GitHub. You can link your GitHub with your Jira project like this.

3) Graded Lab Tasks

Lab Task 1:

Create Bug Reports:

- Login to JIRA.
- Navigate to your project and create a new bug report.
- Fill in details such as summary, description, steps to reproduce, and expected vs. actual results.

Lab Task 2:

Assign and Prioritize Bugs:

- Assign the bug to a team member.
- Set the priority of the bug (e.g., blocker, critical, major, minor).

Lab Task 3:

Track Bug Lifecycle:

- Transition the bug through different statuses (e.g., Open, In Progress, Resolved, Closed).
- Add comments and attachments (e.g., screenshots, logs) to the bug report.

Lab Task 4:

Use Filters and Searches:

- Create filters to find bugs based on different criteria (e.g., status, assignee, priority).
- Save and share filters with your team.

Lab 04

JIRA Kanban Board and Bug Tracking

Objective:

To introduce students to the Kanban methodology using JIRA, enabling them to visualize workflow, manage work in progress, and improve efficiency and finally how to track bugs in JIRA

Activity Outcomes:

- Understanding of Kanban Board in JIRA.
- SCRUM vs KANBAN
- Student will be able to track issues in any of the software products
- Creating detailed bug reports.
- Categorizing and prioritizing bugs.
- Assigning bugs to team members.
- Tracking the lifecycle of a bug from discovery to resolution.
-

1) Useful Concepts

Kanban Methodology: Understanding the key principles of Kanban: visualizing work, limiting work in progress (WIP), and enhancing flow.

Kanban Board: The structure and components of a Kanban board (columns, cards, WIP limits).

Workflow Stages: Typical stages in a Kanban workflow (e.g., To Do, In Progress, Done).

WIP Limits: Setting and managing WIP limits to avoid overloading team members.

Introduction:

Jira is an issue tracker that can be extended into a bug tracker, an incredibly powerful service desk and/or a feature-rich project management tool for agile teams. JIRA is a planning, tracking and managing tool built to cater to projects following the agile approach.
JIRA has three types of boards:

1. SCRUM Template
2. KANBAN Template
3. Bug Tracking

We have already discussed SCRUM in which a scrum board has a set number of tasks and strict deadline to complete them.
A scrum board has a set number of tasks and strict deadline to complete them.

KANBAN Template:

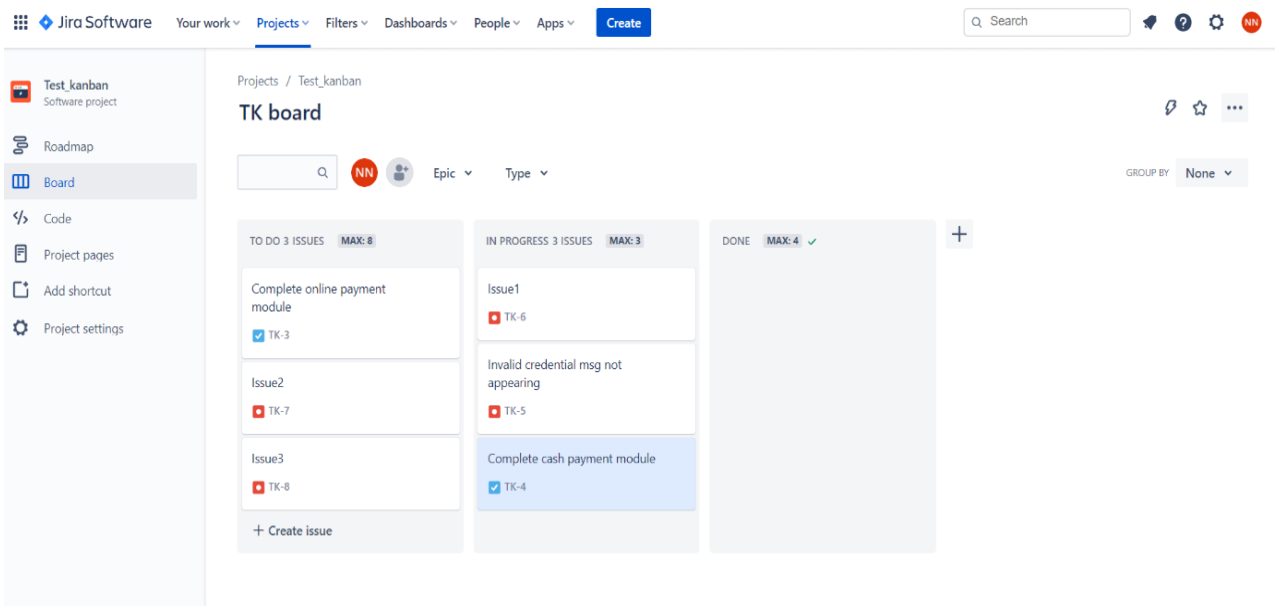
Kanban is a Japanese word meaning visual signal. Kanban boards are a way of visualizing work (in progress and upcoming) to maximize efficiency and the collective workflow. Teams working in a Kanban management framework focus on reducing the time it takes a project—or a user story within a project—to move from start to finish.

Kanban boards use a dynamic assembly of cards and columns to chart and promote continuous improvement in workflow. It helps ensure technology and service teams commit the right amount of work time to hit visible delivery points promised to the client or sponsor of the project.

Difference between KANBAN and SCRUM:

Kanban is a project management method that helps visualize tasks, while Scrum is a method that provides structure to the team and schedule.

This board will provide cards that can easily check the working model of the system



Bug Tracking and Management:

Useful Concepts:

Bug Reporting: Key elements of a detailed bug report (summary, description, steps to reproduce, expected vs. actual results).

Issue Types: Understanding the different issue types in JIRA (bugs, tasks, stories).

Bug Lifecycle: The typical workflow stages for a bug (e.g., Open, In Progress, Resolved, Closed).

Prioritization: Setting and managing priorities for bugs (e.g., blocker, critical, major, minor)

2) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*10*	*Low*	*CLO-4*
2	*10*	*Low*	*CLO-4*
3	*10*	*Medium*	*CLO-4*

Lab Activity 1:

Create a Kanban Board:

- Login to JIRA.
- Navigate to the project and create a new Kanban board.
- Configure the board with appropriate columns (e.g., Backlog, To Do, In Progress, Done).

Lab Activity 2:

Add Tasks to the Board:

- Create tasks (issues) and add them to the backlog.
- Move tasks from the backlog to the To Do column to start work.

Lab Activity 3 :

Manage Work in Progress:

- Set WIP limits for the In Progress column.
- Move tasks through the workflow stages (To Do, In Progress, Done).

Track and Optimize Workflow:

- Use the Kanban board to track task progress.
- Identify bottlenecks and areas for improvement.
- Adjust WIP limits and workflow stages as needed.

3) Graded Lab Tasks

Lab Task 1:

Create Bug Reports:

- Create a new bug report in JIRA.
- Fill in details such as summary, description, steps to reproduce, and expected vs. actual results.

Lab Task 2:

Categorize and Prioritize Bugs:

- Assign the bug to a team member.
- Set the priority of the bug.
- Add labels and components to categorize the bug.

Lab Task 3:

Track Bug Lifecycle:

- Transition the bug through different statuses (Open, In Progress, Resolved, Closed).
- Add comments, attachments (e.g., screenshots, logs), and update fields as needed.

Lab Task 4:

Use Filters and Searches:

- Create filters to find bugs based on different criteria (e.g., status, assignee, priority).
- Save and share filters with the team.

Lab 05

JIRA Automation

Objective:

The objective of this is to guide students through the process of automating tasks in JIRA, a popular project management and issue tracking tool. By the end of this manual, users will be able to:

- Understand basic concepts of JIRA automation.
- Create and configure automation rules.
- Implement common automation scenarios.
- Utilize JIRA's Automation features to improve project management efficiency.

Activity Outcomes:

By completing the activities in this manual, users will:

- Gain familiarity with JIRA's automation capabilities.
- Learn to create automation rules that can trigger actions based on specific events.
- Understand how to use JIRA's built-in templates and customize automation rules.
- Develop skills to troubleshoot and optimize automation workflows.

1) Useful Concepts

1. Automation Rules

Triggers: Events that start the execution of an automation rule. Examples include issue creation, status changes, or scheduled intervals.

Conditions: Criteria that must be met for the rule to proceed. These can include issue fields, JQL queries, or logical conditions.

Actions: Operations performed when a rule is triggered and conditions are met. Examples include creating issues, sending notifications, or editing issue fields.

2. JQL (JIRA Query Language)

JQL: A powerful querying language in JIRA used to filter and sort issues based on specific criteria.

Basic JQL Syntax: Examples include `project = "Project Name" AND status = "Open"` or `assignee = currentUser() AND priority = "High"`.

3. Components of Automation

Templates: Pre-defined automation rules that can be quickly applied and customized.

Custom Rules: User-defined rules tailored to specific project needs.

Logs and Audit: Tools to track the execution and performance of automation rules.

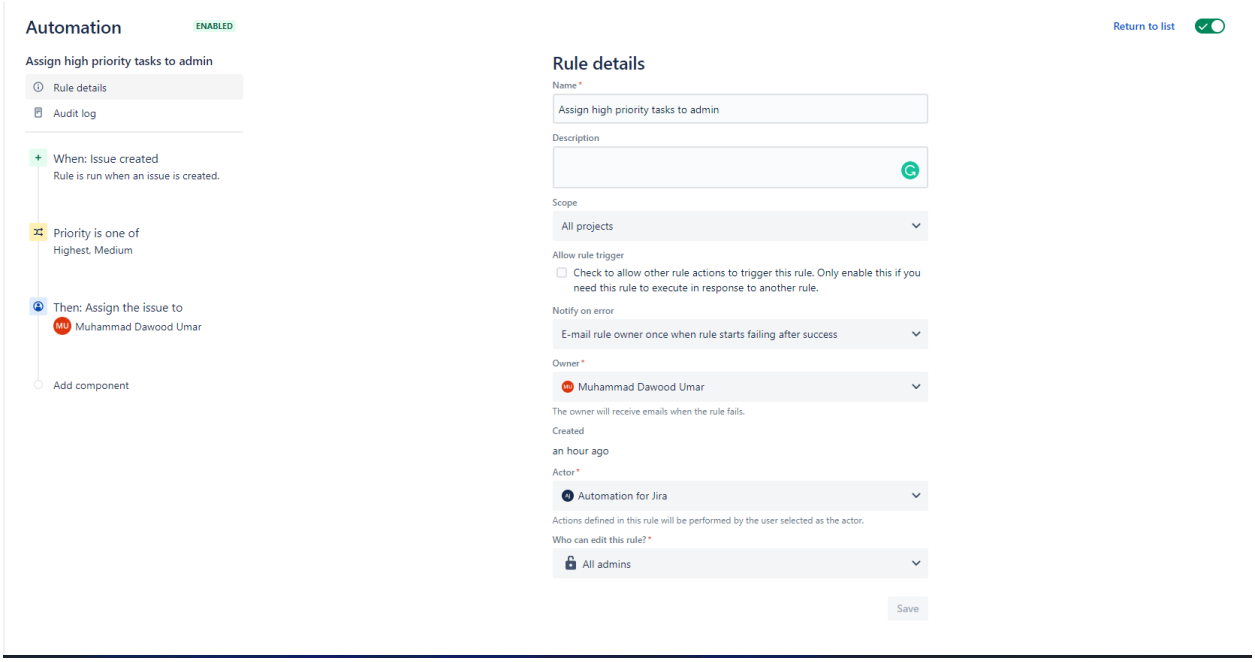
2) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*5*	*Low*	*CLO-4*
2	*5*	*Low*	*CLO-4*
3	*5*	*Low*	*CLO-4*
4	*5*	*Low*	*CLO-4*
5	*10*	*Medium*	*CLO-4*
6	*10*	*Medium*	*CLO-4*
7	*10*	*Medium*	*CLO-4*

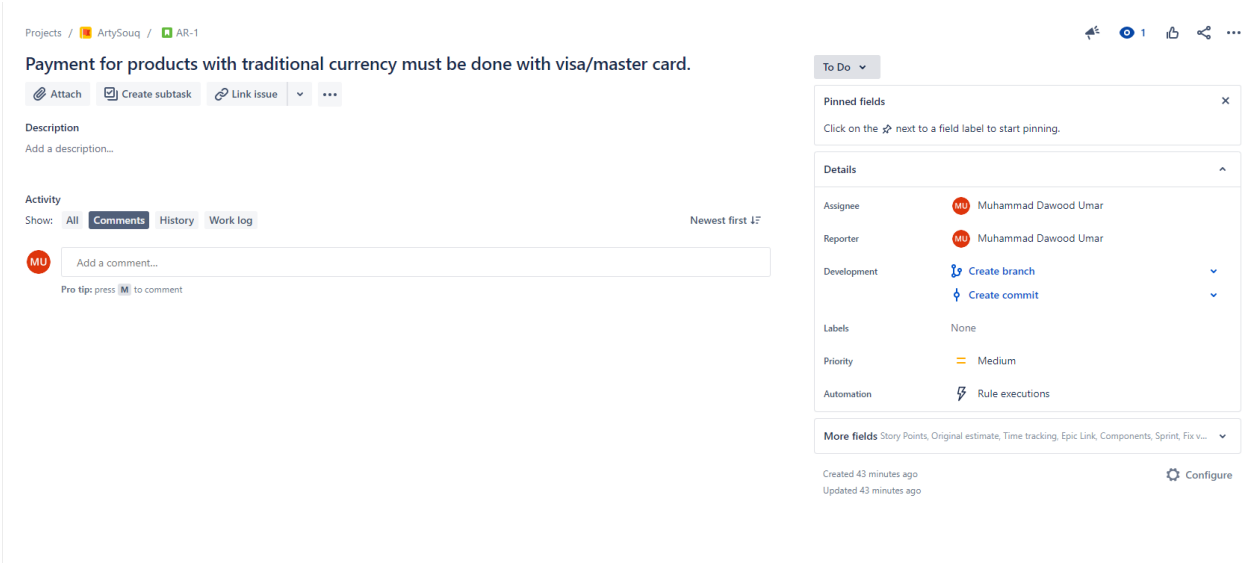
Lab Activity 1:

Rule 1:

When Issue is created with high or medium priority, issue is assigned to user Muhammad Hasnain.



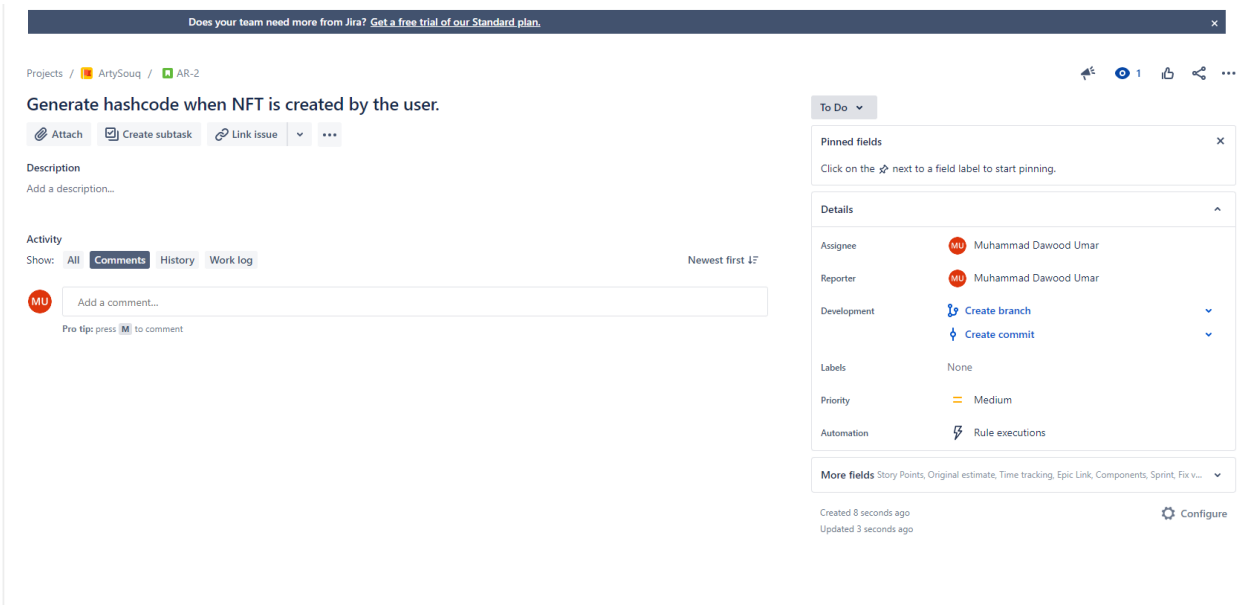
Action



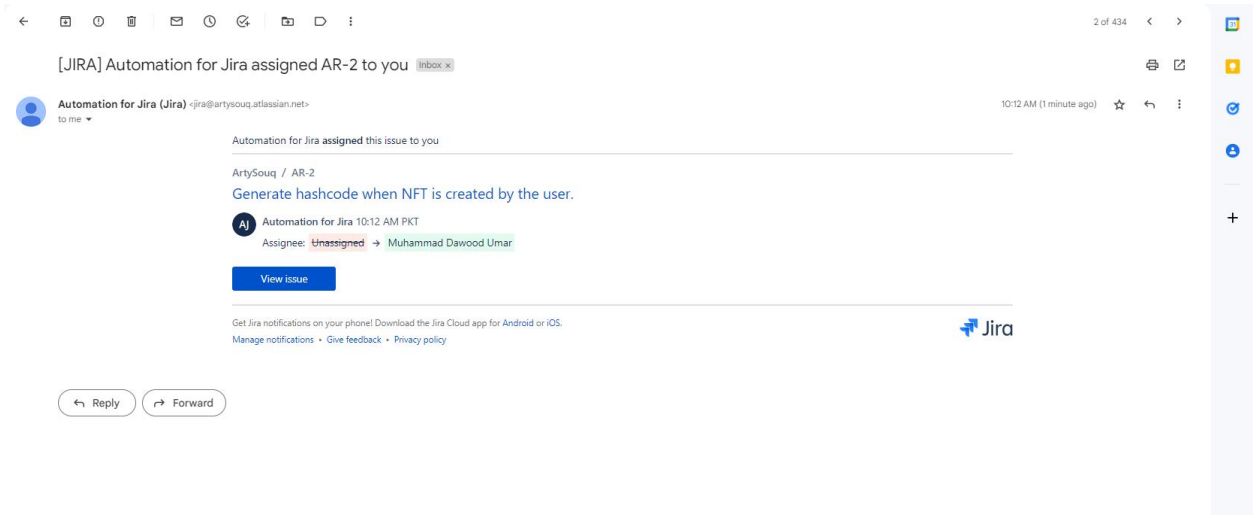
Lab Activity 2:

Rules 2

When new issue is created, an email is send to initiator or assigned issue to user.

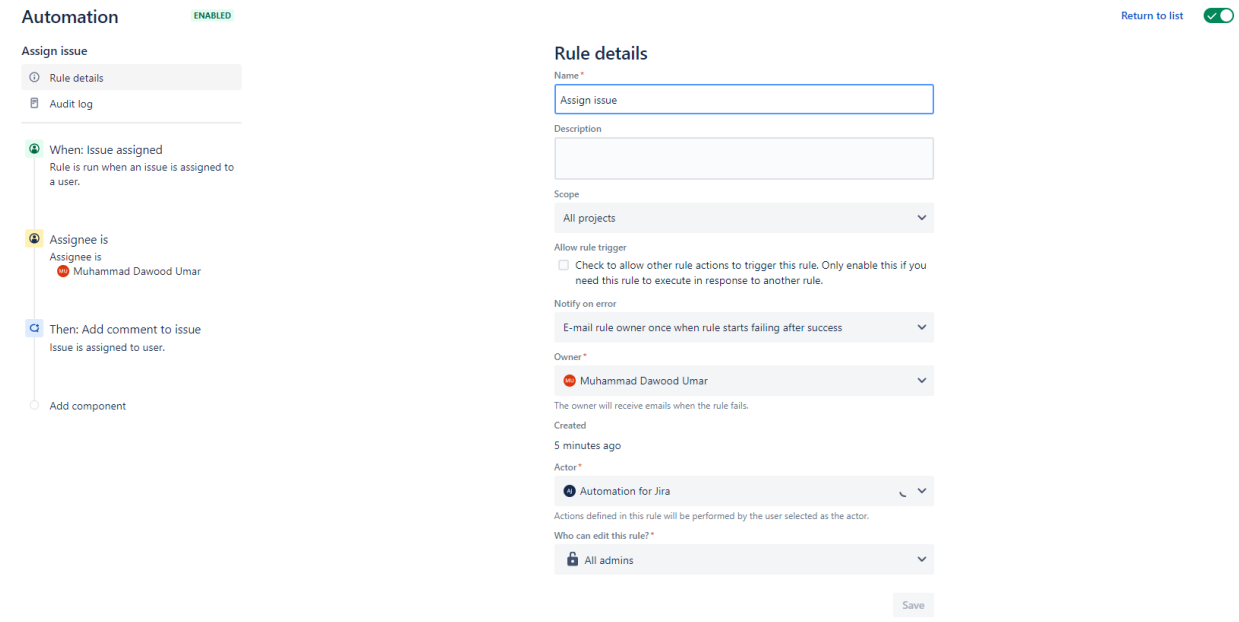


Action

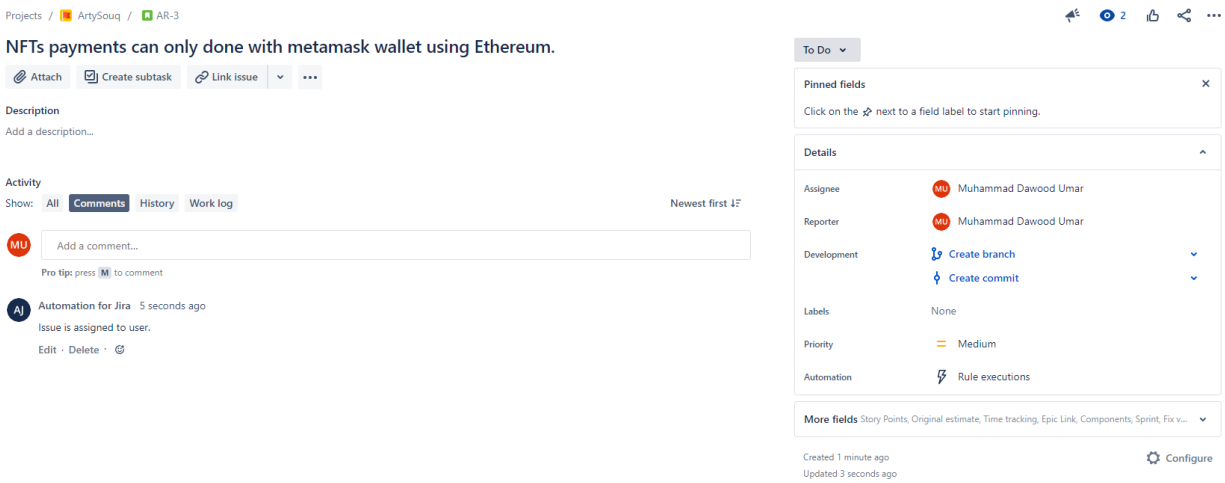


Lab Activity 3:

Rule 3



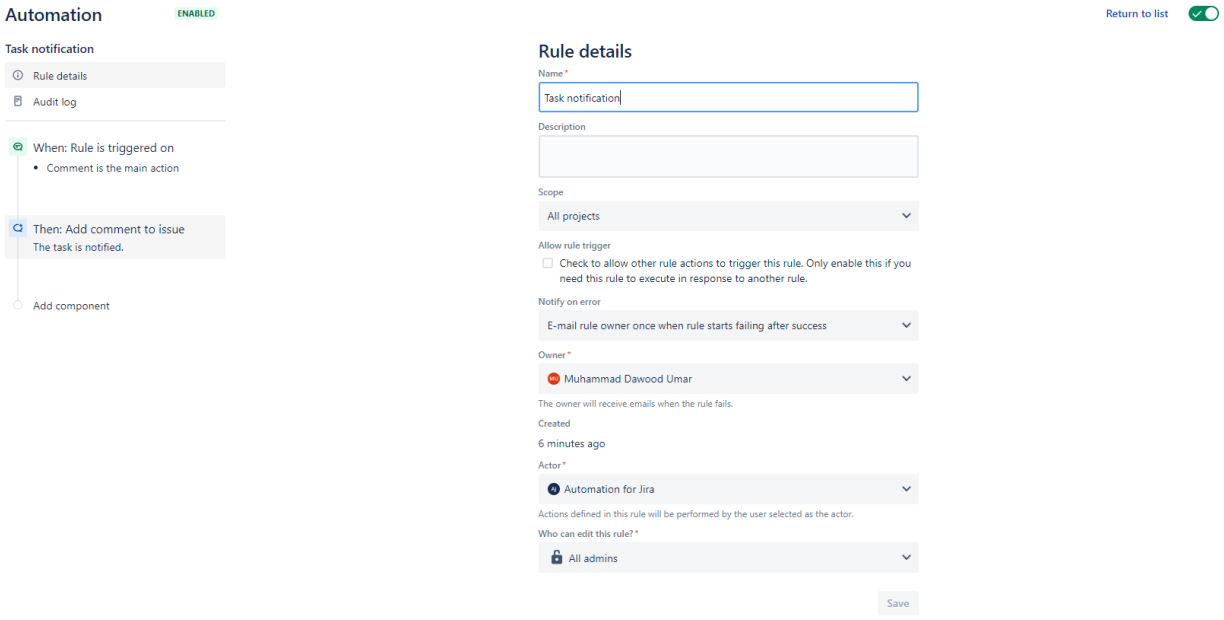
Action



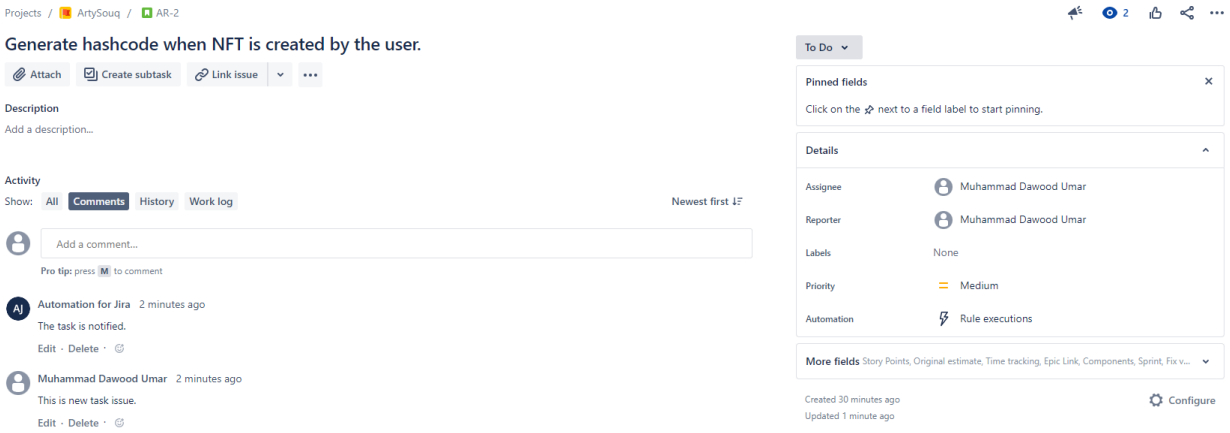
Lab Activity 4:

Rule 4

When user comment on the issue assignee reply with new task issue.



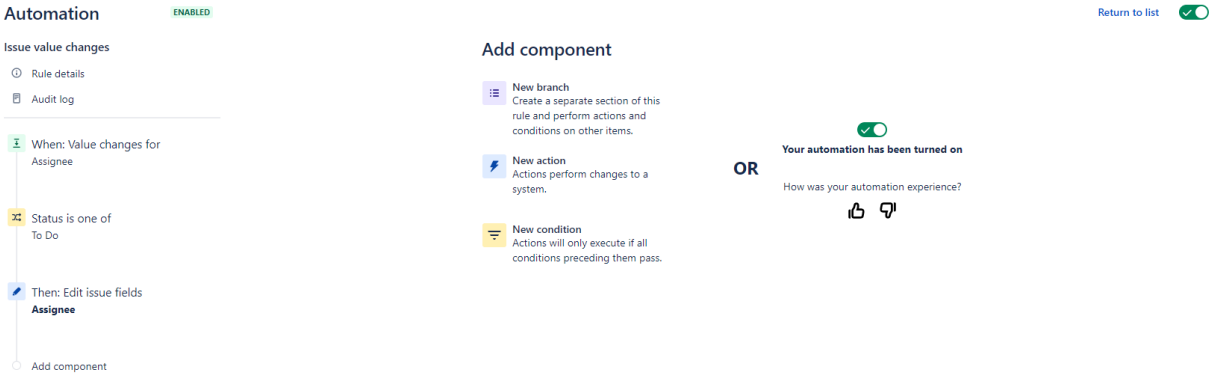
Action



Lab Activity 5:

Rule 5

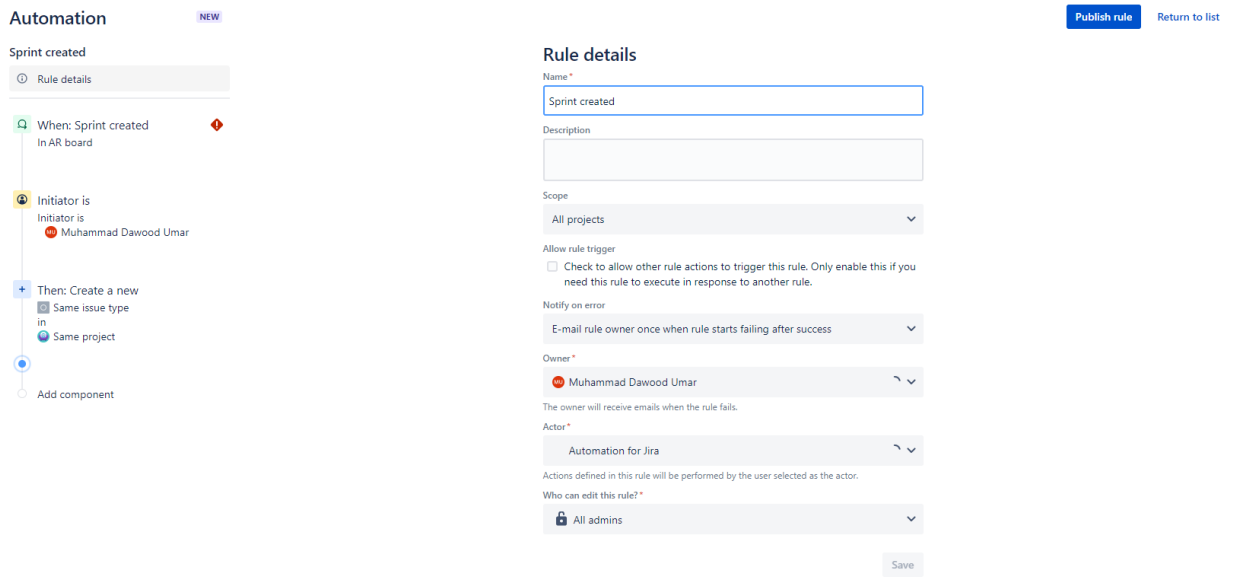
When assignee changes value of issue and status to do, the issue field can be edit by the assignee.



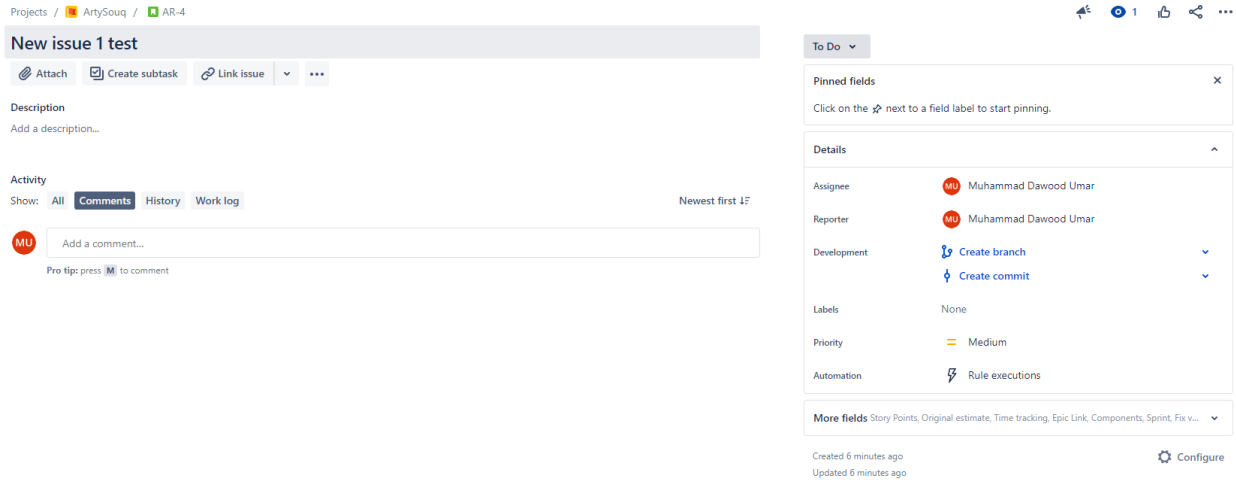
Lab Activity 6:

Rule 6

When new sprint created an issue is created against the sprint.



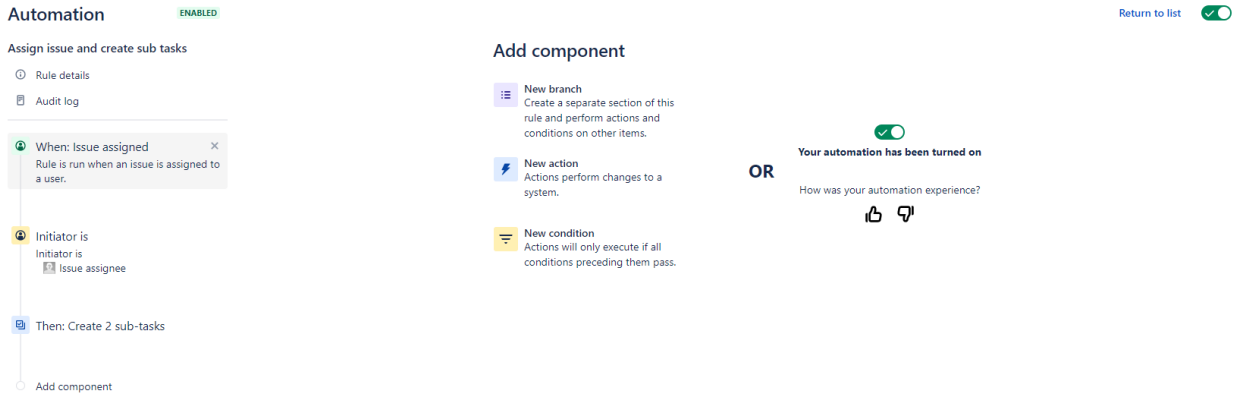
Action



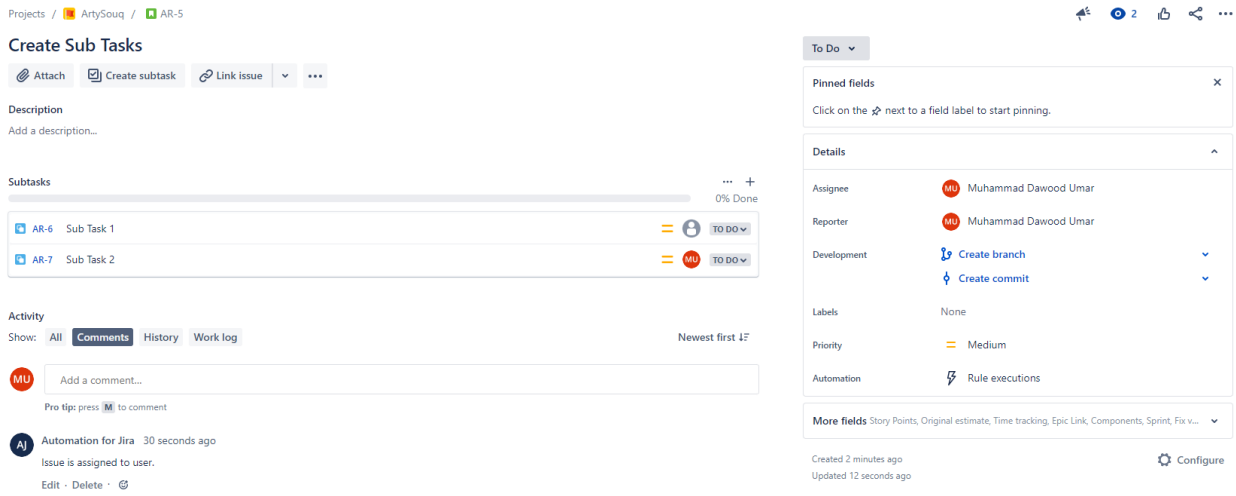
Lab Activity 7:

Rule 7

When issue is assigned to the user, system create sub tasks for user.



Action



3. Graded Lab Tasks

1. When new issue is created, an email is send to initiator or assigned issue to user.
2. When user comment on the issue assignee reply with new task issue.
3. When assignee changes value of issue and status to do, the issue field can be edit by the assignee.
4. When new sprint created an issue is created against the sprint.
5. When issue is assigned to the user, system create sub tasks for user.
6. an issue when the new issue is created. Upon creation it copies the summary of the newly created task and assign it to the cloned task
7. adds a comment to an issue containing name of the epic and the assignee when the assignee changes.
8. when an assignee of an issue is changed and its execution is based on condition that assignee must be Project Manager(Name) Once the assignee is matched the comment is added to the issue and the issue's status is changed to In Progress
9. when the comment is edited and it sends an email to the issue reporter that the comment has been changed.
10. when the description of an Epic is changed and it stores the description of the epic in a variable named "description". Then for each issue and story which is child of the epic are assigned the same description

Lab 06

JIRA (Zyphere Scale)

Objective:

The objective of this is to guide students through the process of managing test cases using Zephyr Scale within JIRA. By the end of this manual, users will be able to:

- Understand the basic concepts of Zephyr Scale for test case management.
- Create and manage test cases, test cycles, and test plans.
- Execute test cases and record test results.
- Generate reports and analyze test metrics.

Activity Outcomes:

By completing the activities in this manual, students will:

- Gain familiarity with Zephyr Scale's interface and features.
- Learn to create, organize, and manage test cases effectively.
- Execute test cycles and record the results.
- Develop skills to generate and interpret test reports.

1) Useful Concepts

1. Test Cases

Test Case: A set of actions executed to verify a particular feature or functionality of your software application.

Test Steps: The individual actions or instructions to be followed to complete the test case.

2. Test Cycles

Test Cycle: A collection of test cases that are executed together as part of a testing phase or sprint.

Execution: The process of running the test cases in a test cycle.

3. Test Plans

Test Plan: A document outlining the scope, approach, resources, and schedule of intended test activities.

Test Plan Management: Organizing and maintaining test plans to ensure comprehensive test coverage.

4. Reports and Metrics

Test Reports: Documents summarizing the results and findings from test executions.

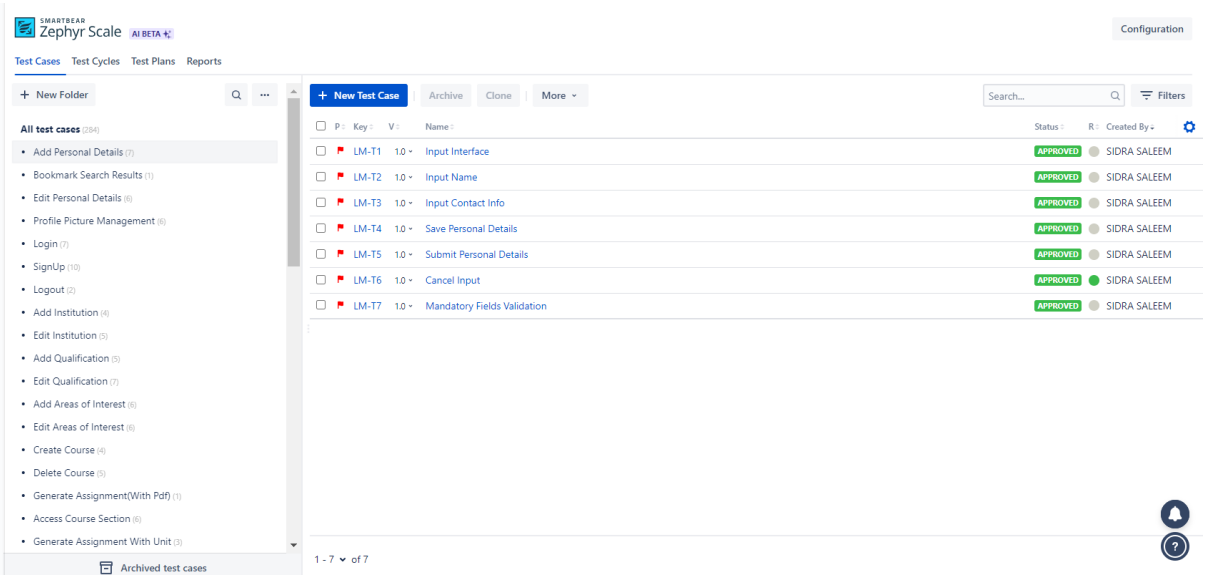
Metrics: Quantitative data used to measure the quality and effectiveness of the testing process.

2) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*10*	*Low*	*CLO-4*
2	*10*	*Low*	*CLO-4*
3	*10*	*Low*	*CLO-4*
4	*10*	*Medium*	*CLO-4*
5	*10*	*Medium*	*CLO-4*

Lab Activity 1:

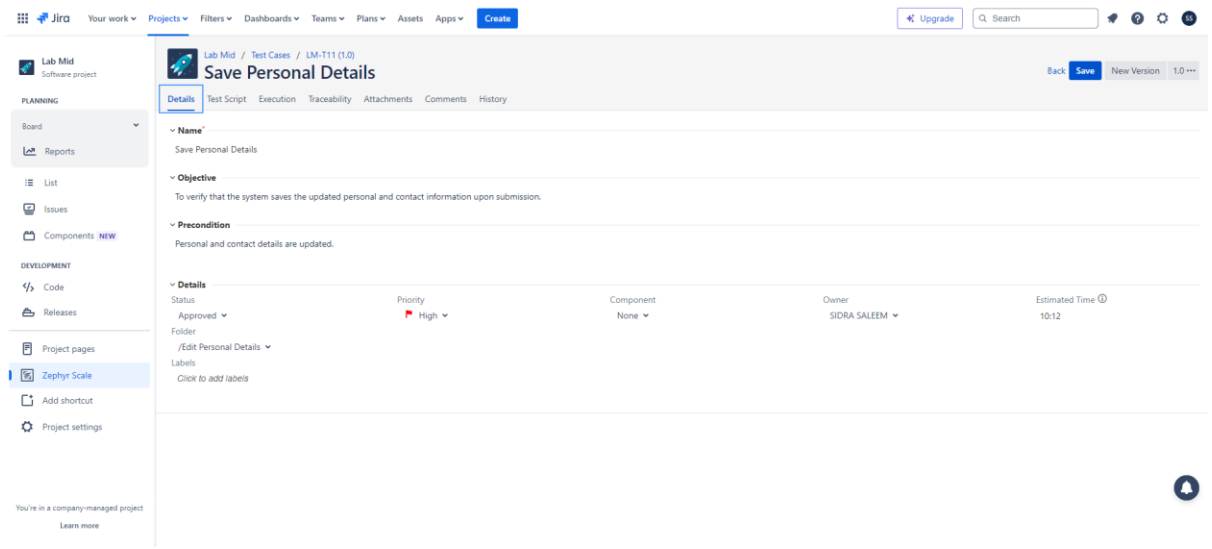
Test Cases :



Test Case Components

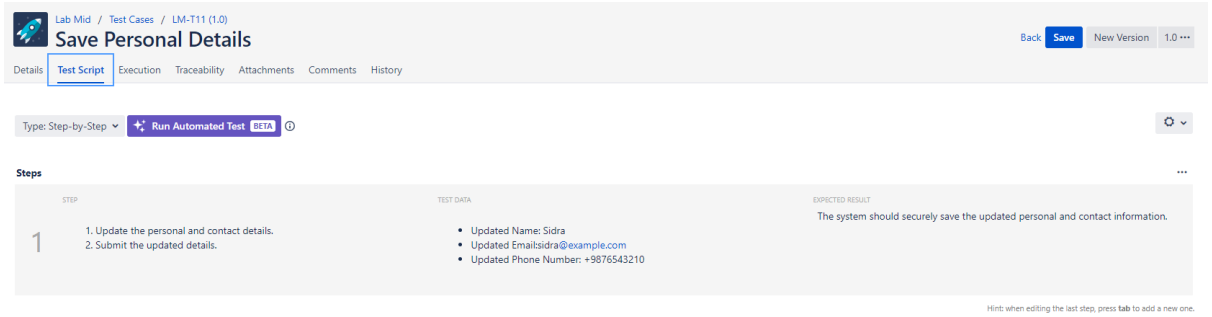
Following are the test case components in each test case in the zypher scale

Details

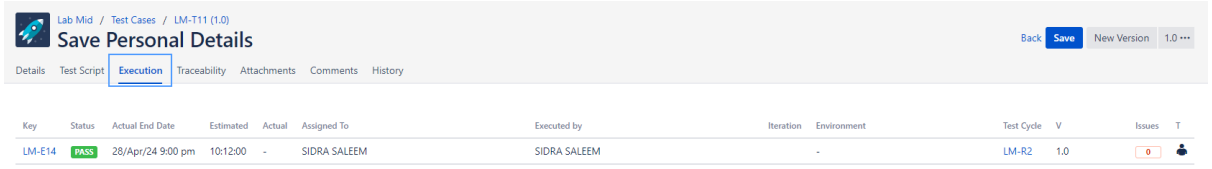


Lab Activity 2:

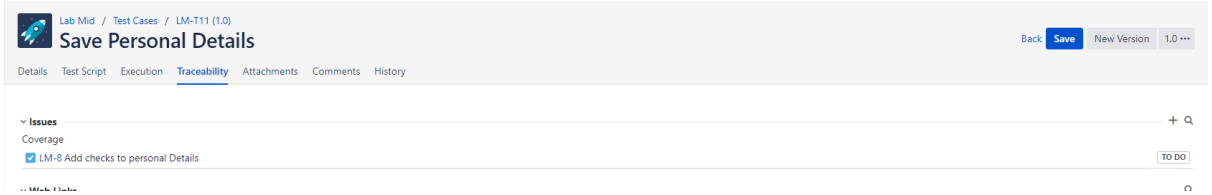
Test Script



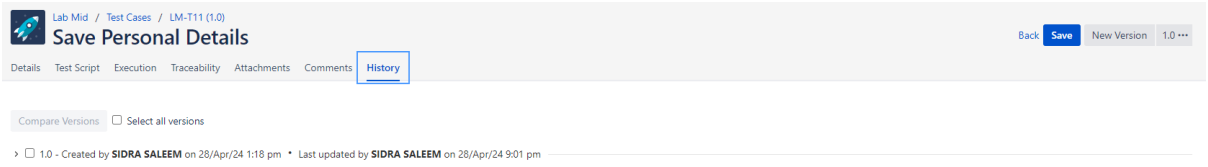
Execution



Traceability

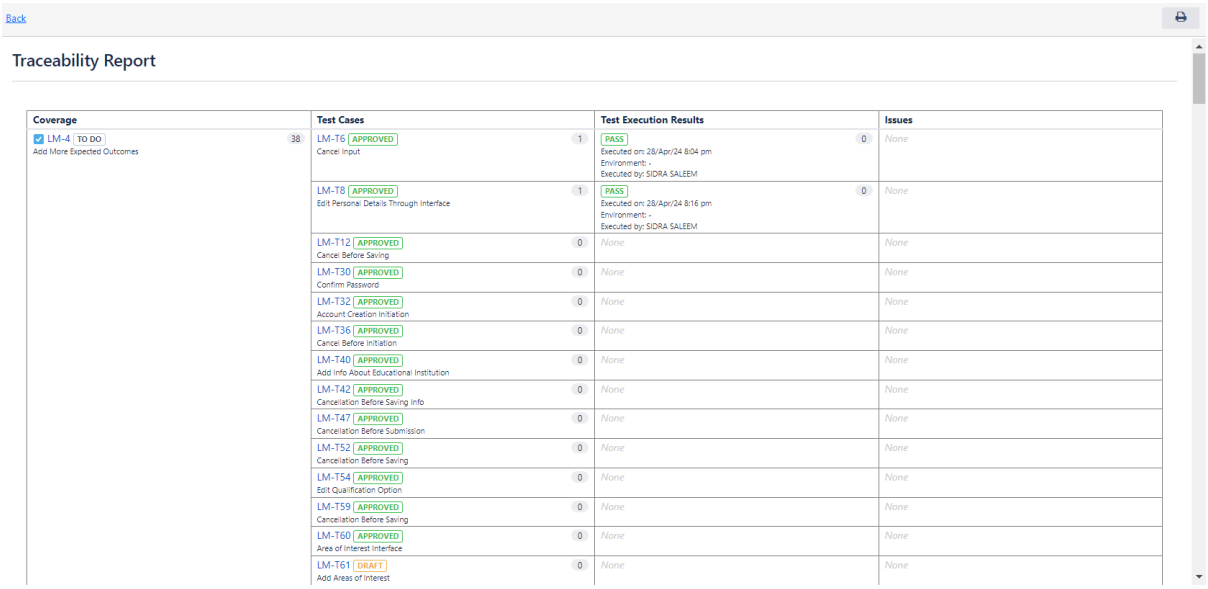


History



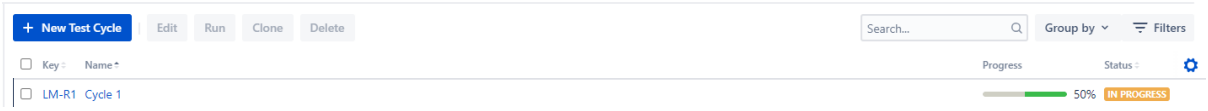
Lab Activity 3:

Test Case Traceability Report

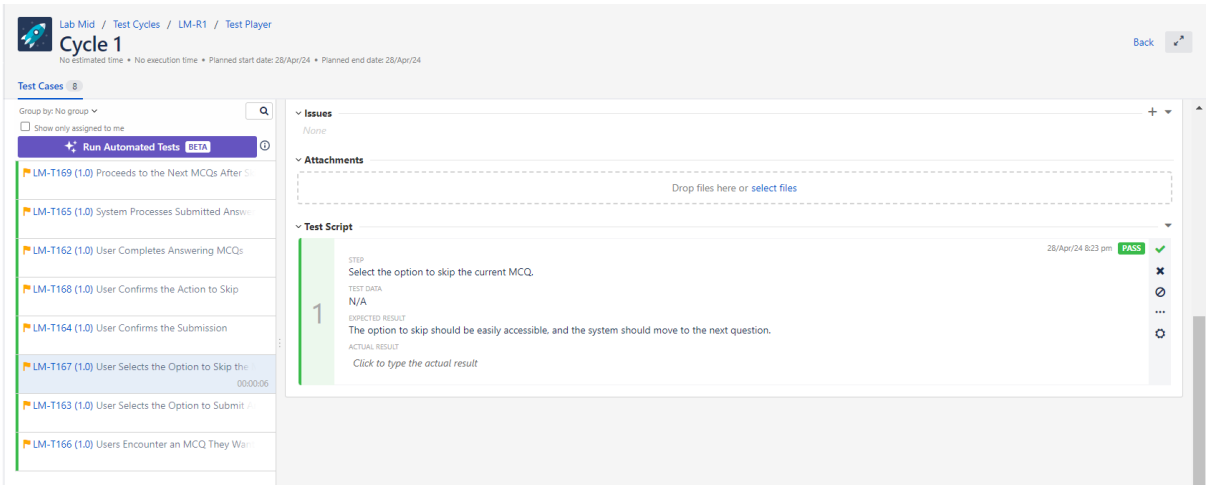


Test Case Cycle

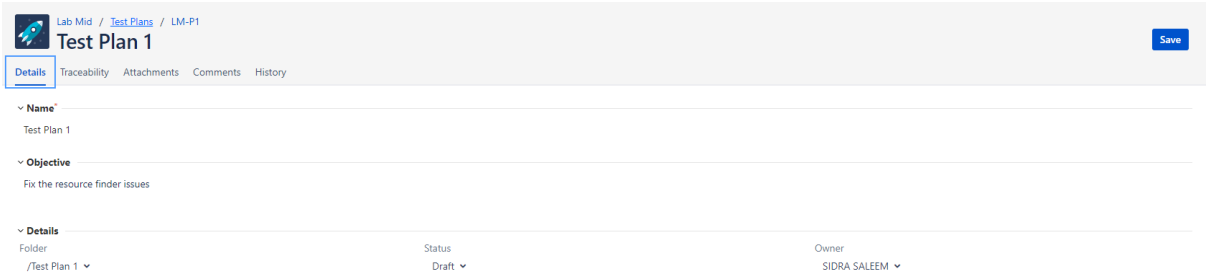
Test Case Cycle is executed by 50 percent , it means Half of the test cases in this cycle have been executed



Test Case Cycle Execution

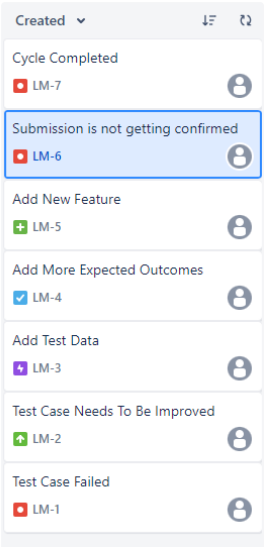


Test Plan

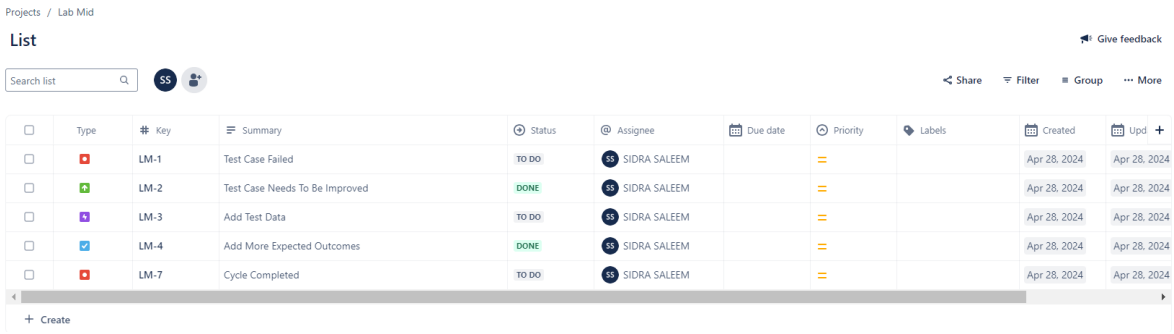


Lab Activity 4:

Issues



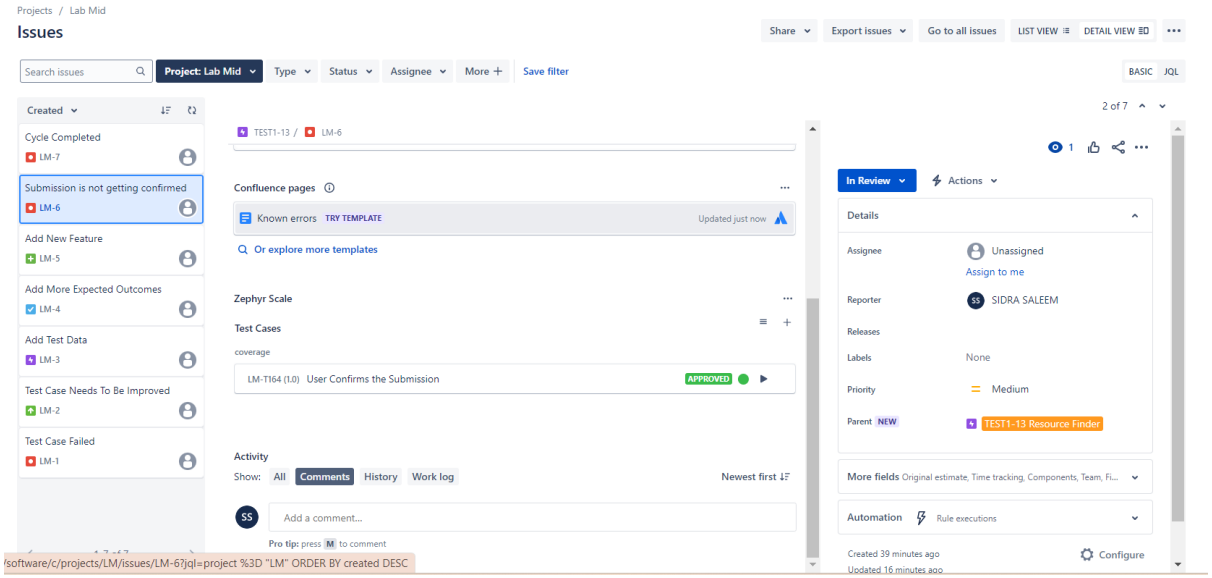
List



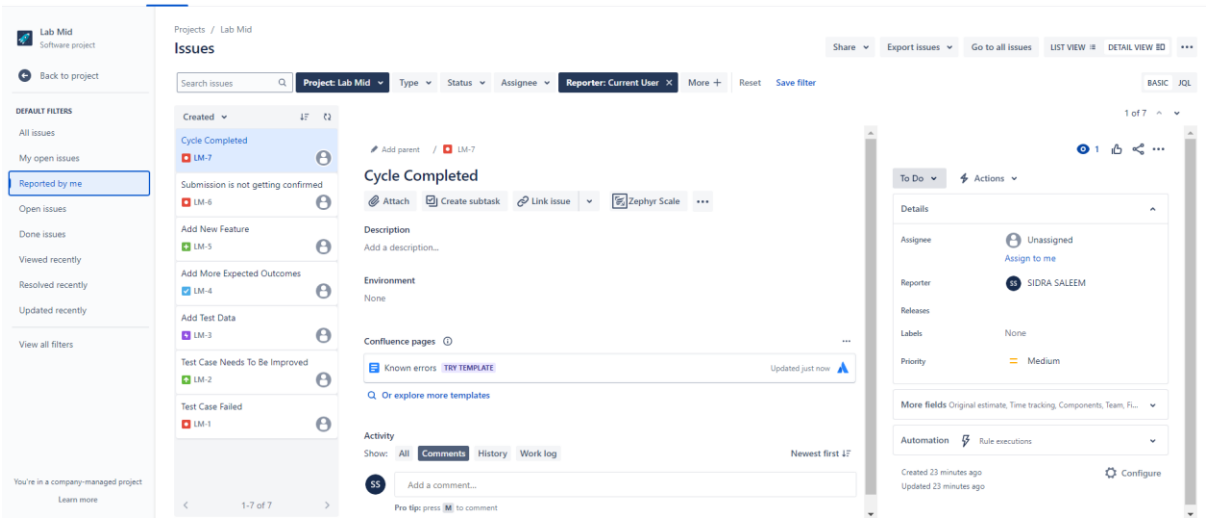
Issues Linkage

The Test Case(164) from zypher scale is linked with the issue and the parent test case here is TC-13

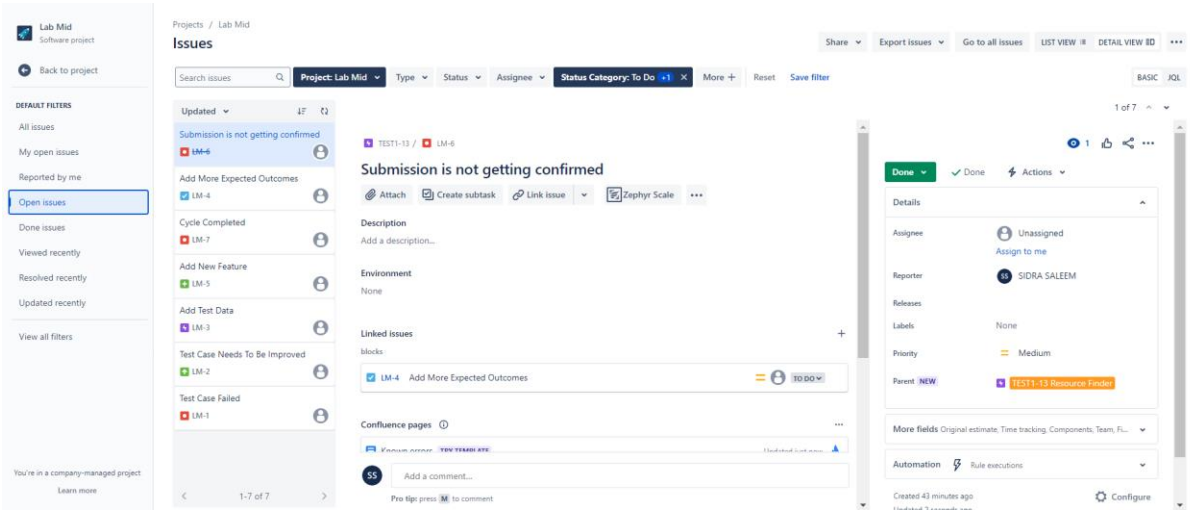
Test Case Assigned to : Sidra Saleem



Issues Reported by a certain user

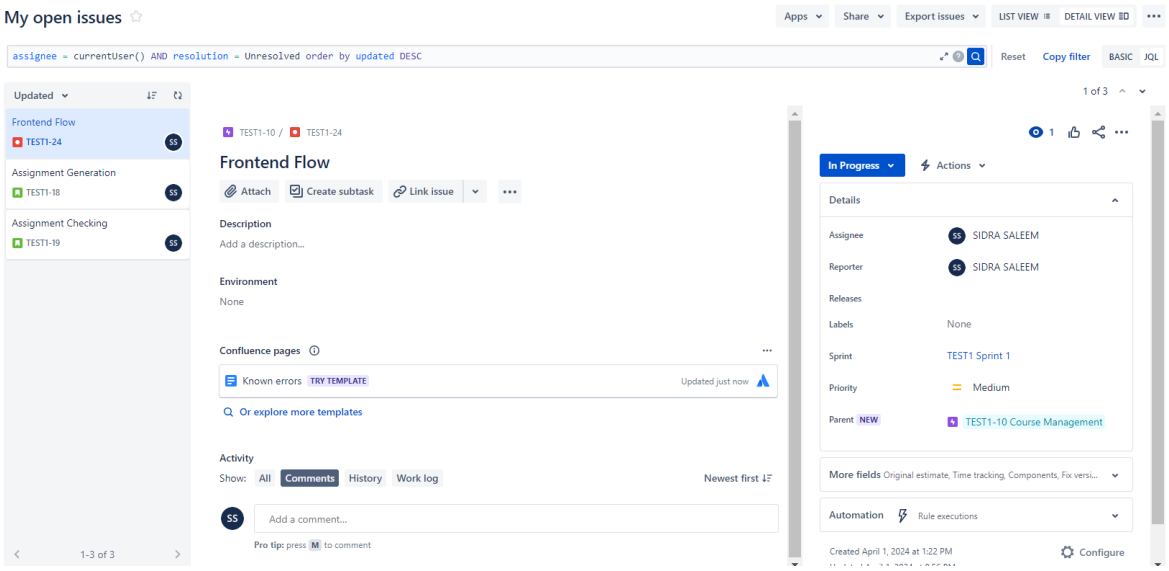


Issues open vs Issues in Progress vs Completed

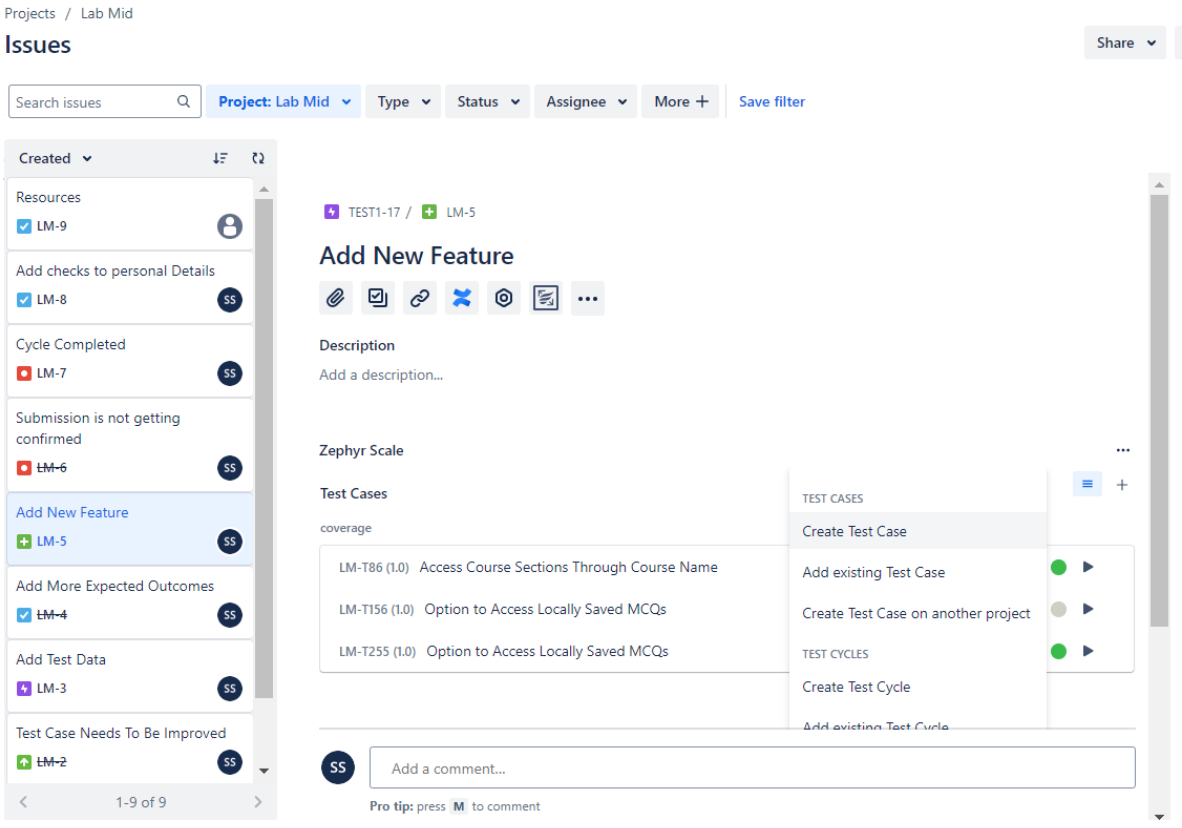


Lab Activity 5:

Open Issues



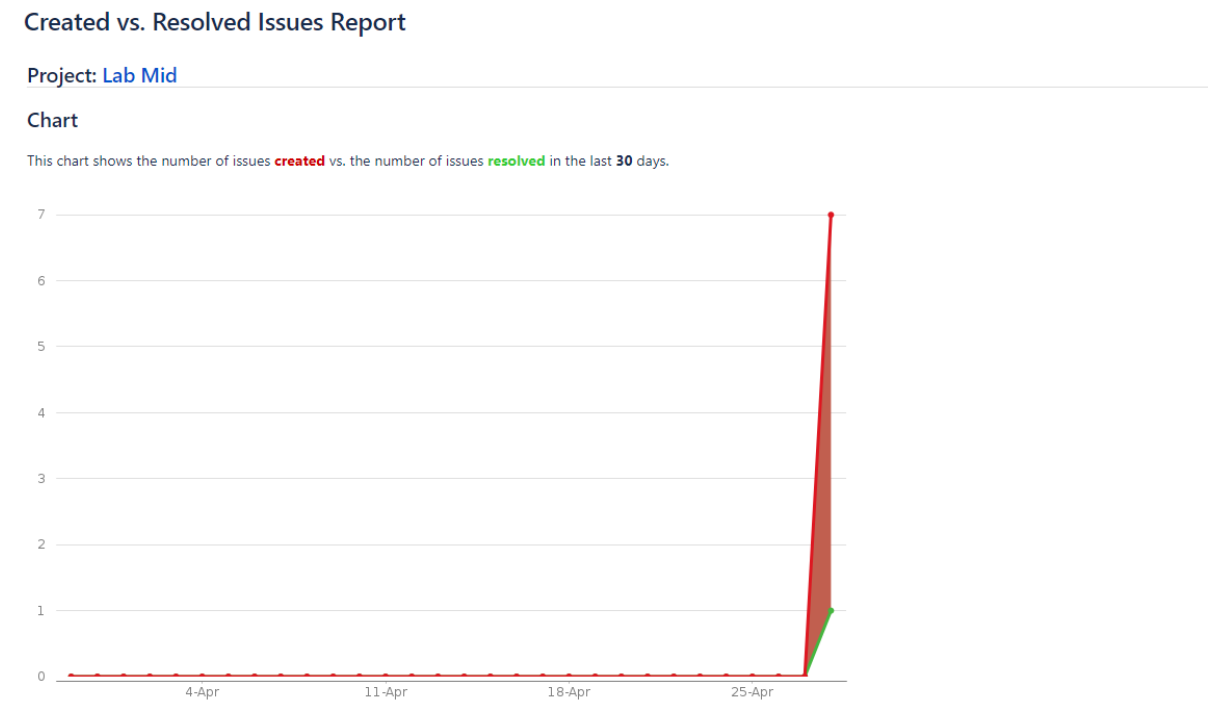
Add the test cases to issues with Zypher



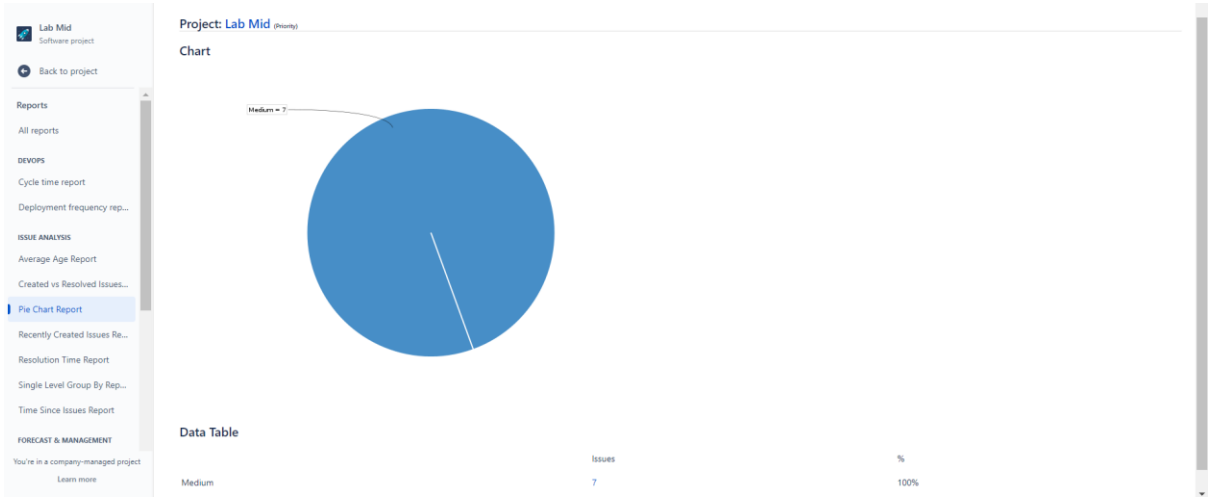
Lab Activity 5:

Reports

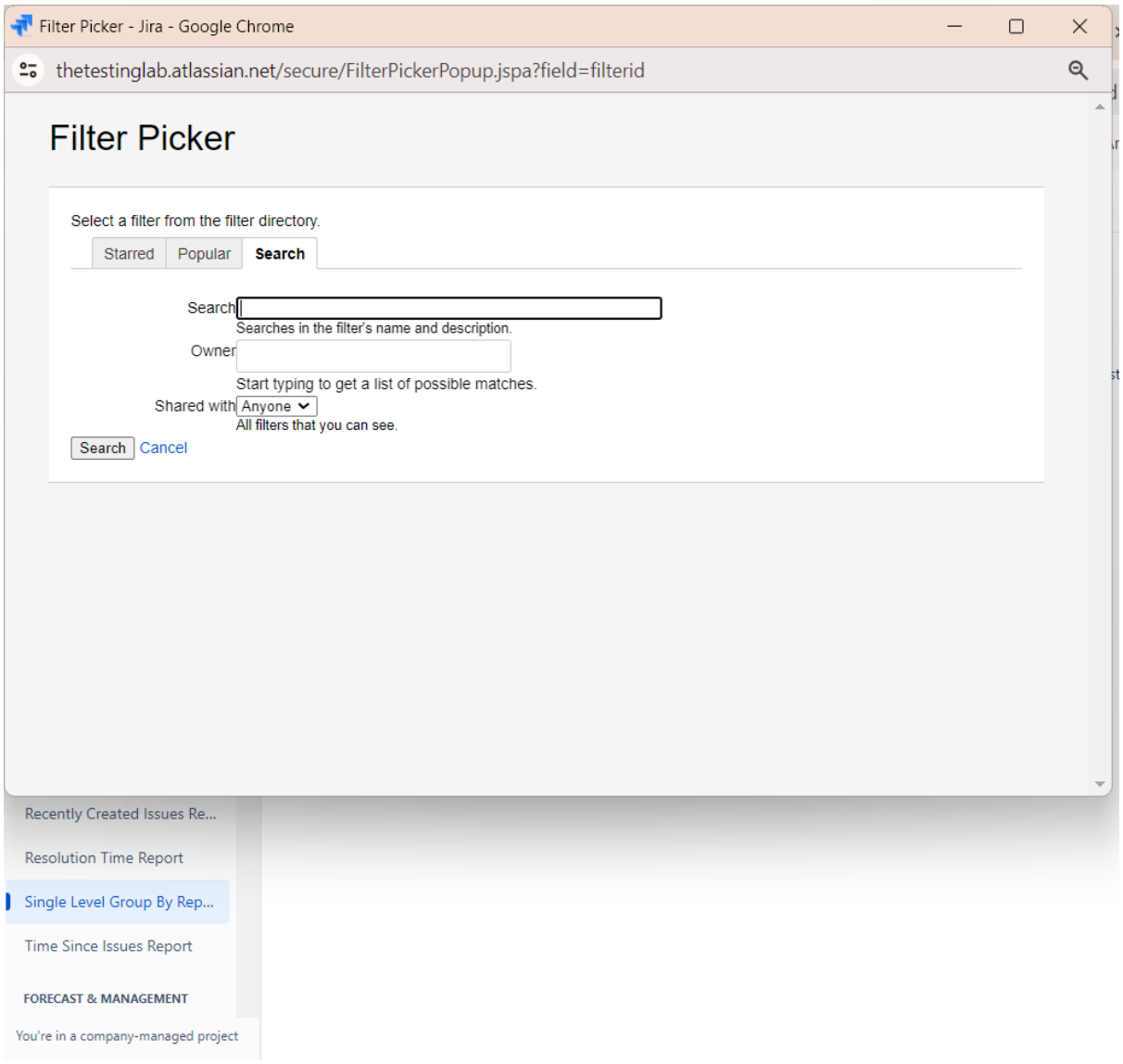
Created vs Resolved



Pie Chart Report



Filter Picker



Time Tracking Report

Time Tracking Report

Description:
Shows the original and current time estimates for issues in the current project. This can help you determine whether work is on track for those issues.

Excel View

Time Tracking Report for Lab Mid

Only including sub-tasks with the selected version

Progress: 100% — **1d** completed from current total estimate of **1d**

Accuracy: 0% — Issues in this version are behind the original estimate of **0m** by **1 day**.

Key		Summary	Original Estimate	Σ	Est. Time Remaining	Σ	Time Spent	Σ	Accuracy	Σ
LM-2	DONE	Test Case Needs To Be Improved	-	-	0m	0m	1d	1d	?	?
LM-1	TO DO	Test Case Failed	-	-	-	-	-	-	-	-
LM-3	TO DO	Add Test Data	-	-	-	-	-	-	-	-
LM-4	DONE	Add More Expected Outcomes	-	-	-	-	-	-	-	-
LM-5	TO DO	Add New Feature	-	-	-	-	-	-	-	-
LM-6	DONE	Submission is not getting confirmed	-	-	-	-	-	-	-	-
LM-7	TO DO	Cycle Completed	-	-	-	-	-	-	-	-
		Total	**0m**		**0m**		**1d**		**-1d**	

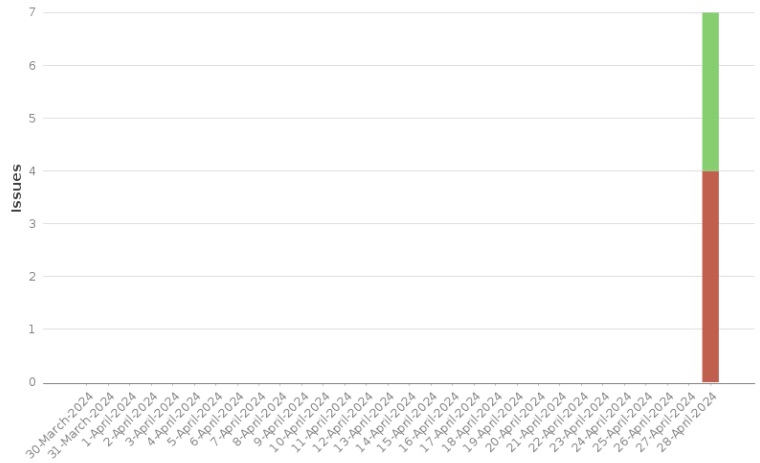
Recently Created Issues

Recently Created Issues Report

Project: Lab Mid

Chart

This chart shows the issues created in the last **30** days



3) Graded Lab Tasks

- Manage your FYP test cases in Zyphyr Scale and provide test cases report

Lab 07

JUNIT

Objective:

- To understand the fundamentals of unit testing with JUnit.
- To learn how to write, run, and interpret the results of JUnit tests.
- To gain practical experience in test-driven development.

Activity Outcomes:

- Ability to set up JUnit in an IDE.
- Proficiency in writing test cases using JUnit annotations.
- Understanding of assertions and assumptions in JUnit.
- Knowledge of test lifecycle and execution order.

Useful Concepts

JUNIT is an open source framework to write repeatable tests for JAVA programming language.. It is an important tool for test driven development. Unit testing is used to verify the small chunks of code by creating a class, function etc.

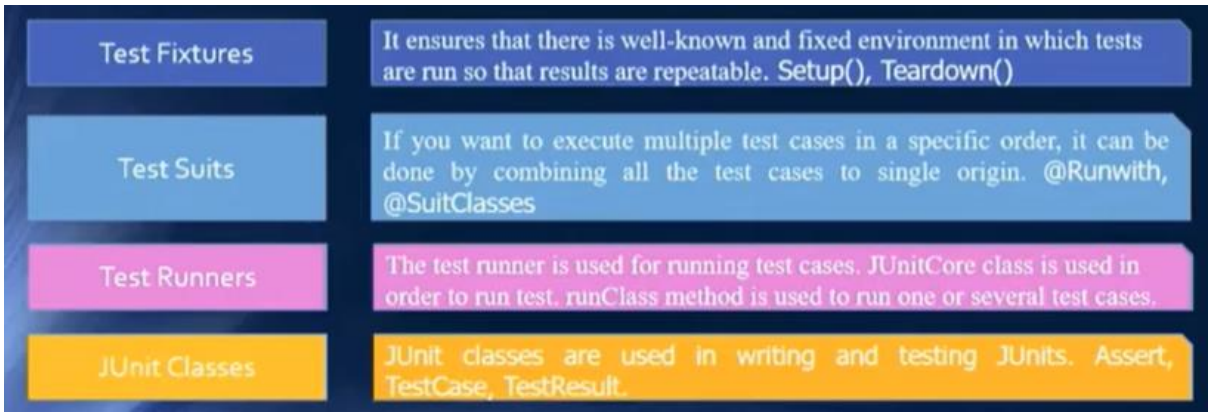
Why JUNIT:

- To write and run repeatable tests.
- JUnit provides a framework to keep tests small and simple to test specific areas of code.
- JUnit shows test progress in a bar that is green if testing is going fine and it turns red when a test fails.
- We can run multiple tests concurrently.
- The simplicity of JUnit makes it possible for the software developer to easily correct bugs as they are found.

Features of JUnit Test Framework

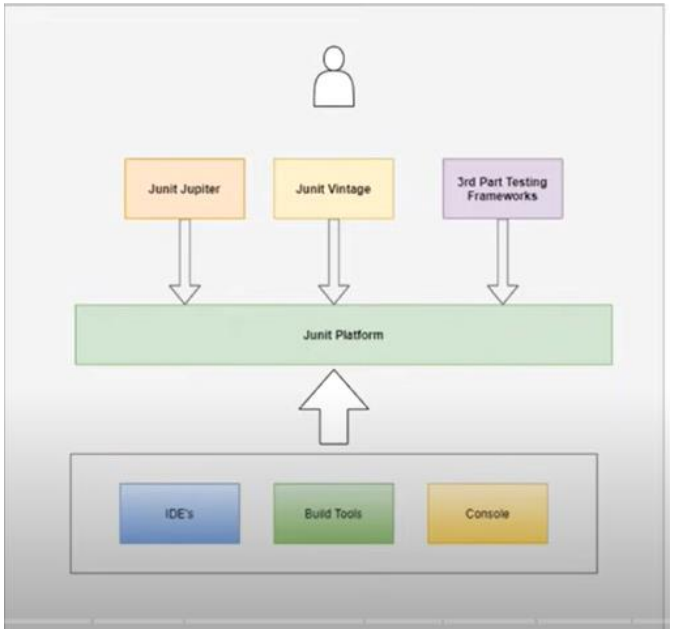
JUnit test framework provides the following important features −

- Fixtures
- Test suites
- Test runners
- JUnit classes



JUNIT 4 and JUNIT 5 Differences:

- JUnit 5 aims to adapt the Java 8 style of coding and to be more robust and flexible than JUnit 4.
- JUnit 4 has everything bundled into a single jar file.
- JUnit 5 is composed of 3 sub-projects i.e. JUnit Platform, JUnit Jupiter and JUnit Vintage.
- **JUnit Platform:** It defines the TestEngine API for developing new testing frameworks that run on the platform. It is platform which provides an API to launch the test from either the IDE's, Biiltin tools or console as shown in figure.
- **JUnit Jupiter:** It has all new JUnit annotations and TestEngine implementation to run tests written with these annotations.
- **JUnit Vintage:** To support running JUnit 3 and JUnit 4 written tests on the JUnit 5 platform.



- Junit 4 requires Java 5 or higher.
- Junit 5 requires Java 8 or higher.

Feature	JUnit 4	Junit 5
Declare a test method	`@Test`	`@Test`
Execute before all test methods in the current class	`@BeforeClass`	`@BeforeAll`
Execute after all test methods in the current class	`@AfterClass`	`@AfterAll`
Execute before each test method	`@Before`	`@BeforeEach`
Execute after each test method	`@After`	`@AfterEach`
Disable a test method/class	`@Ignore`	`@Disabled`
Test factory for dynamic tests	NA	`@TestFactory`
Nested tests	NA	`@Nested`
Tagging and filtering	`@Category`	`@Tag`
Register custom extensions	NA	`@ExtendWith`

Fixtures

- setUp() method, which runs before every test invocation. i.e we will use @Before annotation
- tearDown() method, which runs after every test method. i.e we will use @After annotation
- Usually, there are some repeated tasks that must be done prior to each test case.

  Example:
  create a database connection.
- Likewise, at the end of each test case, there may be some repeated tasks.

  Example: to clean up once test execution is over.
- JUnit provides annotations that help in setup and teardown. It ensures that resources are released, and the test system is in a ready state for next test case.

Annotations:

Annotations is a special form of syntactic meta-data that can be added to Java source code for better code readability and structure.

Some basic annotations:

@Before

This annotation is used if you want to execute some statement such as preconditions before each test case.

@BeforeClass

This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.

@After

This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.

@AfterClass

This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.

@Ignores

This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.

EXAMPLE:

Assertions:

Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org. junit. Assert which extends java. ... Object class. There are various types of assertions like Boolean, Null, Identical etc

Assert Equals

If you want to test equality of two objects, you have the following methods

- assertEquals(expected, actual)

It will return true if: expected.equals(actual) returns true.

Example:

The *assertEquals* assertion verifies that the expected and the actual values are equal:

```
@Test
public void whenAssertingEquality_thenEqual() {
    String expected = "Baeldung";
    String actual = "Baeldung";

    assertEquals(expected, actual);
}
```

It's also possible to specify a message to display when the assertion fails:

```
assertEquals("failure - strings are not equal", expected, actual);
```

Assert Array Equals

If we want to assert that two arrays are equals, we can use the assertArrayEquals:

assertArrayEquals(expected, actual)

Example:

```
@Test
public void whenAssertingArraysEquality_thenEqual() {
    char[] expected = {'J','u','n','i','t'};
    char[] actual = "Junit".toCharArray();

    assertArrayEquals(expected, actual);
}
```

If both arrays are *null*, the assertion will consider them equal:

```
@Test
public void givenNullArrays_whenAssertingArraysEquality_thenEqual() {
    int[] expected = null;
    int[] actual = null;

    assertArrayEquals(expected, actual);
}
```

assertNotSame and assertSame

With assertNotSame, it's possible to verify if two variables don't refer to the same object:

```
@Test
public void whenAssertingNotSameObject_thenDifferent() {
    Object cat = new Object();
    Object dog = new Object();

    assertNotSame(cat, dog);
}
```

Otherwise, when we want to verify that two variables refer to the same object, we can use the *assertSame* assertion.

Boolean

In case we want to verify that a certain condition is true or false, we can respectively use the assertTrue assertion or the assertFalse one:

- assertTrue(condition)
- assertFalse(condition)

```
@Test
public void whenAssertingConditions_thenVerified() {
    assertTrue("5 is greater then 4", 5 > 4);
    assertFalse("5 is not greater then 6", 5 > 6);
}
```

Null object

If you want to check the initial value of an object/variable, you have the following methods:

- assertNull(object)
- assertNotNull(object)

Identical

If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

- assertSame(expected, actual), It will return true if expected == actual
- assertNotSame(expected, actual)

Fail:

When writing unit tests, we can use ***fail*** to explicitly create a failure under desired testing conditions.

- Incomplete Test (We can fail a test when it is incomplete or not yet implemented:)

```
@Test
public void incompleteTest() {
    fail("Not yet implemented");
}
```

- Expected Exception (We can also do it when we think an exception will happen:)

```
@Test
public void expectedException() {
    try {
        methodThrowsException();
        fail("Expected exception was not thrown");
    } catch (Exception e) {
        assertNotNull(e);
    }
}
```

- Unexpected Exception (Failing the test when an exception is not expected to be thrown is another option:)

```
@Test
public void unexpectedException() {
    try {
        safeMethod();
        // more testing code
    } catch (Exception e) {
        fail("Unexpected exception was thrown");
    }
}
```

-

The fail assertion fails a test throwing an AssertionFailedError. It can be used to verify that an actual exception is thrown or when we want to make a test failing during its development.

JUNIT is an open source framework to write repeatable tests for JAVA programming language.. It is an important tool for test driven development. Unit testing is used to verify the small chunks of code by creating a class, function etc.

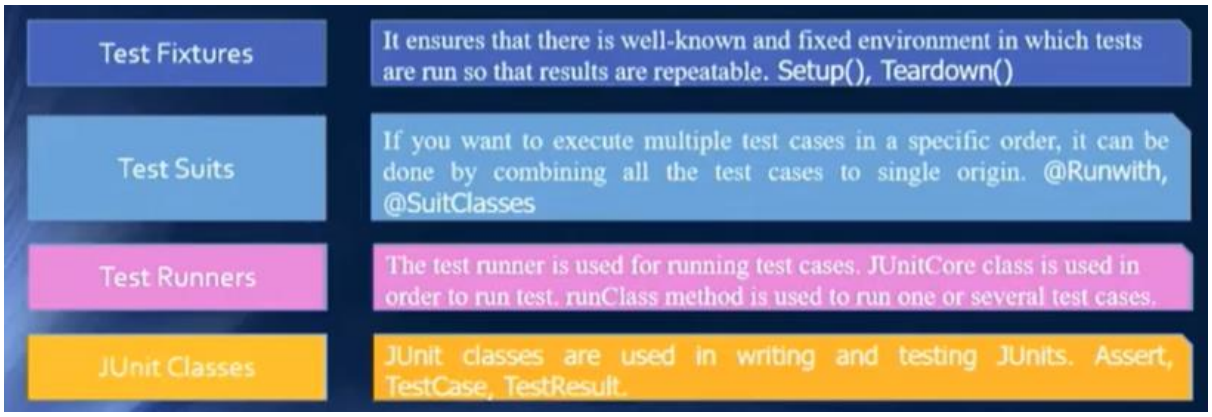
Why JUNIT:

- To write and run repeatable tests.
- JUnit provides a framework to keep tests small and simple to test specific areas of code.
- JUnit shows test progress in a bar that is green if testing is going fine and it turns red when a test fails.
- We can run multiple tests concurrently.
- The simplicity of JUnit makes it possible for the software developer to easily correct bugs as they are found.

Features of JUnit Test Framework

JUnit test framework provides the following important features −

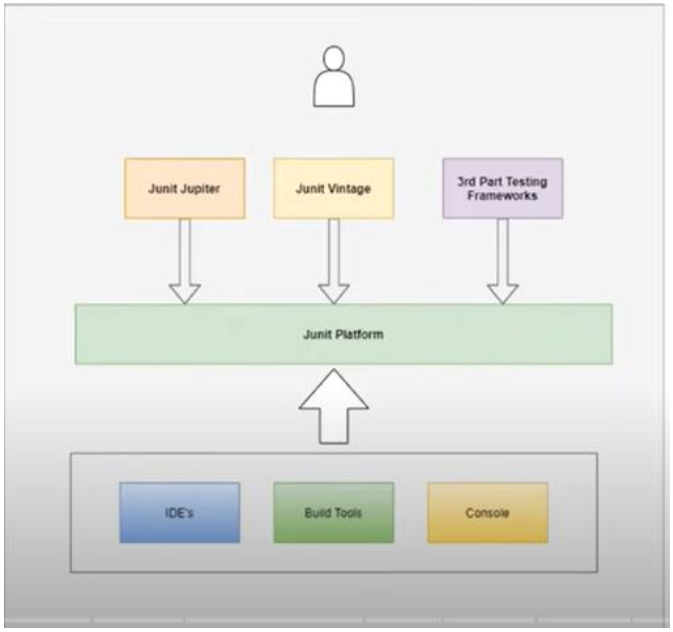
- Fixtures
- Test suites
- Test runners
- JUnit classes



JUNIT 4 and JUNIT 5 Differences:

- JUnit 5 aims to adapt the Java 8 style of coding and to be more robust and flexible than JUnit 4.
- JUnit 4 has everything bundled into a single jar file.
- **JUnit 5 is composed of 3 sub-projects i.e.** JUnit Platform, JUnit Jupiter and JUnit Vintage.

- **JUnit Platform:** It defines the TestEngine API for developing new testing frameworks that run on the platform. It is platform which provides an API to launch the test from either the IDE's, Biiltin tools or console as shown in figure.
- **JUnit Jupiter:** It has all new JUnit annotations and TestEngine implementation to run tests written with these annotations.
- **JUnit Vintage:** To support running JUnit 3 and JUnit 4 written tests on the JUnit 5 platform.



- Junit 4 requires Java 5 or higher.
- Junit 5 requires Java 8 or higher.

Feature	JUnit 4	Junit 5
Declare a test method	`@Test`	`@Test`
Execute before all test methods in the current class	`@BeforeClass`	`@BeforeAll`
Execute after all test methods in the current class	`@AfterClass`	`@AfterAll`
Execute before each test method	`@Before`	`@BeforeEach`
Execute after each test method	`@After`	`@AfterEach`
Disable a test method/class	`@Ignore`	`@Disabled`
Test factory for dynamic tests	NA	`@TestFactory`
Nested tests	NA	`@Nested`
Tagging and filtering	`@Category`	`@Tag`
Register custom extensions	NA	`@ExtendWith`

Fixtures

- setUp() method, which runs before every test invocation. i.e we will use @Before annotation
- tearDown() method, which runs after every test method. i.e we will use @After annotation

- Usually, there are some repeated tasks that must be done prior to each test case.

  Example:
  create a database connection.
- Likewise, at the end of each test case, there may be some repeated tasks.

  Example: to clean up once test execution is over.
- JUnit provides annotations that help in setup and teardown. It ensures that resources are released, and the test system is in a ready state for next test case.

Annotations:

Annotations is a special form of syntactic meta-data that can be added to Java source code for better code readability and structure.

Some basic annotations:

@Before

This annotation is used if you want to execute some statement such as preconditions before each test case.

@BeforeClass

This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.

@After

This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.

@AfterClass

This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.

@Ignores

This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.

EXAMPLE:

Writing Tests in JUnit5

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
```

```
class MyFirstJUnitJupiterTests {

  private final Calculator calculator = new Calculator();

  @Test

  void addition() {

    assertEquals(2, calculator.add(1, 1));

  }

}

public class Calculator {

  public int add(int a, int b) {

    return a + b;

  }

}
```

Explanation

- @Test is an **annotation**. Declares the immediate method after it (in this case ***addition***) as a test method.
- assertEquals is an **assertion**. Takes in two arguments just like "assertEquals(**expected value, actual value, optional_string_message**)".
- There are **93** variations of **assertEquals** method in **Assertions** class.

Assertions:

Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org. junit. Assert which extends java. ... Object class. There are various types of assertions like Boolean, Null, Identical etc

Assert Equals

If you want to test equality of two objects, you have the following methods

- assertEquals(expected, actual)

It will return true if: expected.equals(actual) returns true.

Example:

The *assertEquals* assertion verifies that the expected and the actual values are equal:

```
@Test
public void whenAssertingEquality_thenEqual() {
    String expected = "Baeldung";
    String actual = "Baeldung";

    assertEquals(expected, actual);
}
```

It's also possible to specify a message to display when the assertion fails:

```
assertEquals("failure - strings are not equal", expected, actual);
```

Assert Array Equals

If we want to assert that two arrays are equals, we can use the assertArrayEquals:

assertArrayEquals(expected, actual)

Example:

```
@Test
public void whenAssertingArraysEquality_thenEqual() {
    char[] expected = {'J','u','n','i','t'};
    char[] actual = "Junit".toCharArray();

    assertArrayEquals(expected, actual);
}
```

If both arrays are *null*, the assertion will consider them equal:

```
@Test
public void givenNullArrays_whenAssertingArraysEquality_thenEqual() {
    int[] expected = null;
    int[] actual = null;

    assertArrayEquals(expected, actual);
}
```

assertNotSame and assertSame

With assertNotSame, it's possible to verify if two variables don't refer to the same object:

```
@Test
public void whenAssertingNotSameObject_thenDifferent() {
    Object cat = new Object();
    Object dog = new Object();

    assertNotSame(cat, dog);
}
```

Otherwise, when we want to verify that two variables refer to the same object, we can use the *assertSame* assertion.

Boolean

In case we want to verify that a certain condition is true or false, we can respectively use the assertTrue assertion or the assertFalse one:

- assertTrue(condition)
- assertFalse(condition)

```
@Test
public void whenAssertingConditions_thenVerified() {
    assertTrue("5 is greater then 4", 5 > 4);
    assertFalse("5 is not greater then 6", 5 > 6);
}
```

Null object

If you want to check the initial value of an object/variable, you have the following methods:

- assertNull(object)
- assertNotNull(object)

Identical

If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

- assertSame(expected, actual), It will return true if expected == actual
- assertNotSame(expected, actual)

Fail:

When writing unit tests, we can use ***fail*** to explicitly create a failure under desired testing conditions.

- Incomplete Test (We can fail a test when it is incomplete or not yet implemented:)

```
@Test
public void incompleteTest() {
    fail("Not yet implemented");
}
```

- Expected Exception (We can also do it when we think an exception will happen:)

```
@Test
public void expectedException() {
    try {
        methodThrowsException();
        fail("Expected exception was not thrown");
    } catch (Exception e) {
        assertNotNull(e);
    }
}
```

- Unexpected Exception (Failing the test when an exception is not expected to be thrown is another option:)

```
@Test
public void unexpectedException() {
    try {
        safeMethod();
        // more testing code
    } catch (Exception e) {
        fail("Unexpected exception was thrown");
    }
}
```

-

The fail assertion fails a test throwing an AssertionFailedError. It can be used to verify that an actual exception is thrown or when we want to make a test failing during its development.

```
@Test
public void whenCheckingExceptionMessage_thenEqual() {
    try {
        methodThatShouldThrowException();
        fail("Exception not thrown");
    } catch (UnsupportedOperationException e) {
        assertEquals("Operation Not Supported", e.getMessage());
    }
}
```

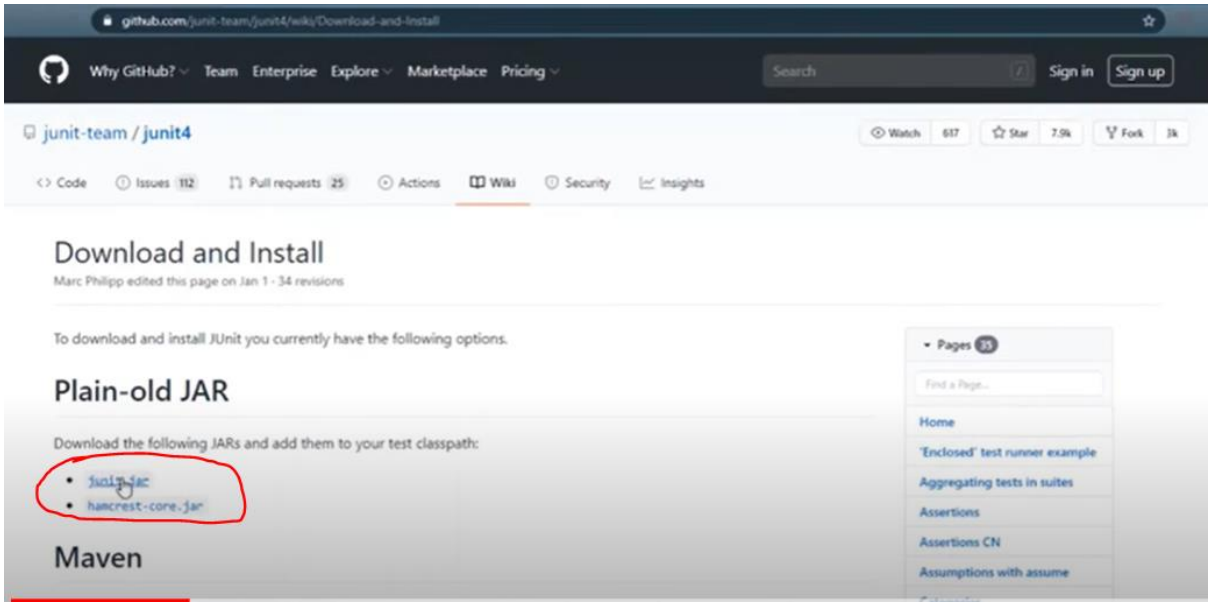
Solved Lab Activites

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class TestAssertions {

   @Test
   public void testAssertions() {
      //test data
      String str1 = new String ("abc");
      String str2 = new String ("abc");
      String str3 = null;
      String str4 = "abc";
      String str5 = "abc";

      int val1 = 5;
      int val2 = 6;

      String[] expectedArray = {"one", "two", "three"};
      String[] resultArray =  {"one", "two", "three"};

      //Check that two objects are equal
      assertEquals(str1, str2);

      //Check that a condition is true
      assertTrue (val1 < val2);

      //Check that a condition is false
      assertFalse(val1 > val2);

      //Check that an object isn't null
      assertNotNull(str1);

      //Check that an object is null
      assertNull(str3);

      //Check if two object references point to the same object
      assertSame(str4,str5);

      //Check if two object references not point to the same object
      assertNotSame(str1,str3);

      //Check whether two arrays are equal to each other.
      assertArrayEquals(expectedArray, resultArray);
   }
}
```

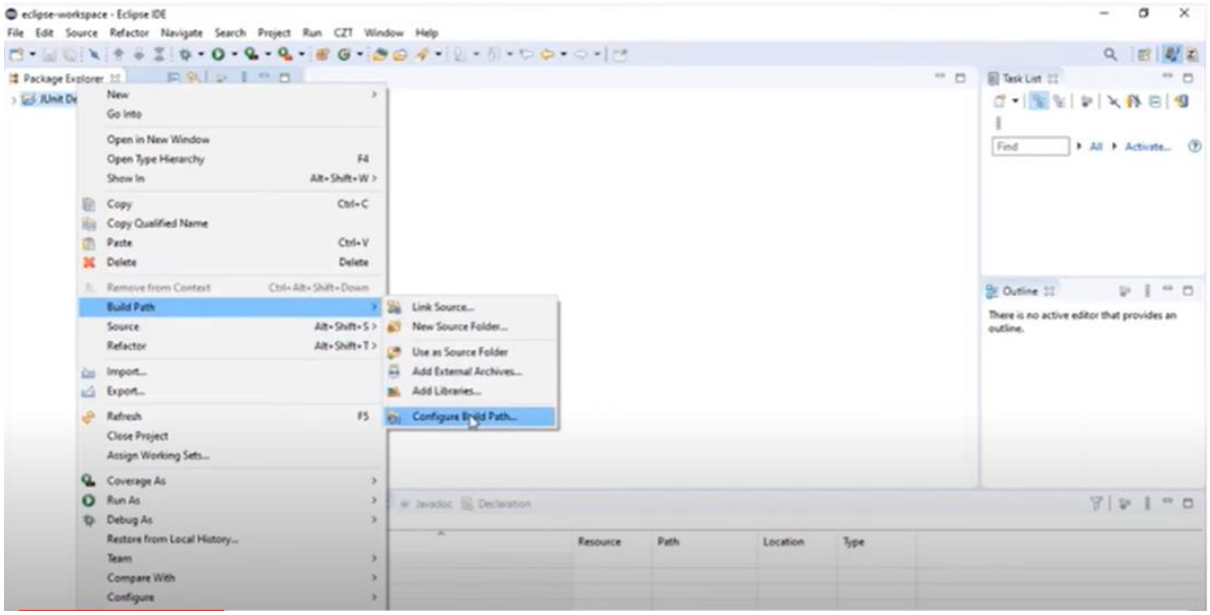
Configuration settings:

Go to Junit.org

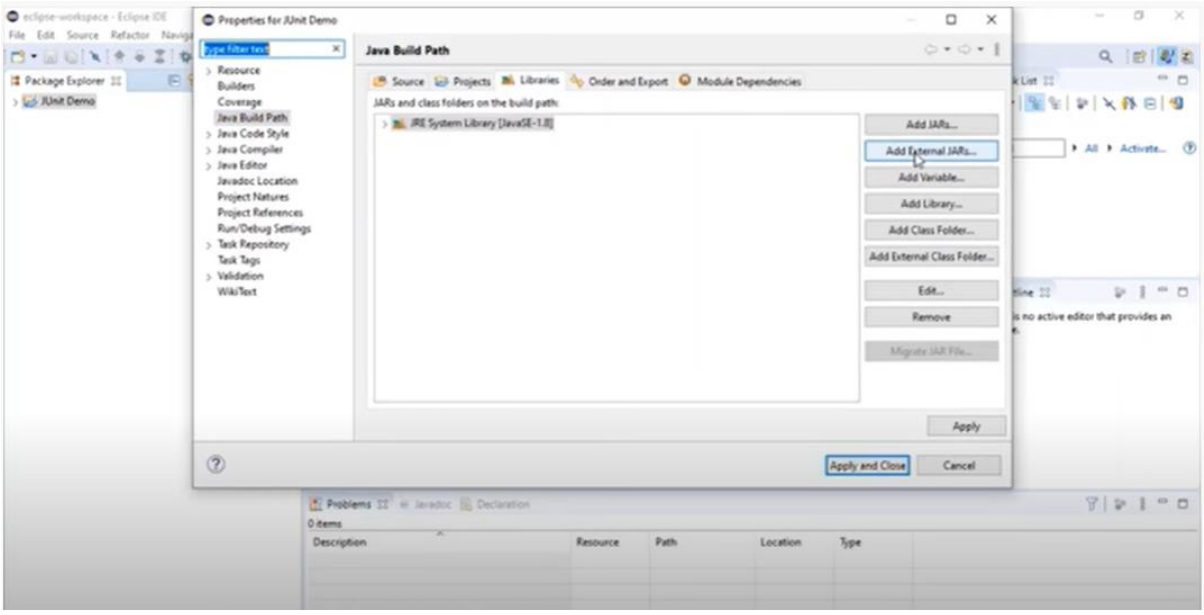
Download both these jar files



- First create Java project for Junit setup right click on java project and go to configure build path in build path option.

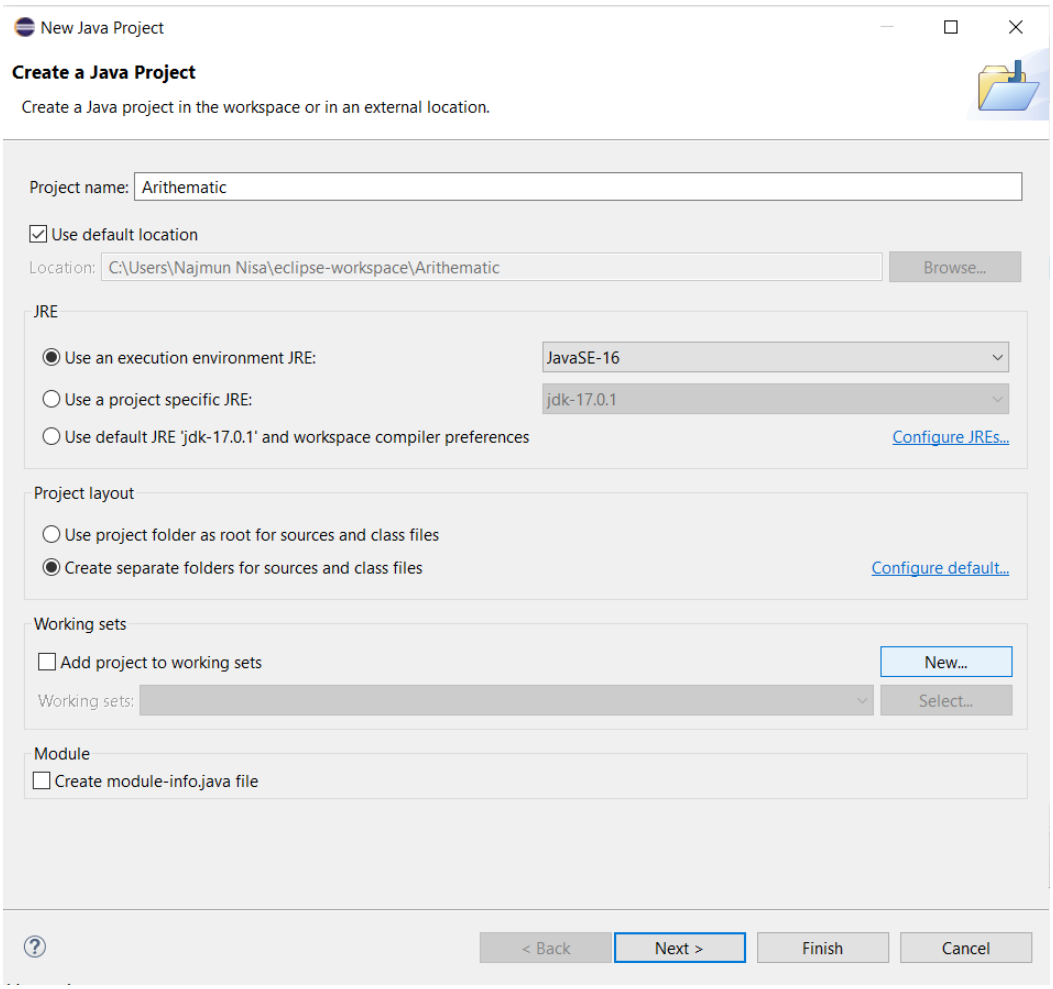


- Go to library→Add external jars and add those jars which you have downloaded earlier and click on apply and close. After these settings right click and refresh project .

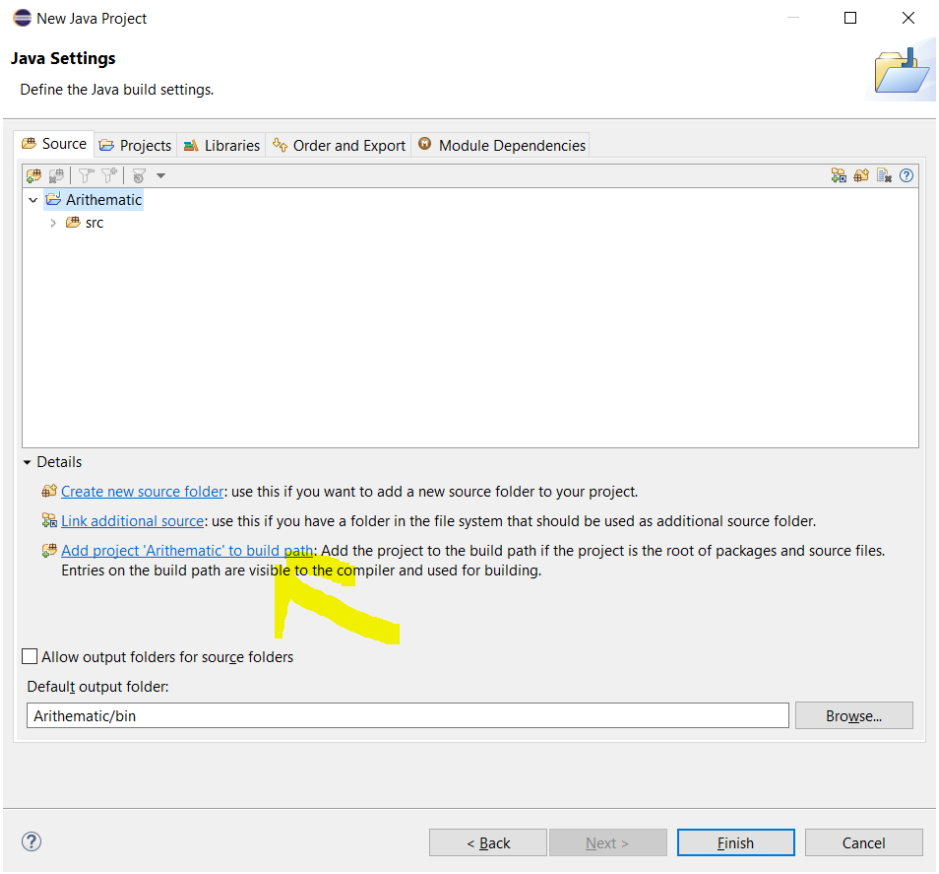


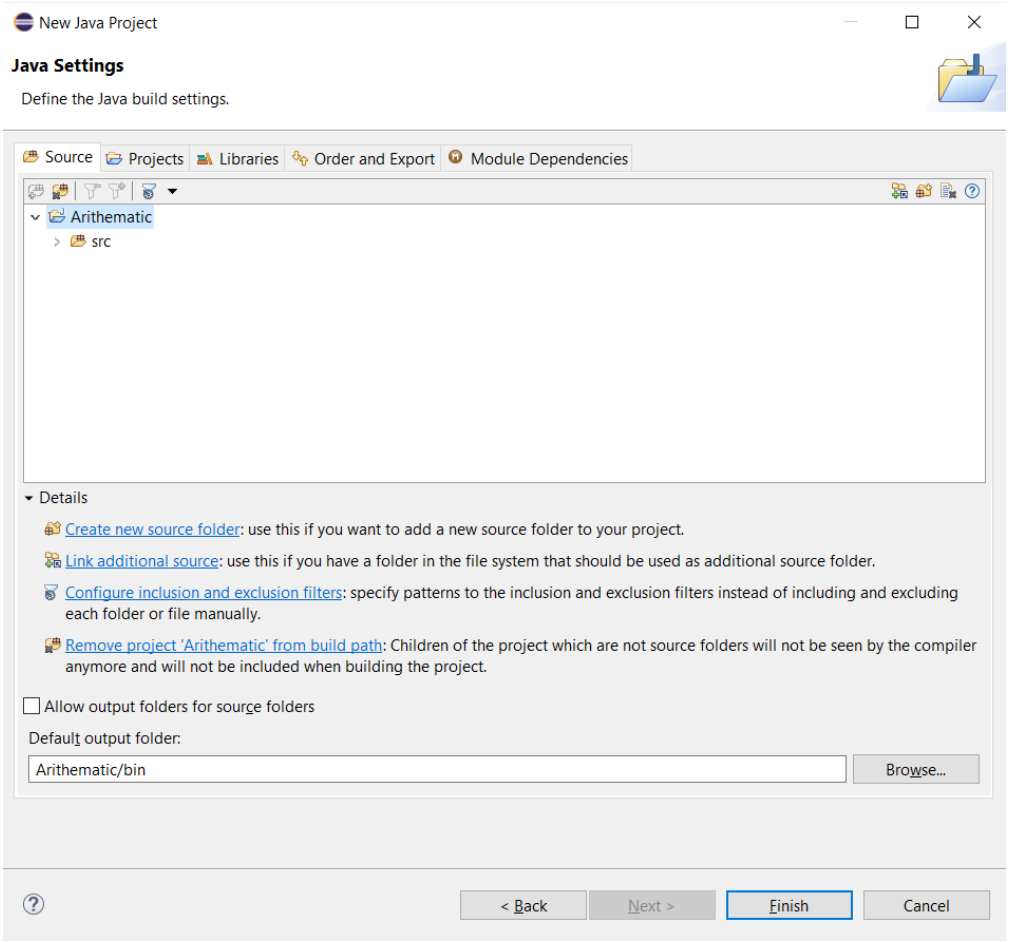
Step by step configuration in Eclipse:

- Create new Java Project, name project and click next

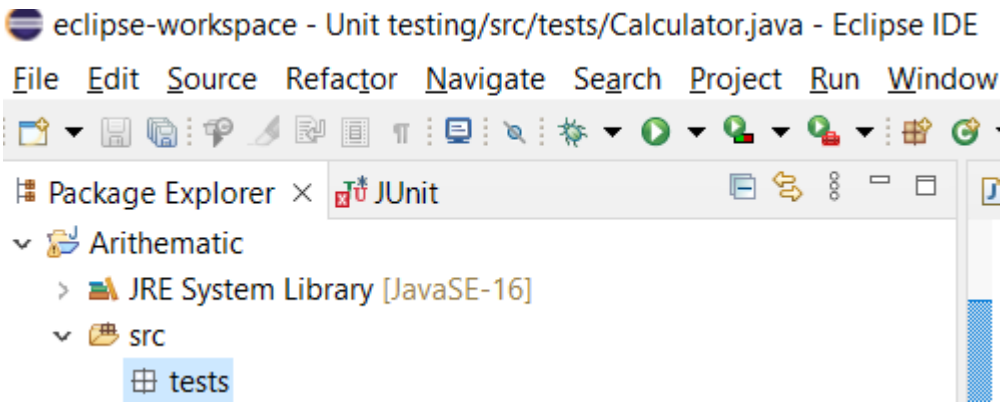


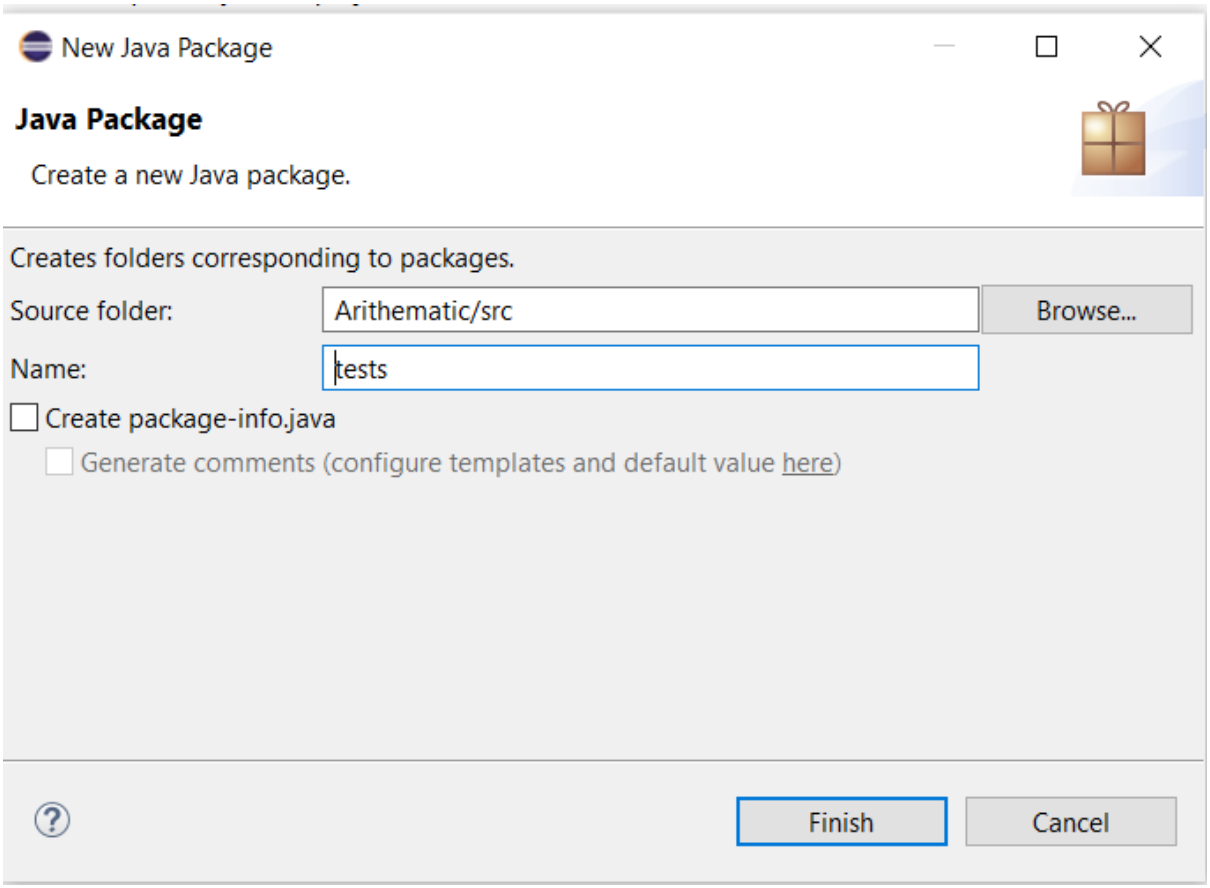
- Click on Add project to build path and click finish


New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
Module Dependencies
Arithematic
src
Details
Create new source folder: use this if you want to add a new source folder to your project.
Link additional source: use this if you have a folder in the file system that should be used as additional source folder.
Add project 'Arithematic' to build path: Add the project to the build path if the project is the root of packages and source files.
Entries on the build path are visible to the compiler and used for building.
Allow output folders for source folders
Default output folder:
Arithematic/bin
Browse...
< Back
Next >
Finish
Cancel

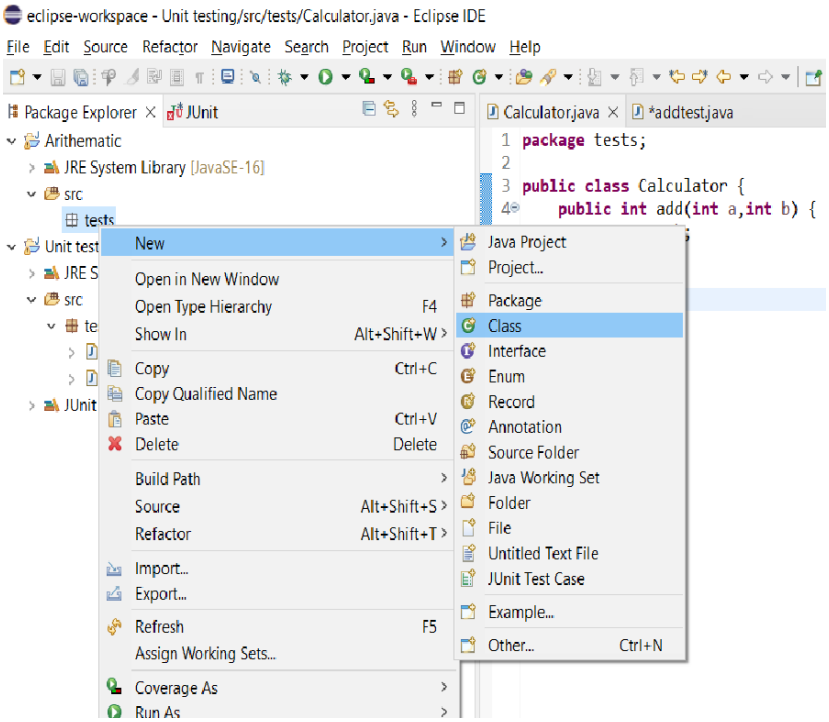


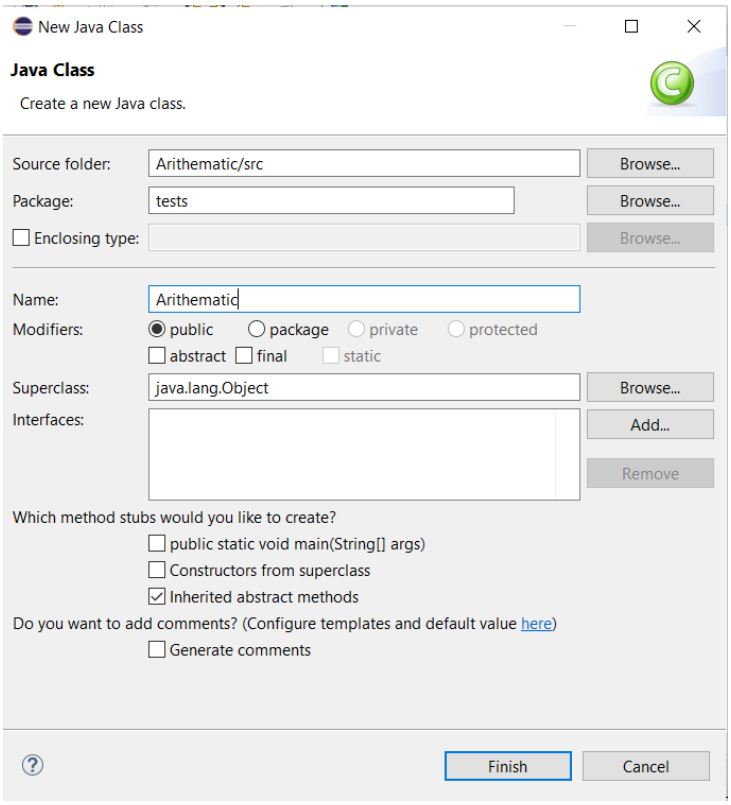
- On the left side under project name you have src(Source) folder . Right click on that and create java package and name it tests



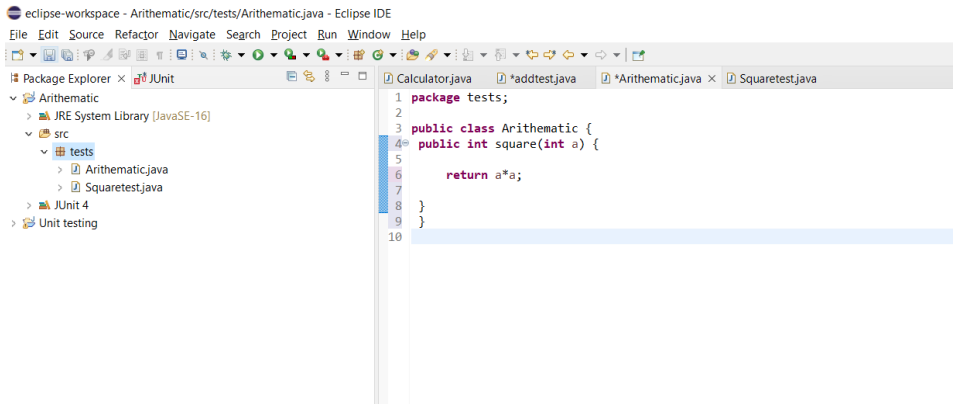


- In this package create a class in which you write some functions, name this class as arithematic.

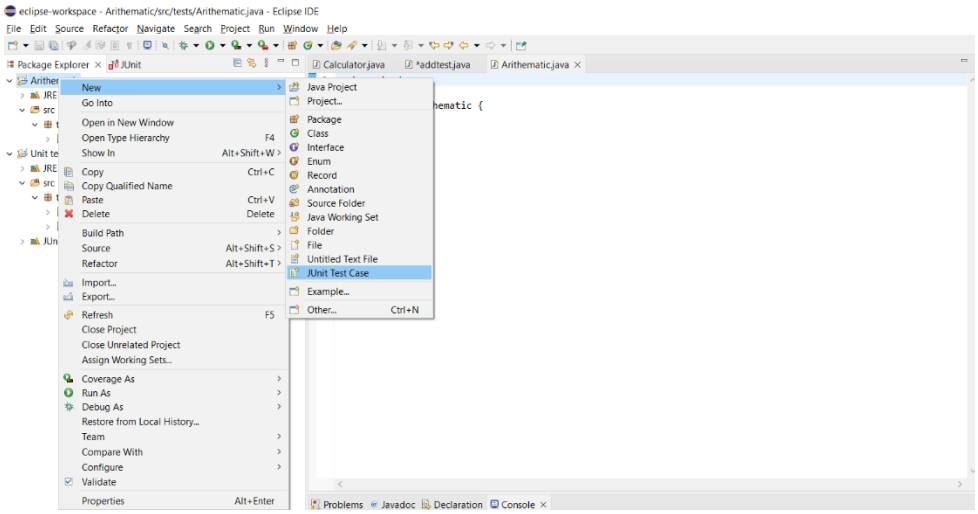

eclipse-workspace - Unit testing/src/tests/Calculator.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Arithematic
JRE System Library [JavaSE-16]
src
tests
Unit test
JRE S
src
JUnit
Calculator.java
*addtest.java
1 package tests;
3 public class Calculator {
4 public int add(int a,int b) {
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Coverage As
Run As
Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Example...
Other... Ctrl+N


New Java Class
Java Class
Create a new Java class.
Source folder: Arithematic/src
Browse...
Package: tests
Browse...
Enclosing type:
Browse...
Name: Arithematic
Modifiers: public package private protected
abstract final static
Superclass: java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

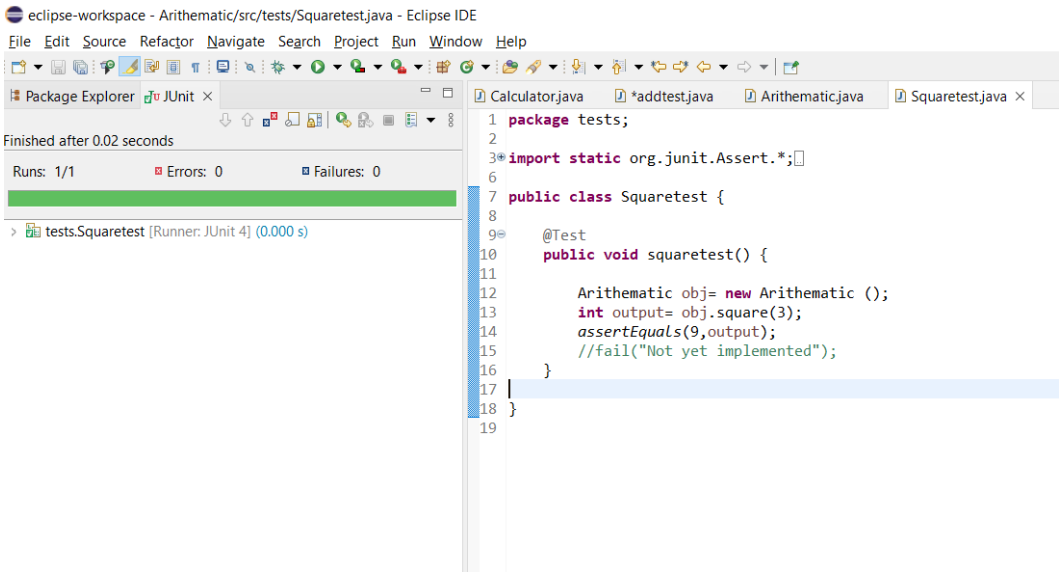
- Add some function in this class like I did square function in arithematic class.

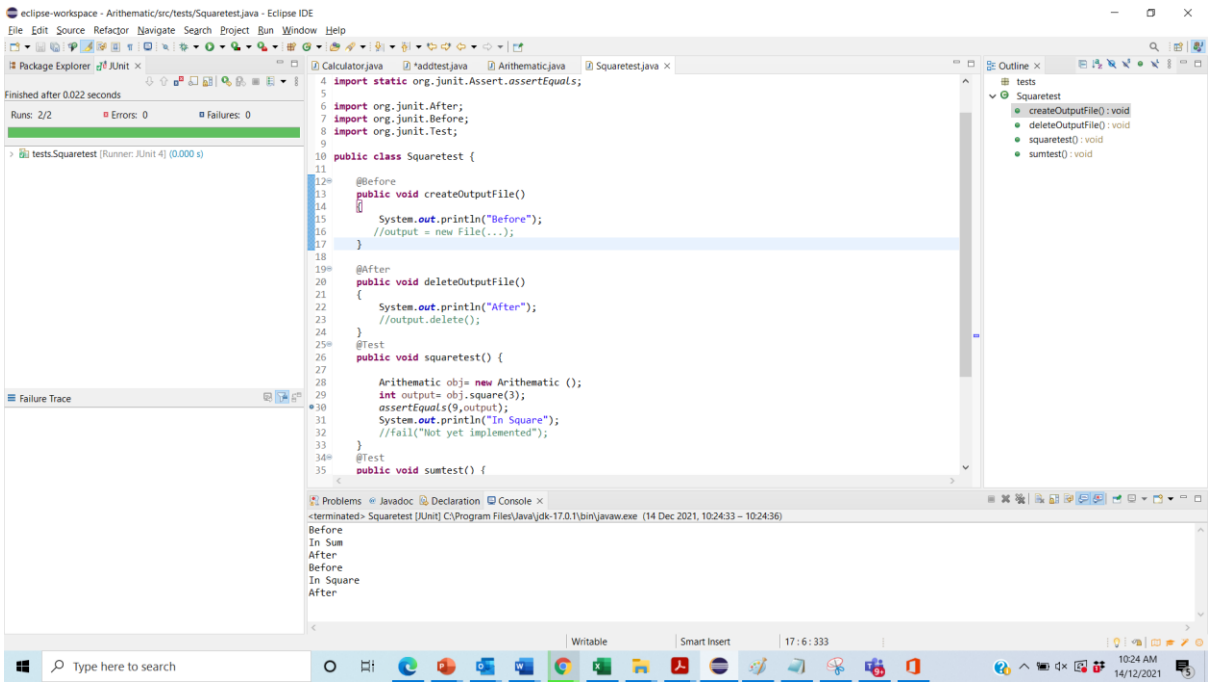


- Go on to the project name and create new JUnit test case here to test the functionality of this square function.



- So here is the first squaretest function with assertEqual method. Green bar on left side





Graded Lab Tasks

1. Write a program in JAVA to solve the triangle problem defined as follows:
   Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all.
   Write unit test cases for the traiangle class. Which will check the functionality of triangle class
2. Write calculator class having methods addition, multiplication , subtract, Square, Cube. Write testcases to test the functionality of calculator class.
3. Write Number class having function ispalindrom(), factorial(), number individual digits sum. Write test cases to test the functionality of Number class

Lab 08

JUNIT (Test Suites and TC Order Execution)

Objective:

- To understand the concept of test suites in JUnit and how they can be used to group and run multiple test cases together.
- To learn how to control the execution order of test cases using prioritization.
- To understand how to write data driven tests using parameterized test cases

Activity Outcomes:

- Ability to create and execute test suites in JUnit.
- Proficiency in ordering test cases using JUnit's @TestMethodOrder and @Order annotations.
- Understanding of the importance of test case order in certain testing scenarios.
- Ability to execute parameterized test cases

Useful Concepts with solved lab acitivites

Test Suite:

In JUnit 5, a test suite is a way to combine multiple test classes and run them together as a single unit. The @Suite annotation on the test class is used to define a test suite. You can also include specific test classes in the suite using annotations like @SelectClasses, @SelectPackages, or @SelectClasspathRoots, that help to filter out the test classes you want to run together.

Activity 1 :Test Suite with example:

Main:

```
public class DemoTesting {
	public int square(int value) {
		return value*value;
	}
	public int countA(String word) {
		int count=0;
		for(int i=0;i<word.length();i++) {
```

```
                if(word.charAt(i)== 'a' || word.charAt(i)== 'A') {
                      count++;              }
                }     return count;
      }
}
```

Test Case Class 1:

```
Class CountAtest {
      @BeforeEach
      void test() {
            DemoTesting test= new DemoTesting();
            int output=test.square(2);
            assertEquals(4, output);
      }
      @Test
      void test2() {
            DemoTesting test= new DemoTesting();
            int output=test.countA("abca");
            assertEquals(2, output);
      }
      @Test
      void test3() {
            DemoTesting test= new DemoTesting();
            int output=test.countA("abca");
            assertEquals(2, output);
      }
```

Test Case Class 2:

```
public class squareTest {
      @Test
      public void test() {
            DemoTesting test= new DemoTesting();
            int output=test.square(2);
            assertEquals(4,output);
      }
```

```
}
```

Test suite:

```
@RunWith(Suite.class)
@SuiteClasses({ squareTest.class ,CountAtest.class })
public class AllTests {
}
```

Test Execution Order

By default, test methods will be ordered using an algorithm that is **deterministic** but intentionally **nonobvious**. This ensures that subsequent runs of a test suite execute test methods in the same order, thereby allowing for repeatable builds. [Ref].

To control the order in which test methods are executed, annotate your test class or test interface with @TestMethodOrder and specify the desired **MethodOrderer** implementation. You can implement your own custom MethodOrderer or use one of the following built-in MethodOrderer implementations.

- ***Alphanumeric***: sorts test methods alphanumerically based on their names and formal parameter lists.
- ***OrderAnnotation***: sorts test methods numerically based on values specified via the @Order annotation.
- ***Random***: orders test methods pseudo-randomly and supports configuration of a custom seed.

Activity 2: Example

```
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

@TestMethodOrder(OrderAnnotation.class)

class OrderedTestsDemo {

    @Test

    @Order(1)

    void nullValues() {

        // perform assertions against null values

   }

    @Test

    @Order(2)
```

```
    void emptyValues() {

        // perform assertions against empty values

    }

    @Test

    @Order(3)

    void validValues() {

        // perform assertions against valid values

    }

}
```

Repeated Tests

JUnit Jupiter provides the ability to repeat a test a specified number of times by annotating a method with @RepeatedTest and specifying the total number of repetitions desired. Each invocation of a repeated test behaves like the execution of a regular @Test method with full support for the same lifecycle callbacks and extensions.

Example

```
@RepeatedTest(10)

void repeatedTest() {// ...}
```

Parameterized Tests

Parameterized tests make it possible to run a test multiple times with different arguments. They are declared just like regular @Test methods but use the @ParameterizedTest annotation instead. In addition, you must declare **at least one source** that will provide the arguments for each invocation and then consume the arguments in the test method. [Ref].

Activity 3: Example

```
@ParameterizedTest

@ValueSource(strings = { "racecar", "radar", "able was I ere I saw elba" })

void palindromes(String candidate) {

    assertTrue(StringUtils.isPalindrome(candidate));

}
```

Graded Lab Tasks

1. Create a class named Sandwich that stores information about a single sandwich. It should contain following:
   - Private instance variables to store the name of sandwich(chicken, egg) size of the sandwich (either small, medium, or large), the number of cheese toppings, the number of pepperoni toppings.
   - A public method named TotalCost( ) that returns a double that is the cost of the pizza.

     Chicken sandwich cost is determined by:

     Small: $10 + $2 per topping

     Medium: $15 + $3 per topping

     Large: $20 + $4 per topping

     Egg sandwich cost is determined by:

     Small: $5 + $1 per topping

     Medium: $8 + $1 per topping

     Large: $10 + $1 per topping

   - A public method named getDescription( ) that returns a String containing the name of sandwich size, quantity of each topping, and the sandwich cost as calculated by TotalCost( ).
   - Write JUnit test cases to check the functionality of Sandwich class with following methods assertion assertNotSame, assertEquals, assertNULL and annotation methods @Test , @Before, @BeforeAll, @AfterAll
   - Provide Parameterized test cases to test the functionality of the above class.

2. Provide parameterized test cases for the calculator class.

Lab 10

Selenium IDE

Objective:

The objective of this lab is to guide students through the basics of using Selenium IDE for automated testing of web applications. By the end of this lab, students will be able to:

- Understand the core concepts of Selenium IDE.
- Record, edit, and execute test scripts.
- Use Selenium commands and assertions to enhance tests.
- Debug and troubleshoot test scripts.
- Export test cases for further use in Selenium WebDriver.

Activity Outcomes:

By completing the activities in this manual, students will:

1. Gain familiarity with Selenium IDE's interface and capabilities.
2. Learn to record and playback automated test scripts.
3. Enhance test scripts using commands and assertions.
4. Develop skills in debugging and troubleshooting test scripts.
5. Export test scripts for use with advanced testing frameworks like Selenium WebDriver.

Useful Concepts

1. Selenium IDE Basics

- **Selenium IDE**: A browser extension for recording and playing back web application tests.
- **Test Case**: A single automated test script that performs a sequence of actions.
- **Test Suite**: A collection of test cases grouped together.

2. Recording and Playing Back Tests

- **Record Button**: Starts recording user interactions with the web application.
- **Playback Button**: Executes the recorded test script.

3. Commands and Assertions

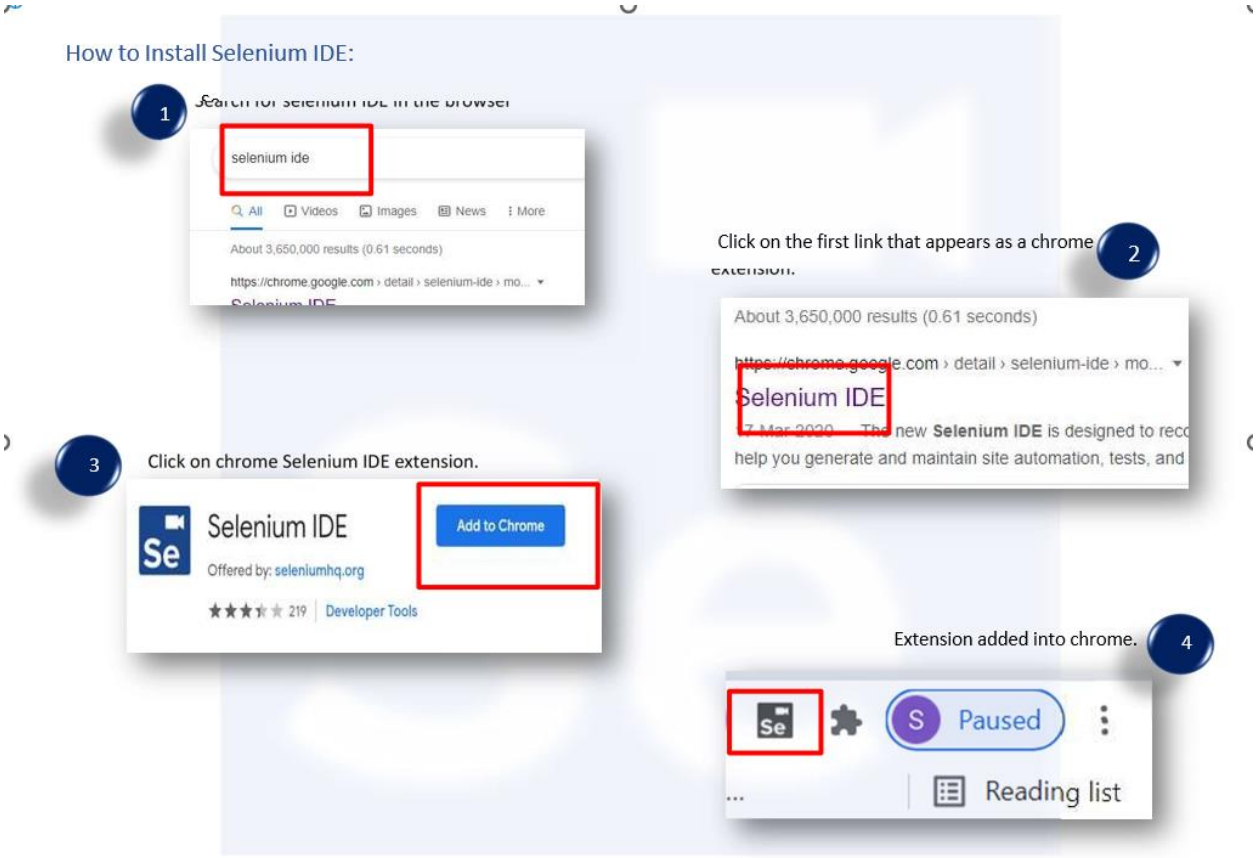
- **Command**: An instruction in the test script (e.g., `click`, `type`, `open`).
- **Assertion**: A validation step to verify the expected state of the application (e.g., `assertText`, `assertElementPresent`).

4. Debugging and Troubleshooting

- **Breakpoints**: Pauses test execution at a specific step to allow inspection.
- **Log Panel**: Displays execution logs and error messages for diagnosing issues.

5. Exporting Test Cases

- **Export Options**: Allows exporting recorded test cases to different formats and programming languages for use with Selenium WebDriver.



Solved Lab Activites:

Lab Activity 1:

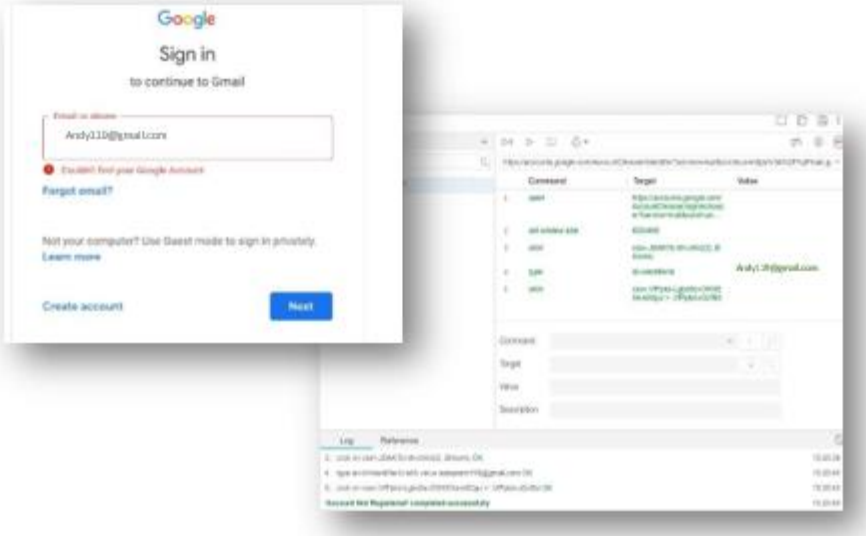
Commands Used to explore Features of Selenium IDE

Login:

Email Not yet Registered as Gmail Account

Recorded Commands:

Output:

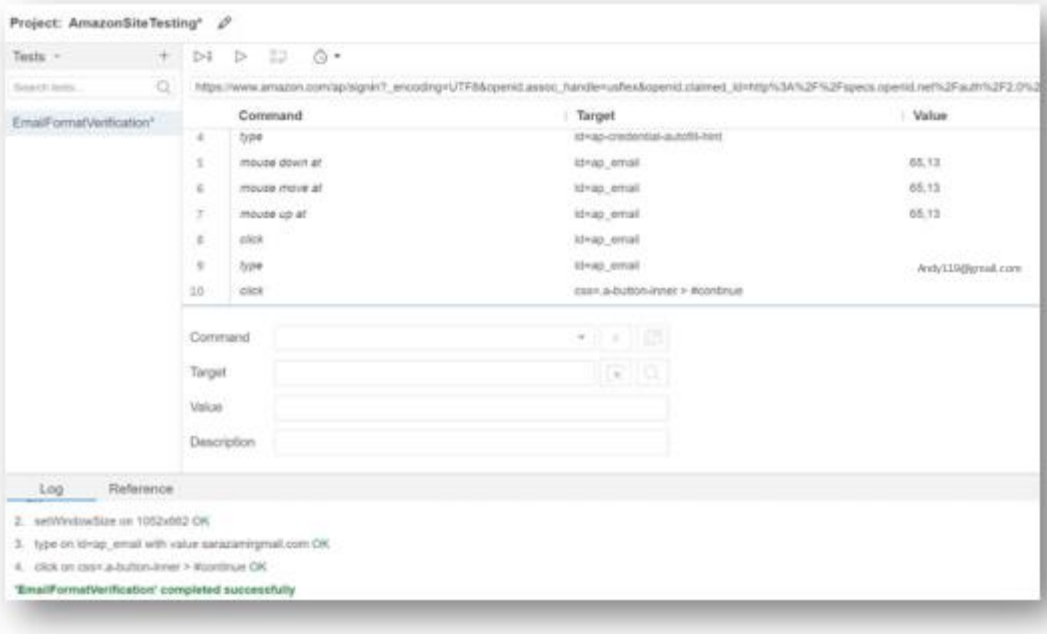


Lab Activity 2:

Email Format Verification

Output:

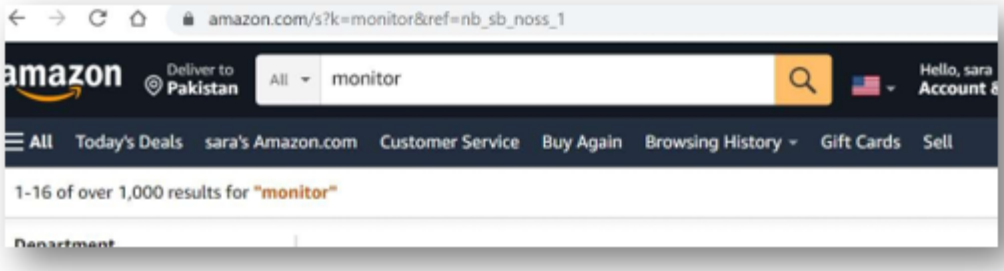
Test passed as entered email was with incorrect format.



Lab Activity 3:

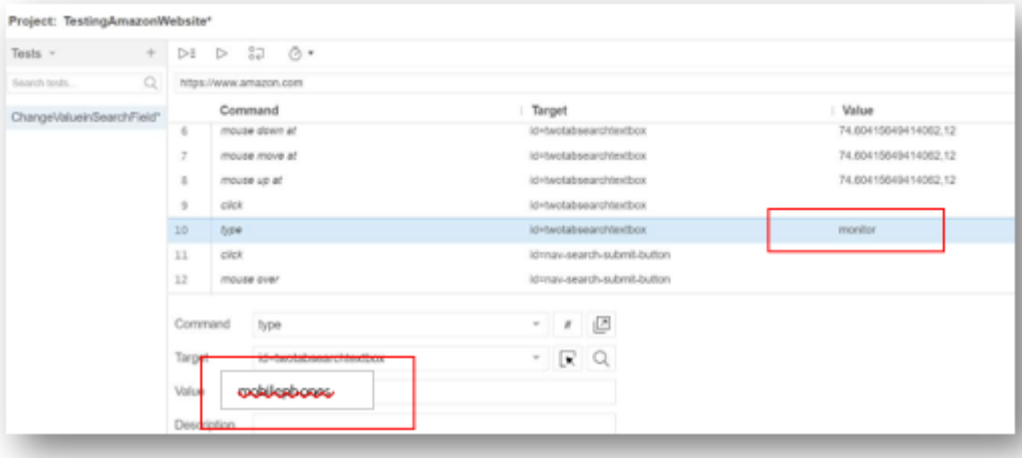
Search

Change Value in Search Field



Output:

Before Changing Value



After changing value

After changing value in selenium id, the test case is run, and it changes the value from "monitor" to "mobilephone" and gives results for it. Thus, our test executed successfully as our search is editable and can change values.

Lab Activity 4:

Search By Tag

In this search user search the item by clicking on the tag display to him/her. For example: in my case user want to search the item related to fashion and beauty the user searches the item by clicking on this tag.

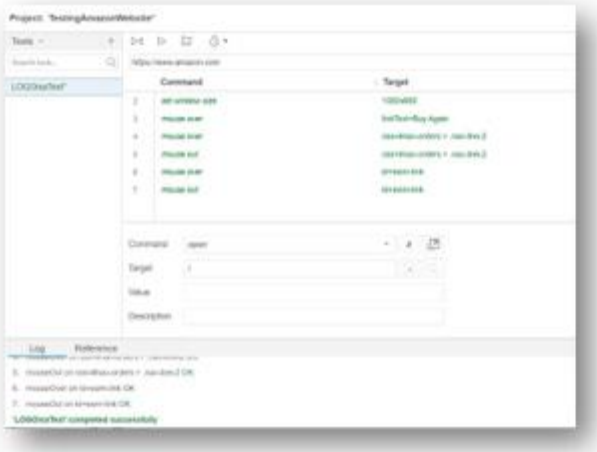
Logo

Logo(is it a text or not)

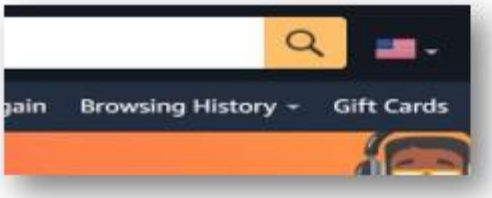
Check logo for any website if it is a text or not.

Output:

Amazon logo is a text which act as a link.



Link(editable or not?)



Output:

The link is not editable therefore the test failed.

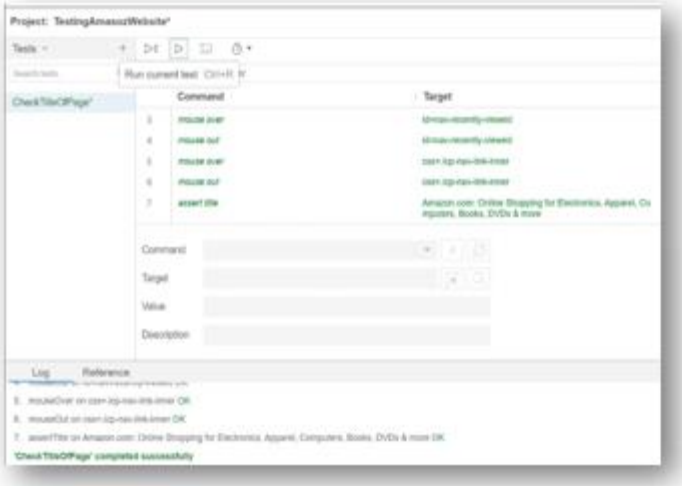
Lab Activity 5:

Page Title

Check Page Title of Page

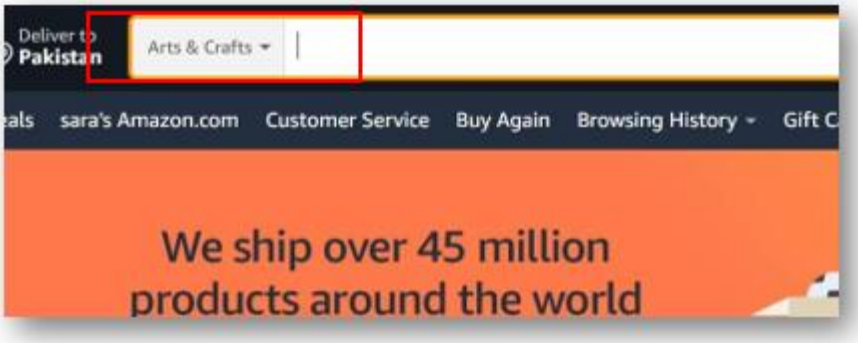
Output:

By clicking anywhere on webpage, user can check for the title of the page.



Change Label Value

Firstly, we recorded the test case with the label "Arts and craft". But changed label value to "book" before executing the test to check if we can change labels in the drop down or not.



Output:

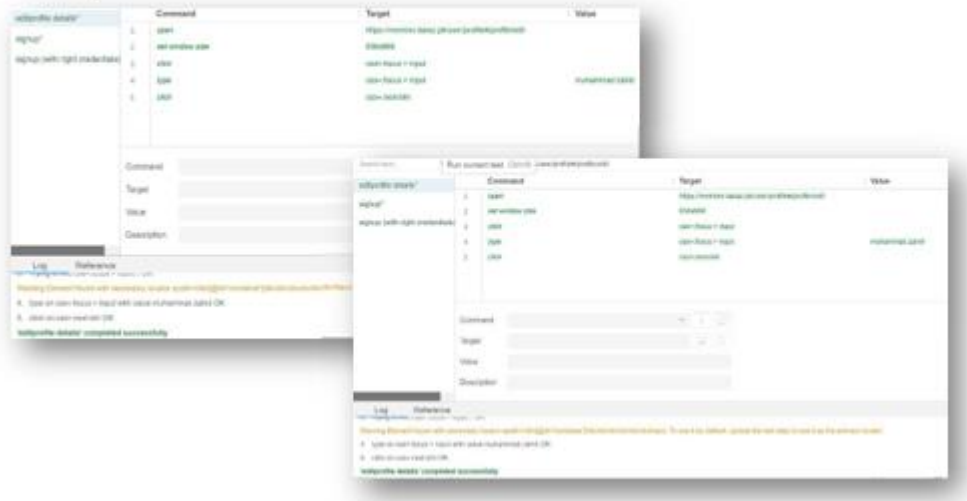
Labels are defined by the amazon so we can only select them in order to search our required services, but we are not able to change the value of label. That is why this test failed.

Lab Activity 6:

Profile

Edit profile details(after signup)

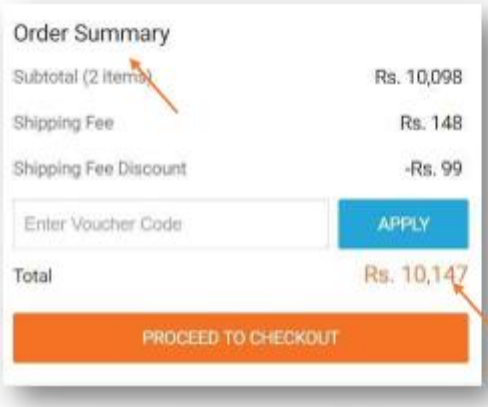
Edited the name "Muhammad zahid" to "Muhammad zamir" which was stored in daraz site and edited successfully.



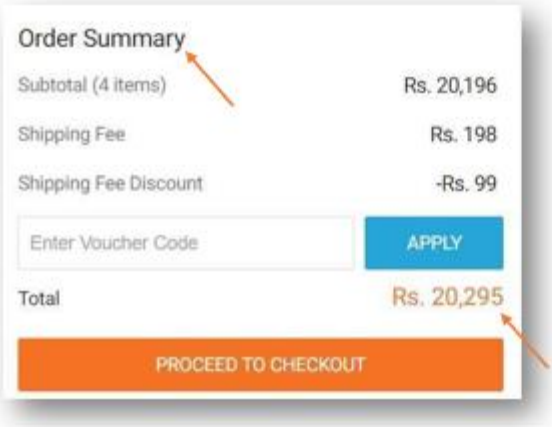
Checkout For product:

Total Bill After Addition Of Products

- Total before product addition



- Total after product addition



Output:

TEST SUITES

A Selenium Test Suite is a collection of Selenium test cases. It is a set of test cases that are part of a scenario or process that we want to get tested. A group of test cases may also contain prerequisite statusor steps, and descriptions of the following tests.

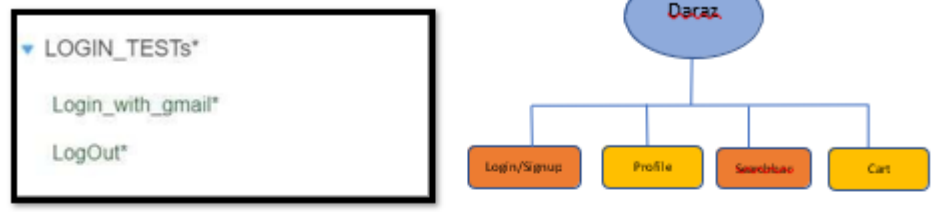
Running Selenium Test Suites

In the same way as with Selenium test cases, Selenium test suites are intended to be run.

Lab Activity 7:

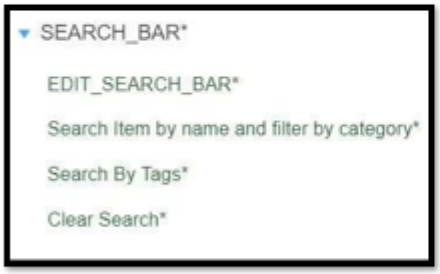
Example to demonstrate Test suites:

Login Test Suite



Profile Test Suite:

Search bar Test Suite:



Cart Test Suite:

Graded Lab Tasks

1. **Search for a Product**: Automate the search functionality on Daraz.com.
2. **Filter Search Results**: Apply filters to the search results.
3. **Verify Product Details**: Verify specific details on a product page.
4. **Add Product to Cart**: Add a product to the shopping cart and verify it.

Lab 11

Selenium WebDriver

Introduction:

WebDriver drives a browser natively, as a user would, either locally or on a remote machine using theSelenium server, marks a leap forward in terms of browser automation.

Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

If you want to create robust, browser-based regression automation suites and tests, scale and distribute scripts across many environments, then you want to use Selenium WebDriver, a collection of language specific bindings to drive a browser - the way it is meant to be driven. WebDriver uses browser automation APIs provided by browser vendors to control browser andrun tests. This is as if a real user is operating the browser. Since WebDriver does not require itsAPI to be compiled with application code, it is not intrusive. Hence, you are testing the same application which you push live.

How Selenium WebDriver Works:

- Selenium supports automation of all the major browsers in the market through the use of WebDriver. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behavior of web browsers. Each browser is backed by a specific WebDriver implementation, called a driver. The driver is the component responsible for delegating down to the browser, and handles communication to and from Selenium and the browser.

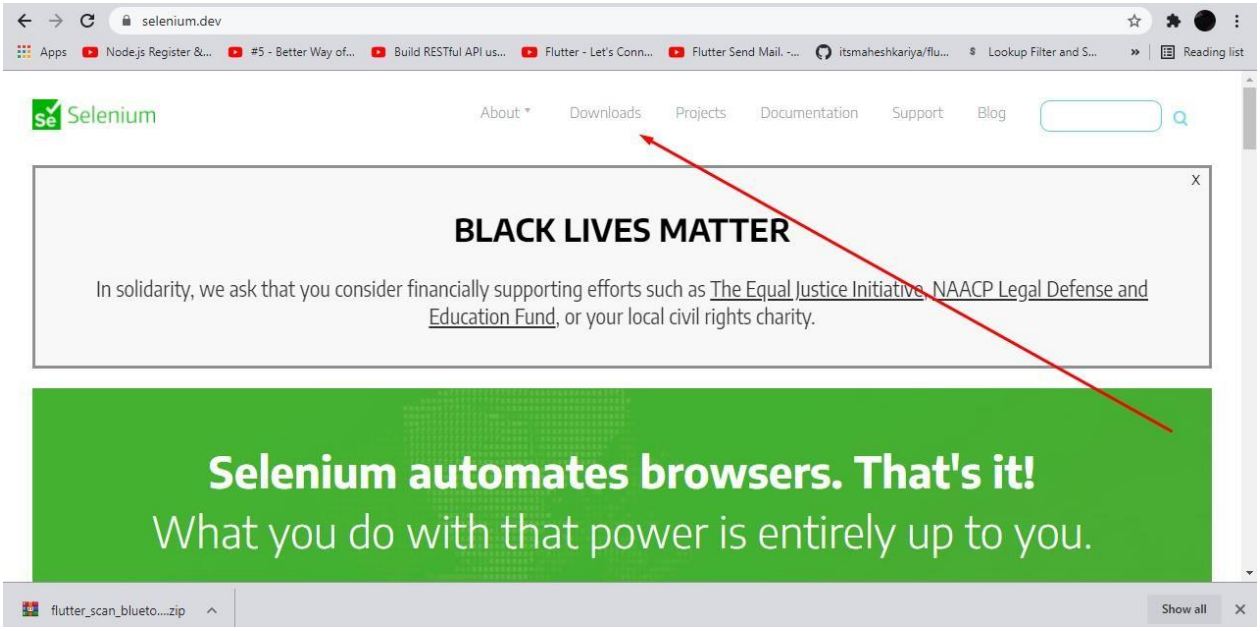
- This separation is part of a conscious effort to have browser vendors take responsibility for the implementation for their browsers. Selenium makes use of these third party drivers where possible, but also provides its own drivers maintained by the project for the cases when this is not a reality.

- The Selenium framework ties all of these pieces together through a user-facing interface that enables the different browser backends to be used transparently, enabling cross- browser and cross-platform automation.

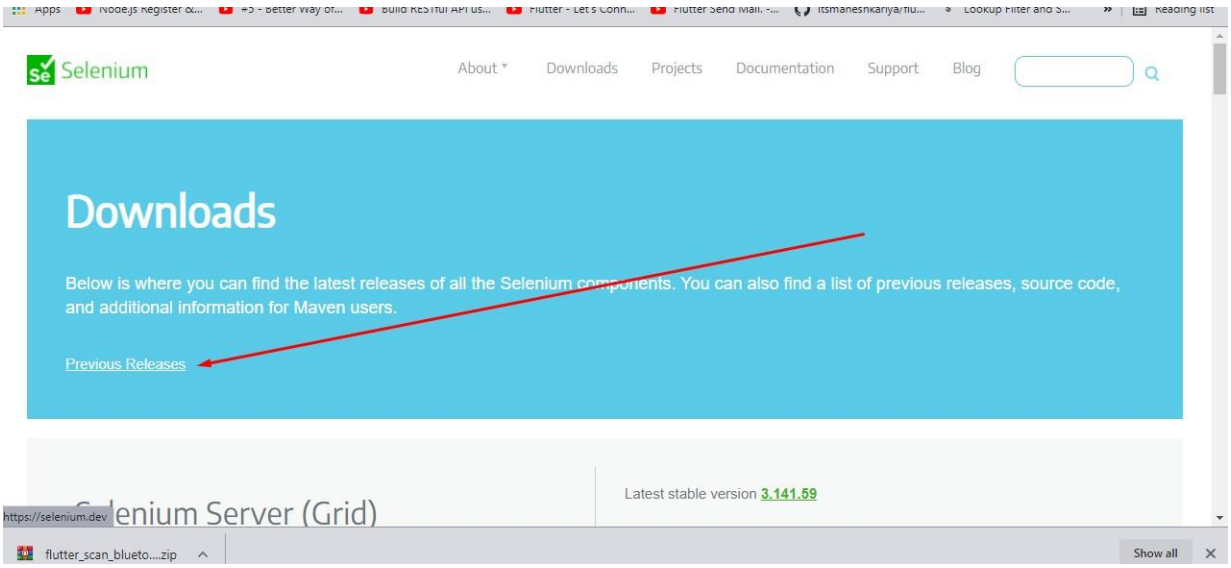
How to install Selenium Webdriver:

Configuration Steps:

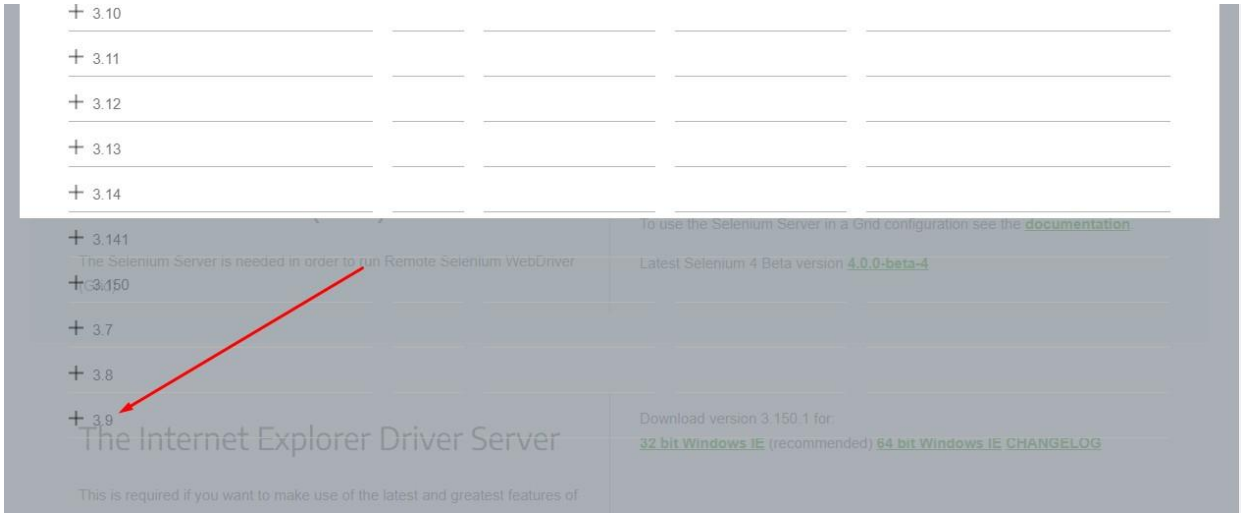
1: Go to selenium.dev



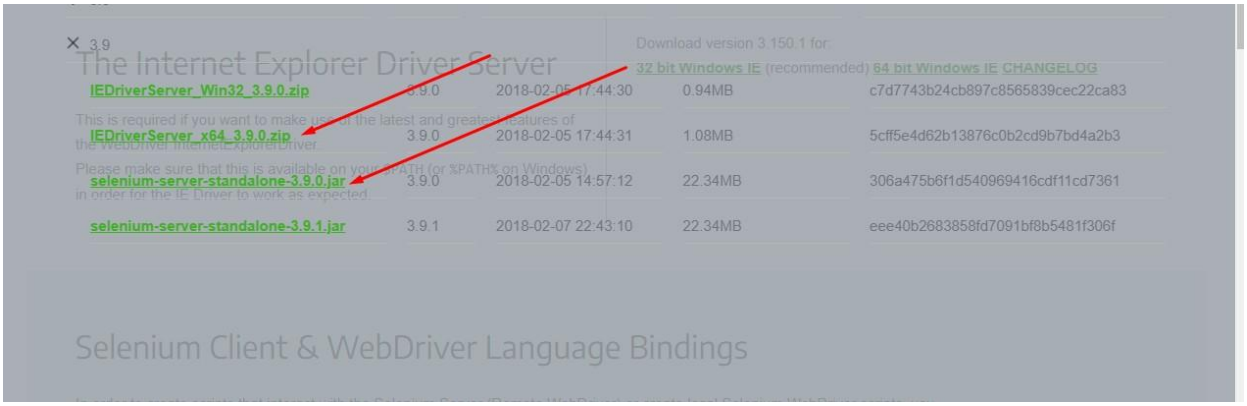
2: Navigate to Downloads -> Previous Releases -> HereButton



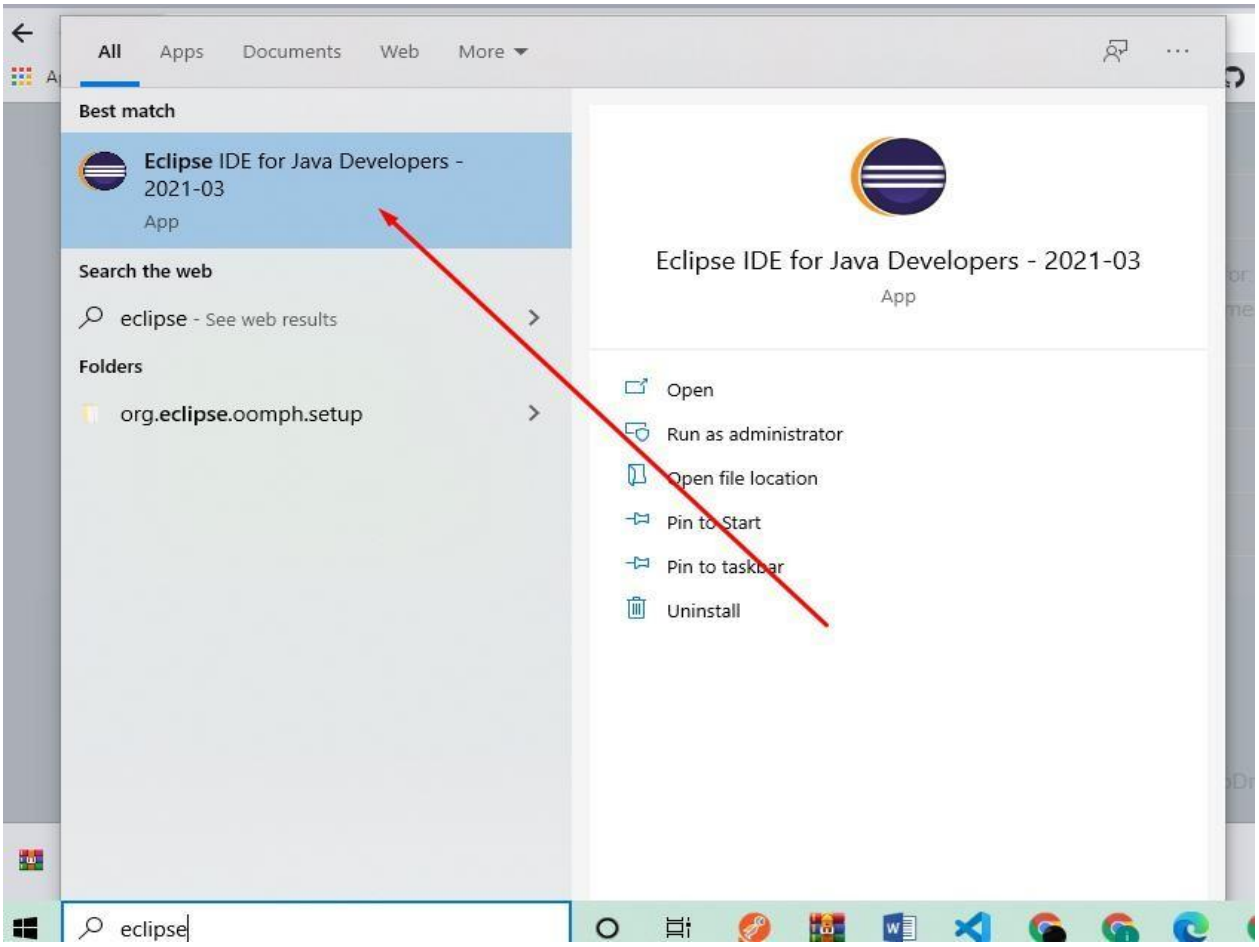
3: Choose Version 3.9



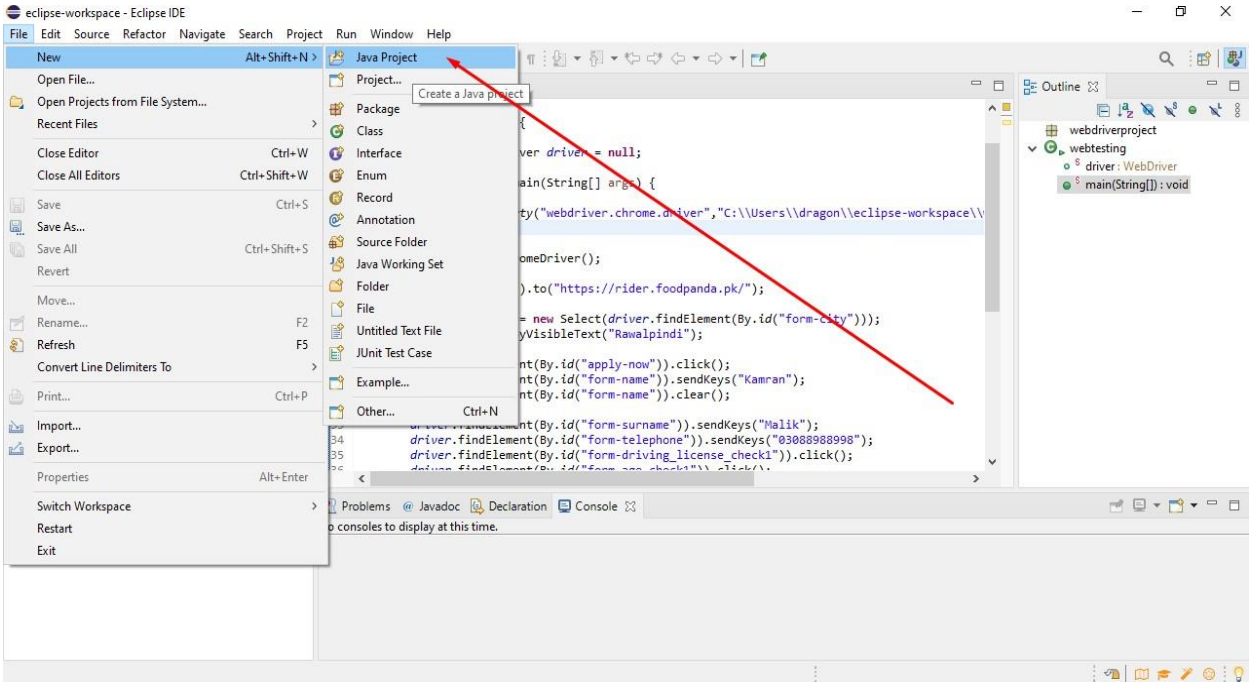
4: Download Selenium-Server-StandAlone.JAR Selenium-Server.ZIP



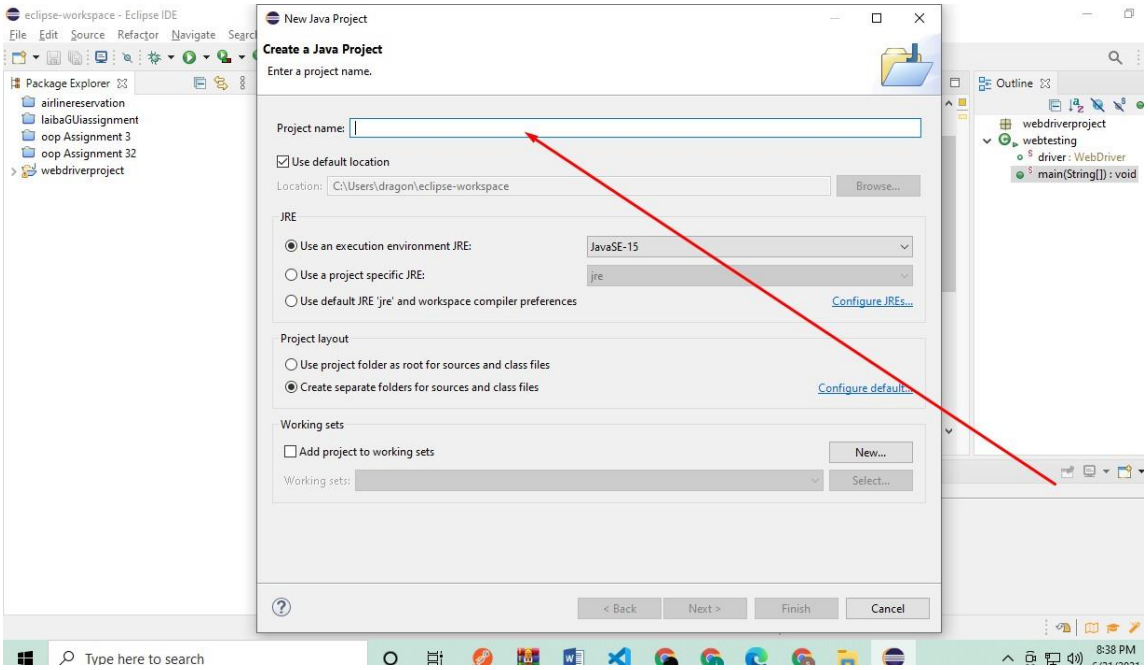
5: Open Eclipse:



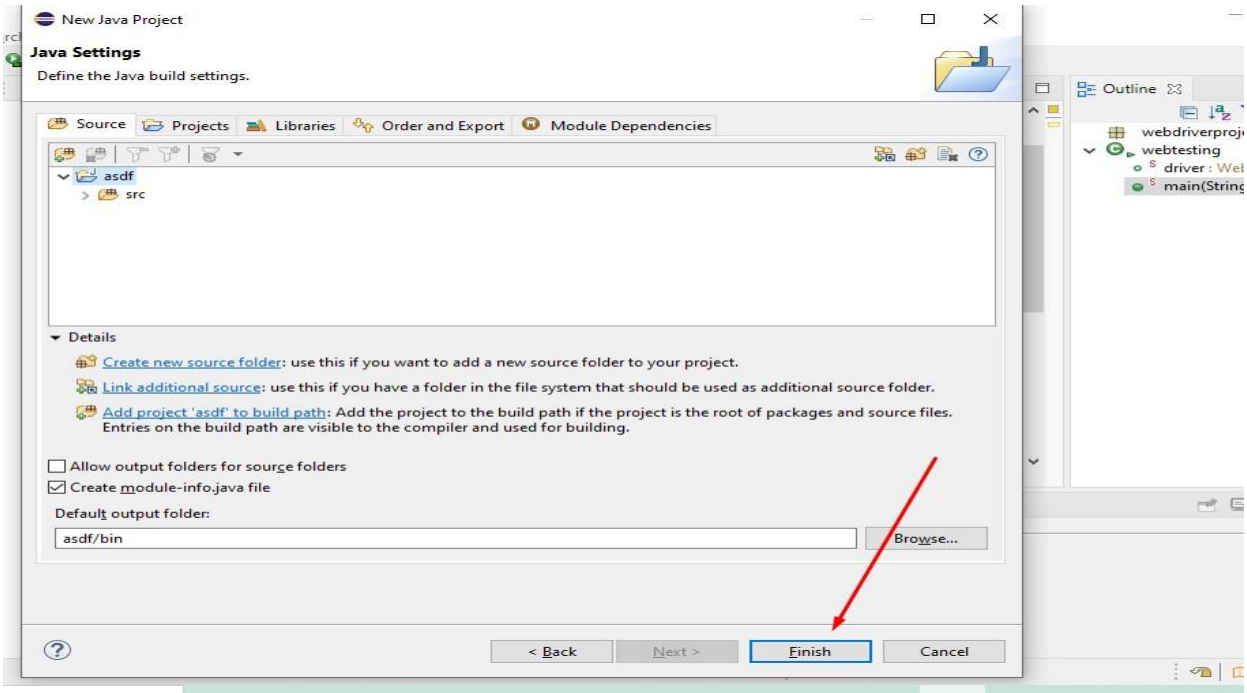
6: Create new Project



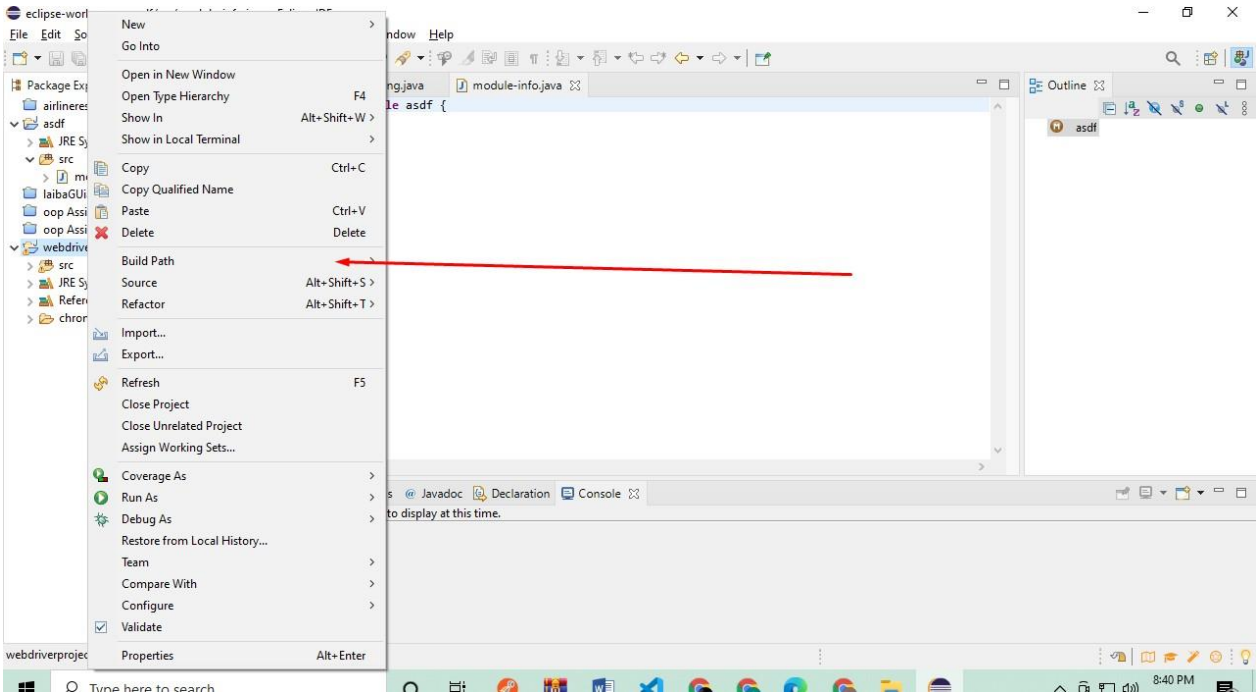
7: Enter name of Project



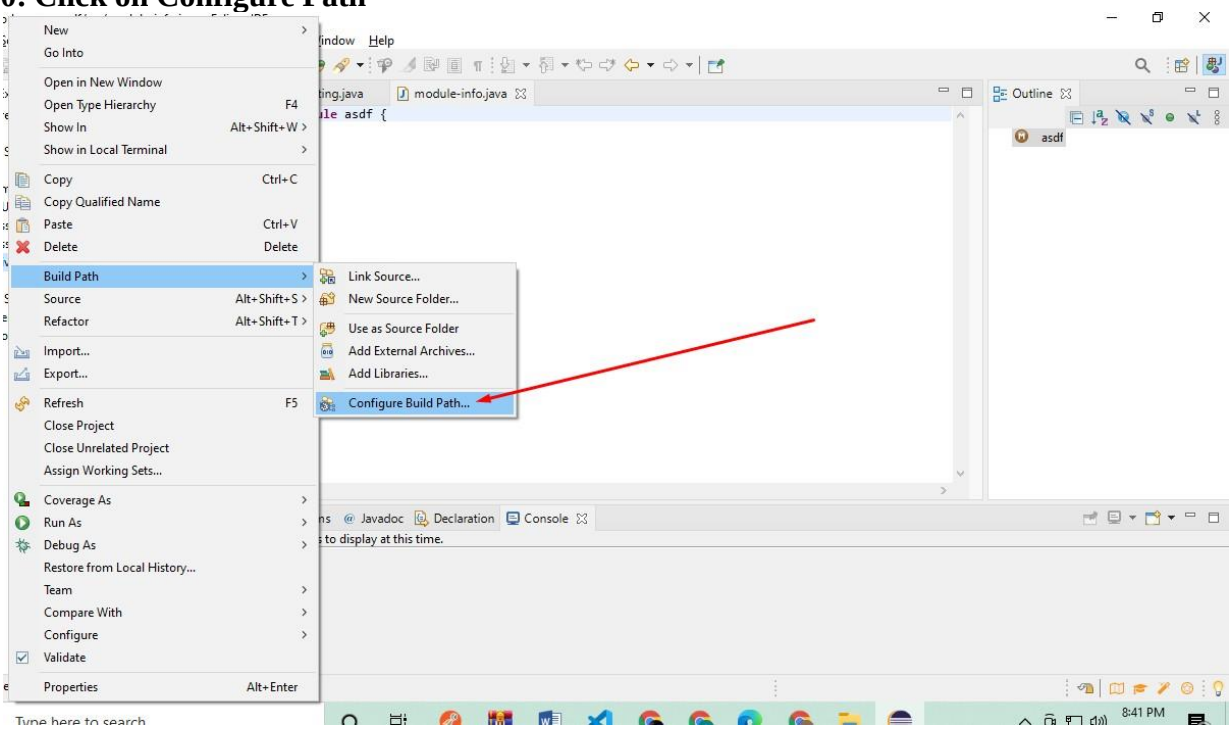
8: Finish



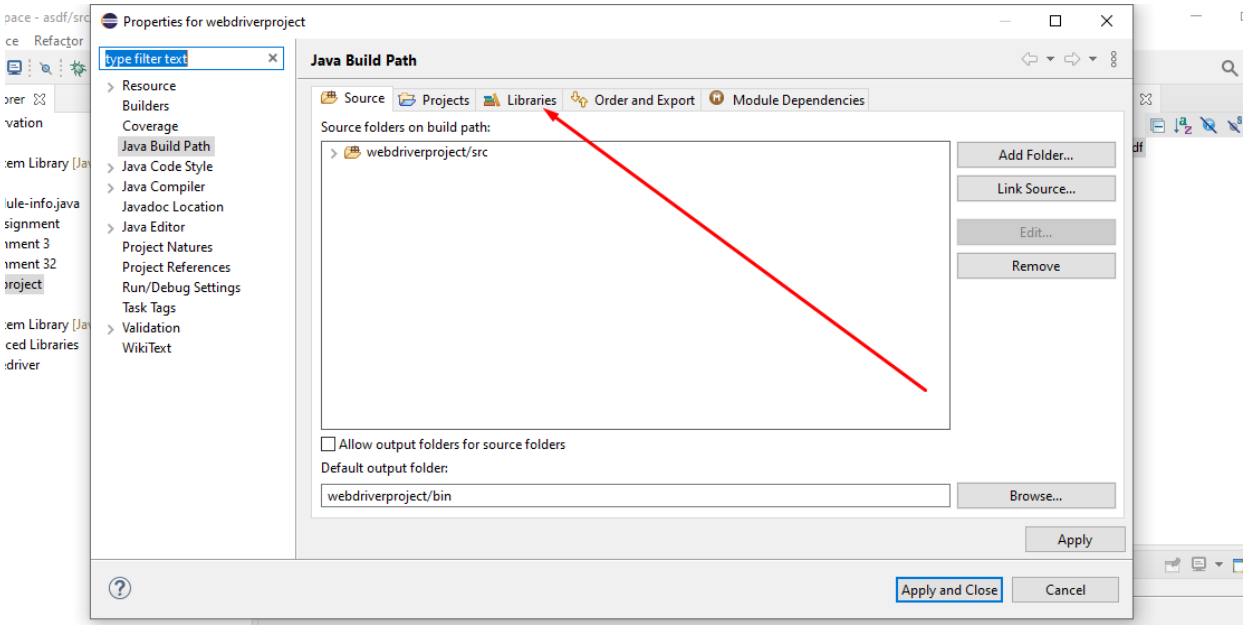
9: Right Click on project and click on Build Path



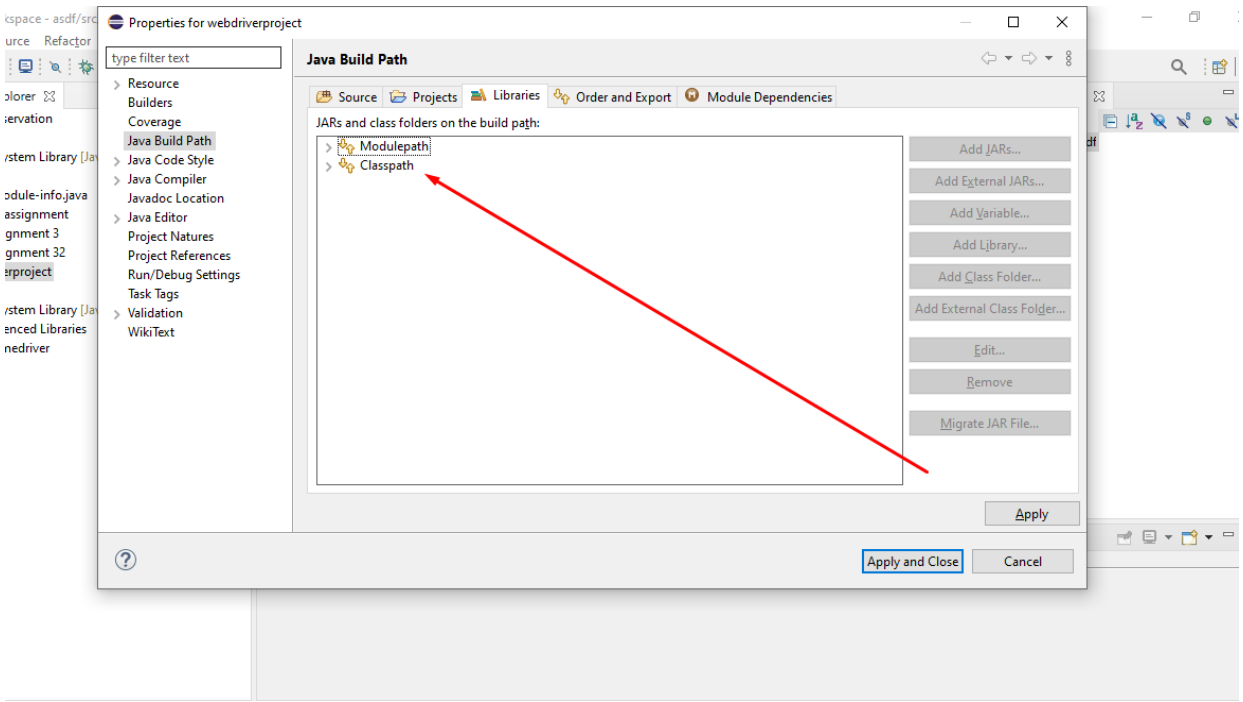
10: Click on Configure Path



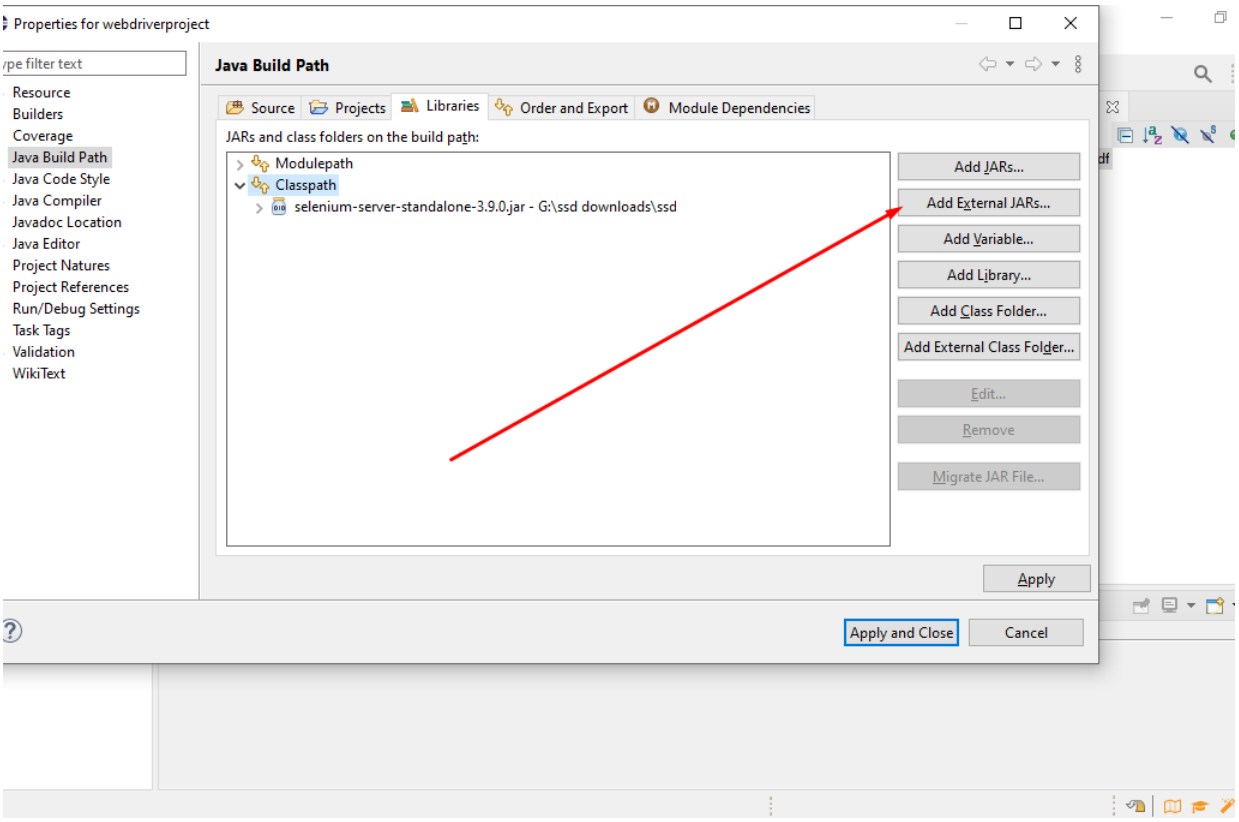
11: Go to Libraries



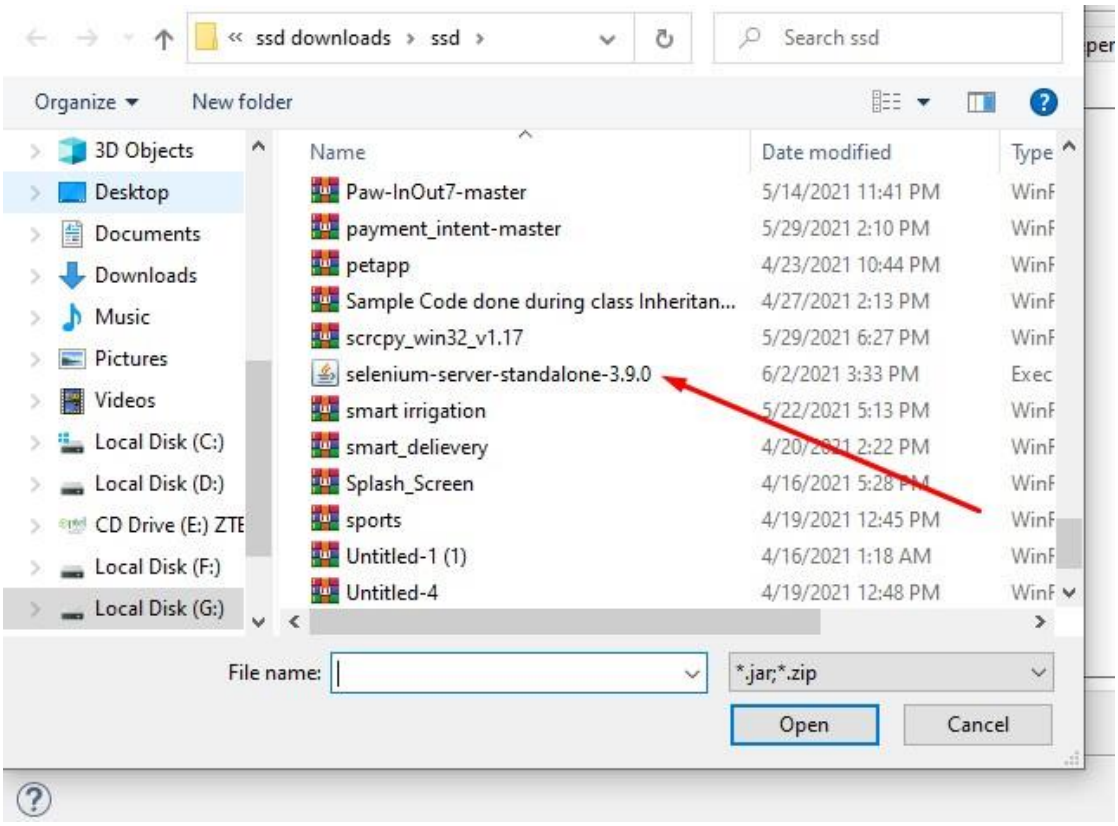
12: Open class path



13: Click on Add External Jar

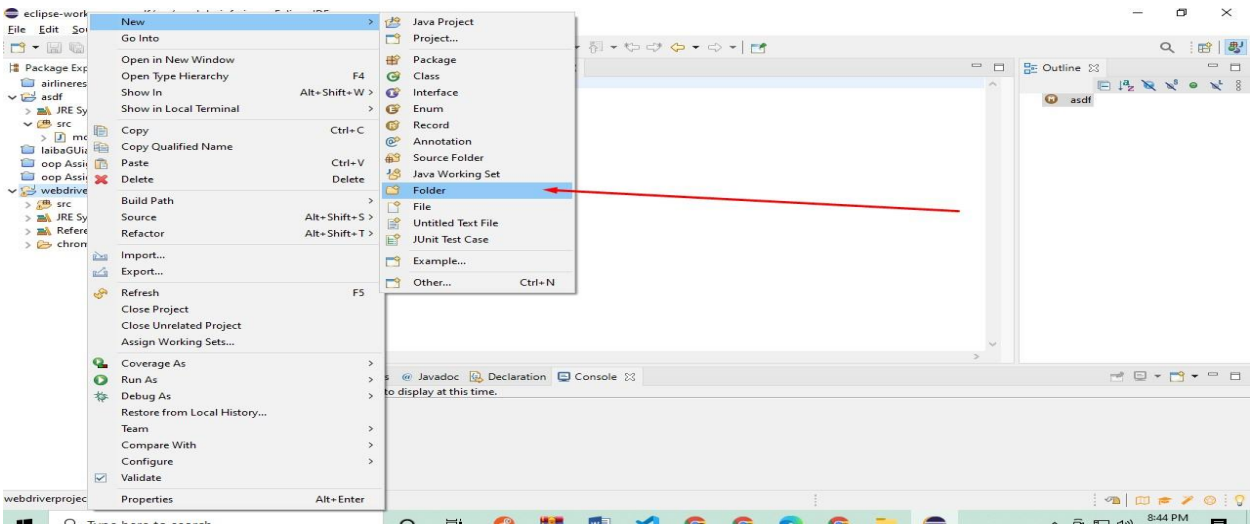


14: Go to downloads and select the file

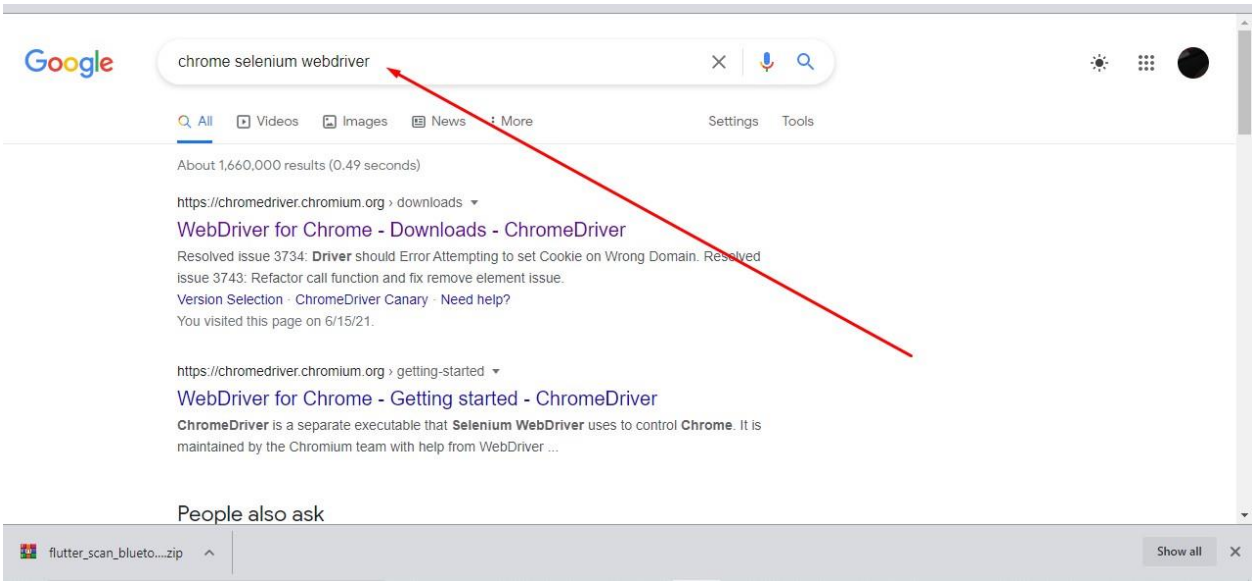


15: Right Click on project and create new folder named itchrome.

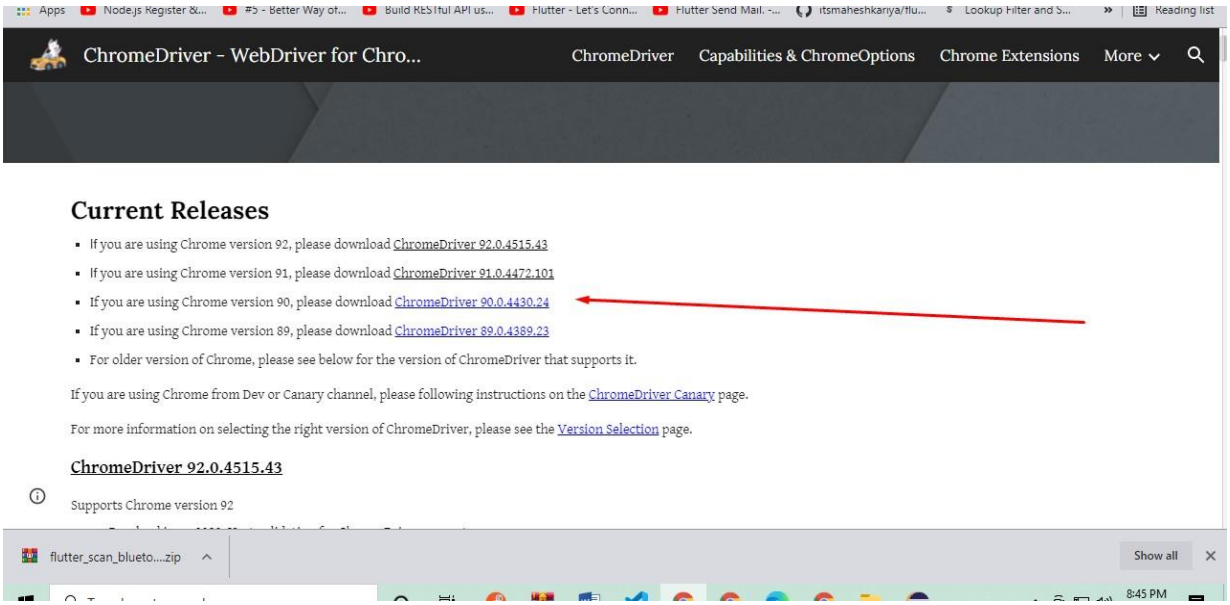
-



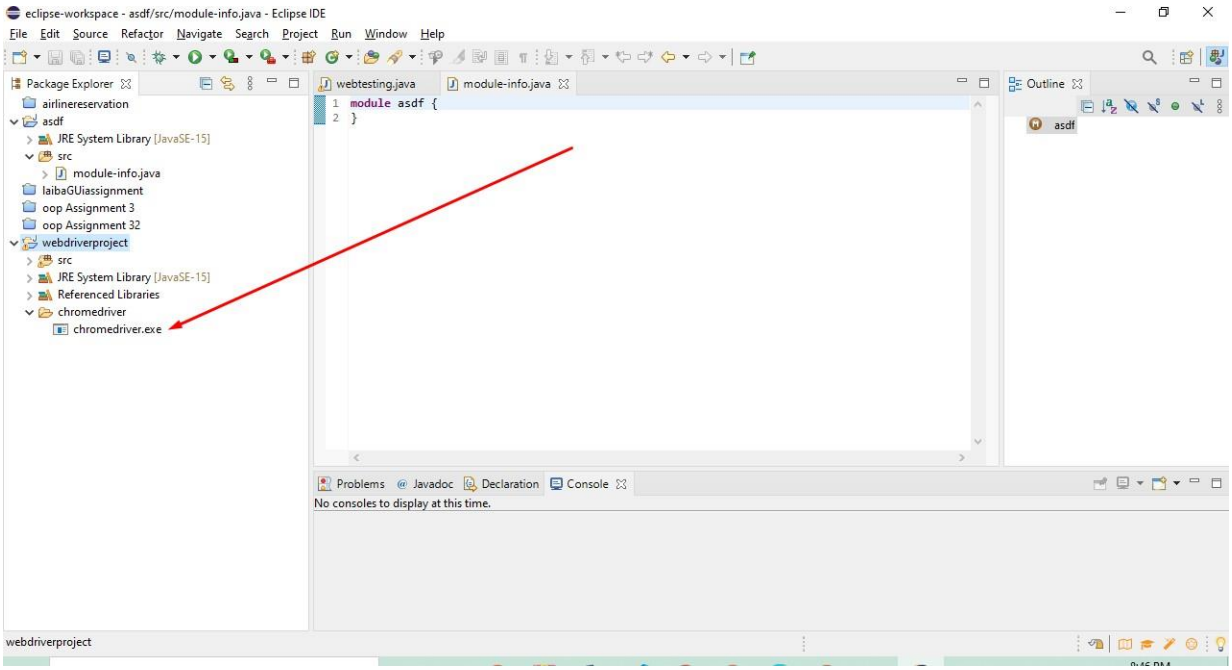
16: Go to Google and Search Chrome selenium web driver.



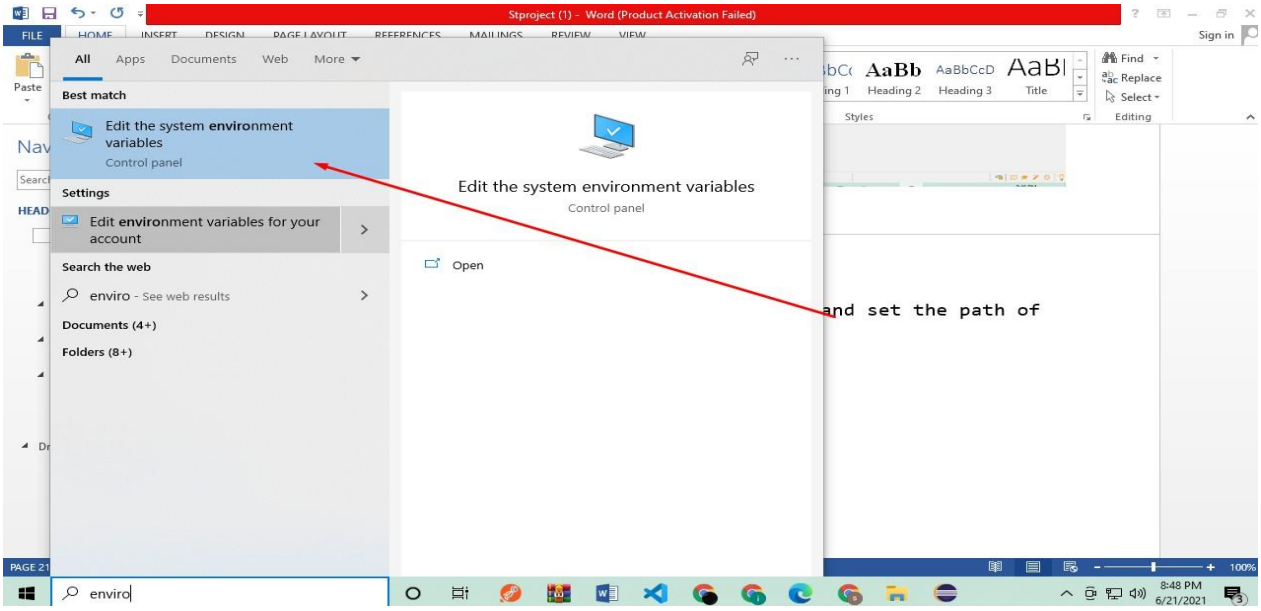
17: Download Selenium According to your version of chrome.(To check your chrome version go to help and then about of chrome)

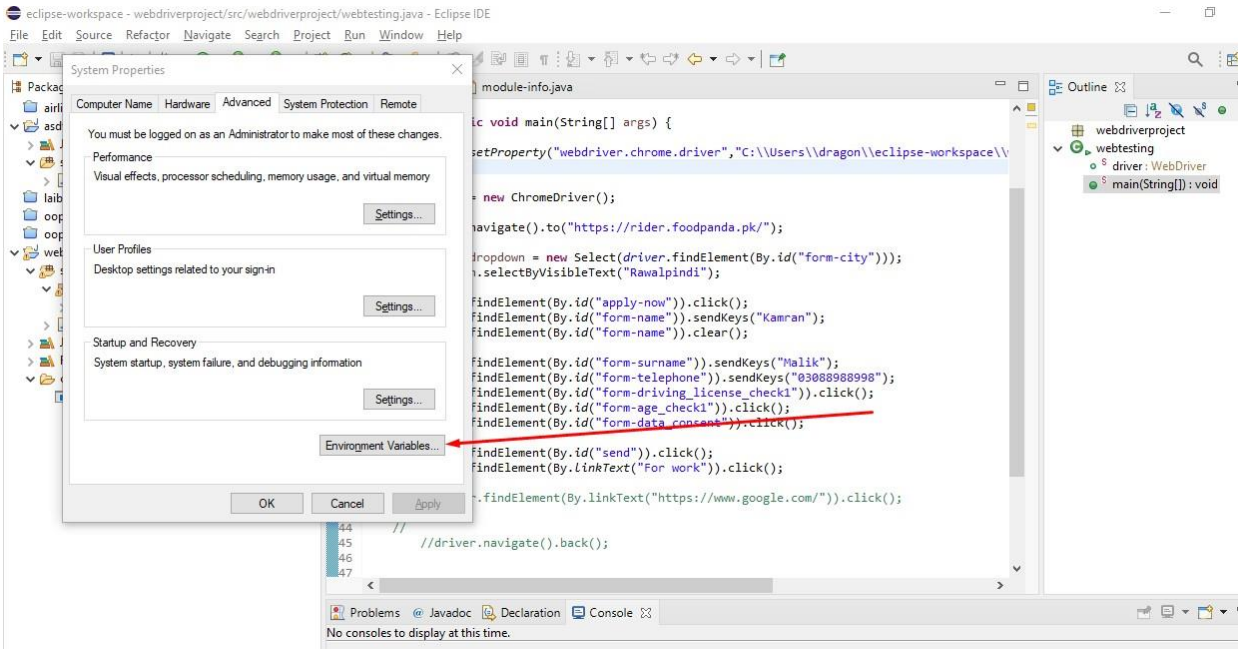


18: Copy the chrome driver that you have downloaded andPaste it into this folder.

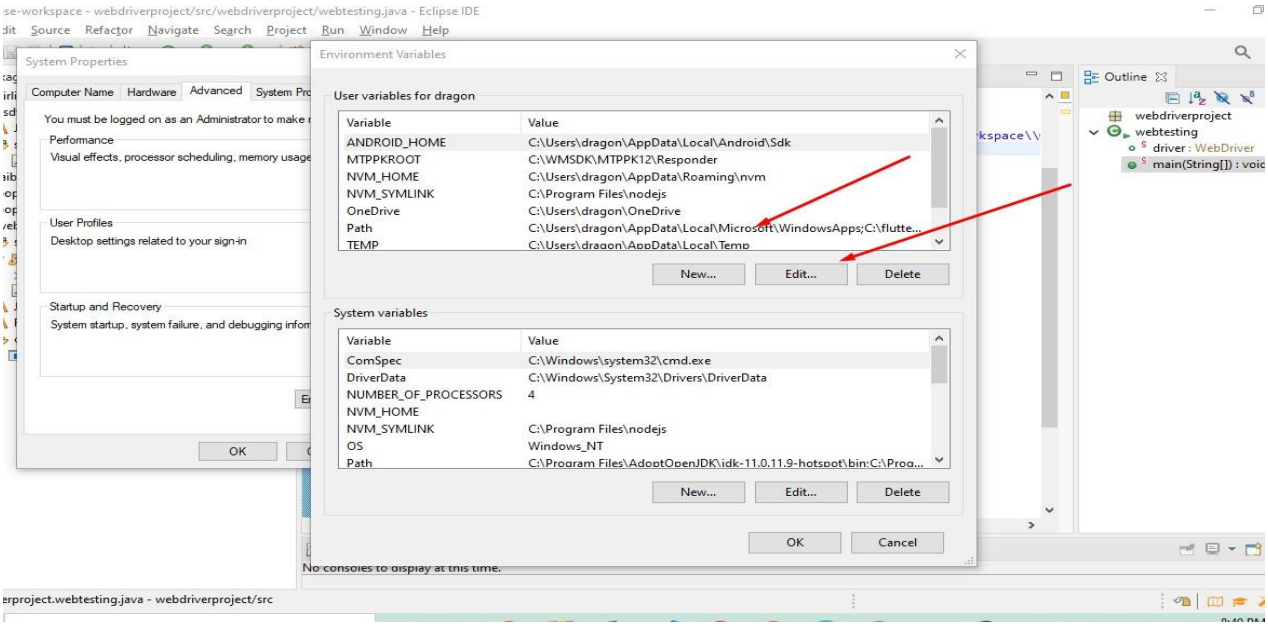


19: Go to environment variables and set the path of chrome.

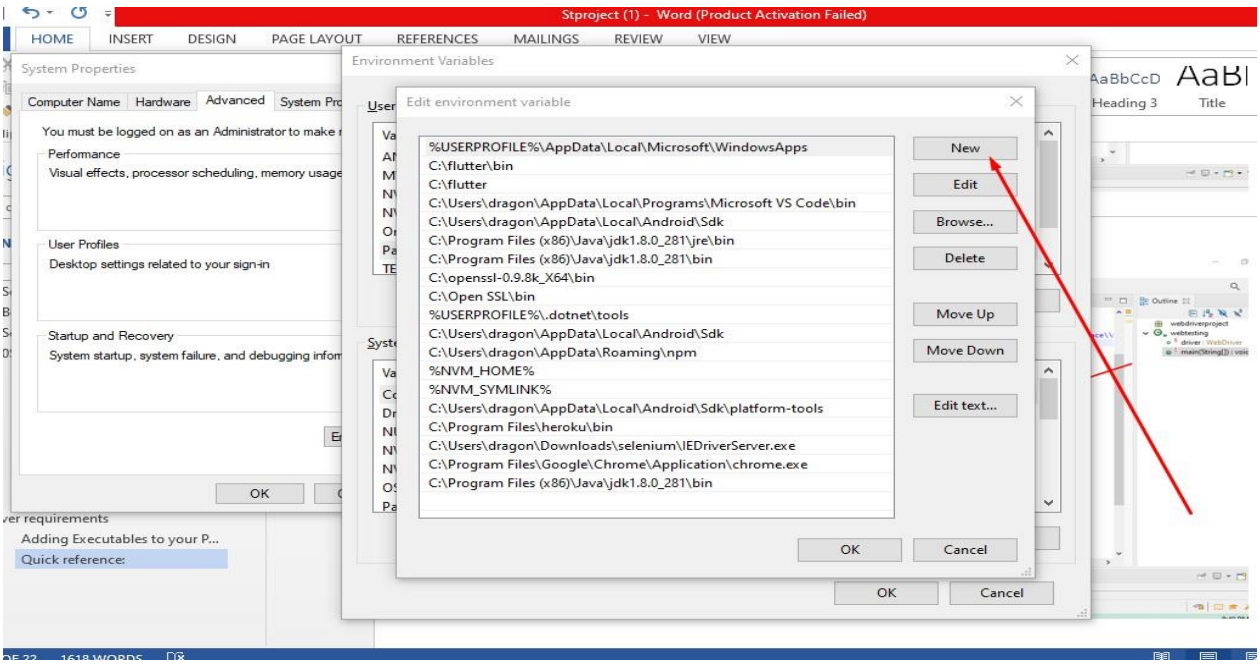




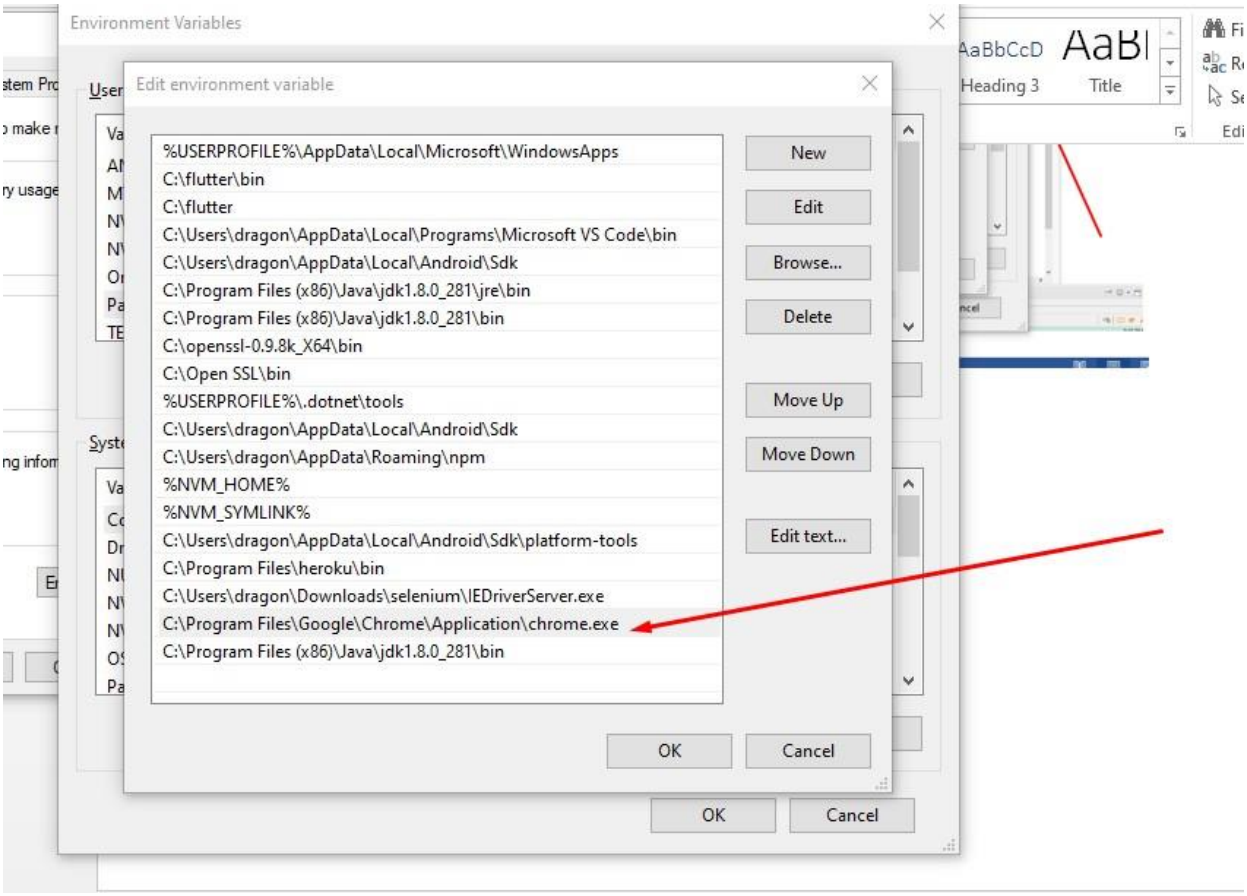
20: Click on path and then click on edit button.



21: Click on New button and add path of chrome.



(hint) you can access the chrome path from your program files(x86) folder then go to google thenchrome then application and there you can find your chrome…



22: Create a class of webtesting in project.

-

```
webtesting.java   module-info.java
 1  package webdriverproject;
 2  import java.util.concurrent.TimeUnit;
10
11
12
13  public class webtesting {
14
15      public static WebDriver driver = null;
16
17      public static void main(String[] args) {
18
19          System.setProperty("webdriver.chrome.driver","C:\\Users\\dragon\\eclipse-workspace\\
20
21
22          driver = new ChromeDriver();
23
```

23: Initialize web driver as null.

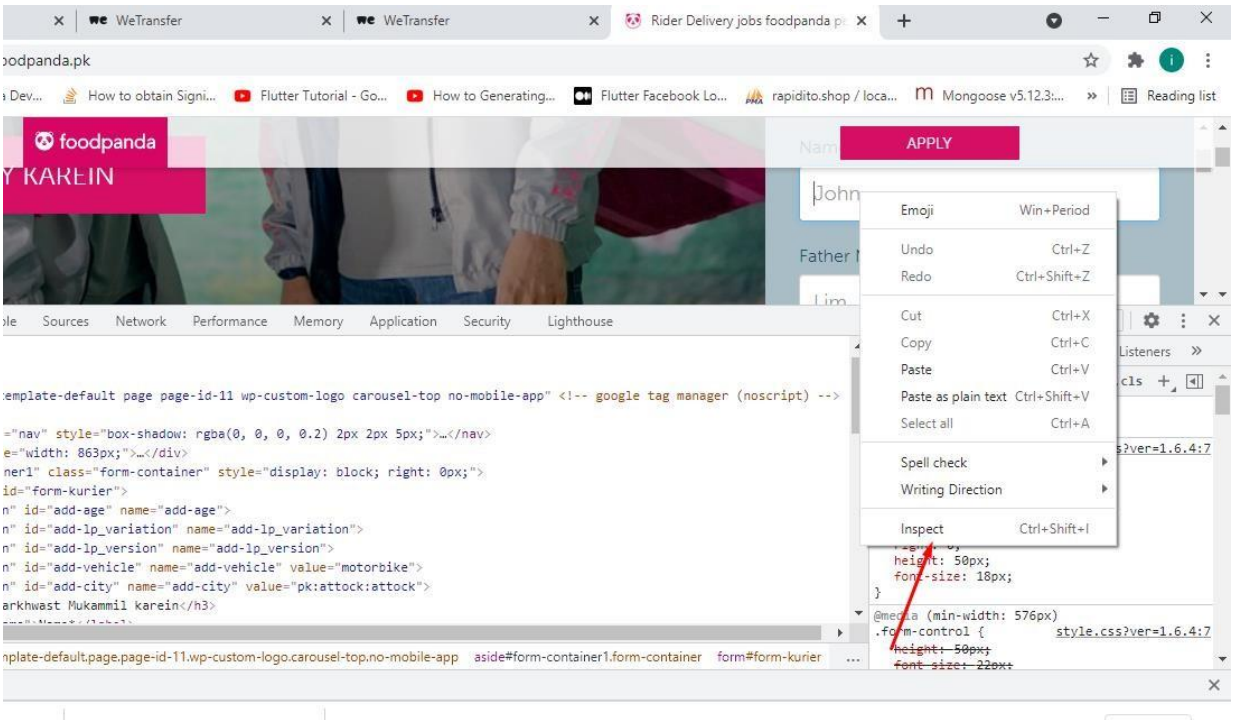
```
public static WebDriver driver = null;
```

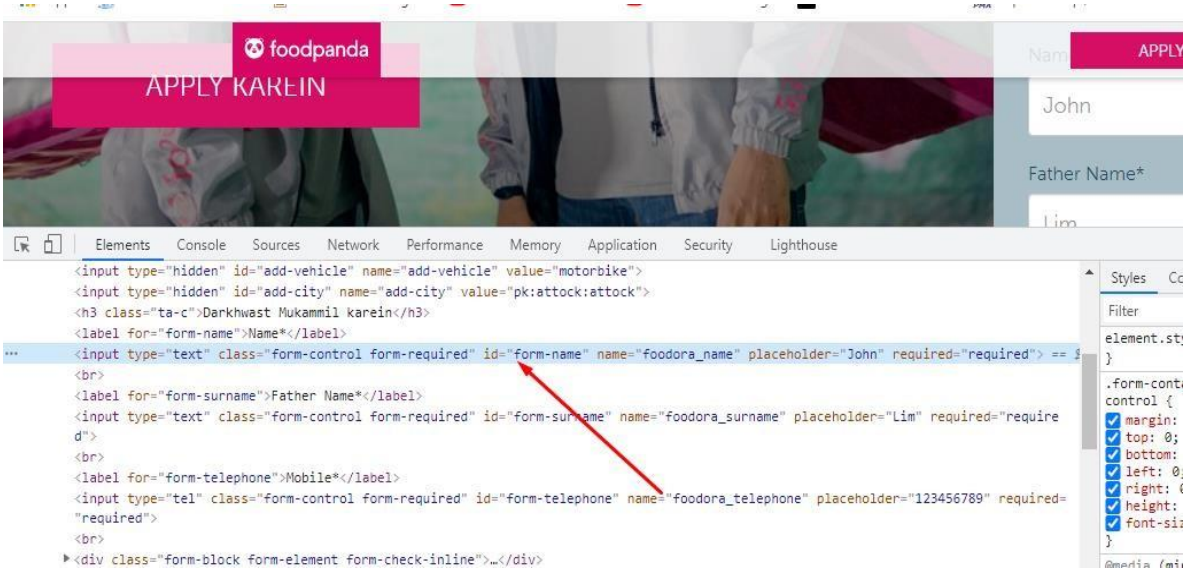
24: give the path of chrome web driver that you havedownloaded for browser in set property field.

```
System.setProperty("webdriver.chrome.driver","C:\\Users\\dragon\\eclipse-workspace\\webdriverproject\\chromedriver\\chromedriver.exe");
```

25: Give the link of website that you want to test.

26: put the id of element that you wanna check(you can get idof element by following this image below).





27: You have to give the id of element either button or text totrig any action.

```
        driver = new ChromeDriver();

        driver.navigate().to("https://rider.foodpanda.pk/");

        Select dropdown = new Select(driver.findElement(By.id("form-city")));
        dropdown.selectByVisibleText("Rawalpindi");

        driver.findElement(By.id("apply-now")).click();
        driver.findElement(By.id("form-name")).sendKeys("Kamran");
        driver.findElement(By.id("form-name")).clear();

        driver.findElement(By.id("form-surname")).sendKeys("Malik");
        driver.findElement(By.id("form-telephone")).sendKeys("03088988998");
        driver.findElement(By.id("form-driving_license_check1")).click();
        driver.findElement(By.id("form-age_check1")).click();
        driver.findElement(By.id("form-data_consent")).click();

        driver.findElement(By.id("send")).click();
        driver.findElement(By.linkText("For work")).click();

        //driver.findElement(By.linkText("https://www.google.com/")).click();

    //
        //driver.navigate().back();
```

Lab Activities:

1. Amazon automated testing using selenium web driver

Code:

```
package seleniumwebdriver;
```

```
import java.util.List;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.Select;

public class AmazonTesting {

	public static WebDriver driver = null;
```

Test case 1: Opening most recent review of a product starting from home page

```
	public static void mostRecentReview() {
		// getting most recent review of a product
		System.setProperty("webdriver.chrome.driver",
				"C:\\Users\\user\\Downloads\\chromedriver_win32
(1)\\chromedriver.exe");
		driver = new ChromeDriver();
		driver.navigate().to("https://www.amazon.com/");
		driver.findElement(By.id("desktop-grid-5")).click();
		List<WebElement> elements = driver
				.findElements(By.className("_octopus-search-result-
card_style_apbSearchResultItem__2-mx4"));
		elements.get(3).click();
```

```
            driver.findElement(By.linkText("See all reviews")).click();

            Select reviewOptions = new Select(driver.findElement(By.id("sort-order-
dropdown")));

            reviewOptions.selectByValue("recent");

//          System.out.println("hello");

            try {

                  driver.findElement(By.xpath(


      "/html/body/div[1]/div[3]/div/div[1]/div/div[1]/div[5]/div[3]/div/div[2]/div/d
iv/div[2]/a[2]"))

                              .click();

            } catch (org.openqa.selenium.StaleElementReferenceException ex) {

                  driver.findElement(By.xpath(


      "/html/body/div[1]/div[3]/div/div[1]/div/div[1]/div[5]/div[3]/div/div[2]/div/d
iv/div[2]/a[2]"))

                              .click();

            }

      }
```

Test case 2: Search a product and filter results

```
      public static void search() {

            System.setProperty("webdriver.chrome.driver",

                        "C:\\Users\\user\\Downloads\\chromedriver_win32
(1)\\chromedriver.exe");

            driver = new ChromeDriver();

            driver.navigate().to("https://www.amazon.com/");

            driver.findElement(By.id("twotabsearchtextbox")).sendKeys("airpods" +
Keys.ENTER);

            driver.findElement(By.id("p_n_cpf_eligible/21512497011")).click();

            try {
```

```
                TimeUnit.SECONDS.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            driver.findElement(By.id("p_36/1253507011")).click();
        }
```

Test case 3: Open carrier information

```
        public static void carrierInfo() {
            System.setProperty("webdriver.chrome.driver",
                        "C:\\Users\\user\\Downloads\\chromedriver_win32
(1)\\chromedriver.exe");
            driver = new ChromeDriver();
            driver.navigate().to("https://www.amazon.com/");
            driver.findElement(By.linkText("Customer Service")).click();
            driver.findElement(By.id("foresight-help-topic-2")).click();
            driver.findElement(By.className("help-card-wrapper")).click();
        }


        public static void main(String[] args) {
            //mostRecentReview();
            //search();
            carrierInfo();
        }
}
```

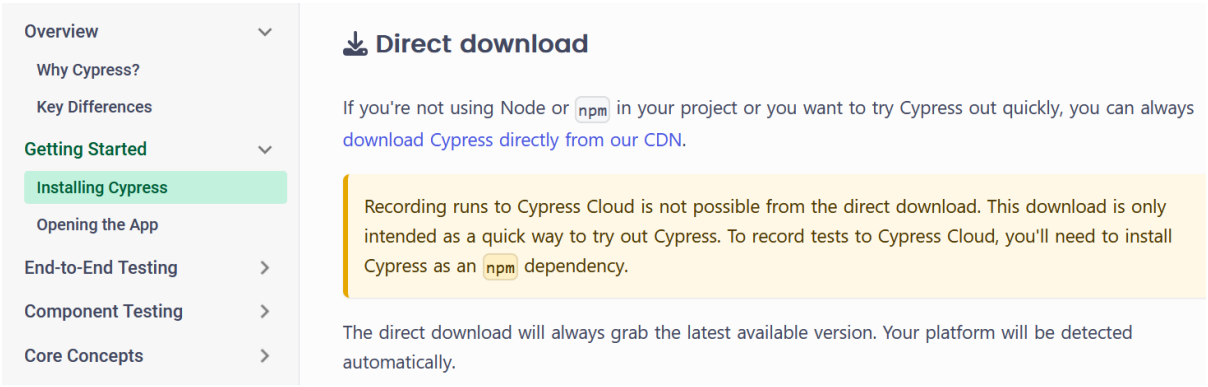
Lab 12

Cypress

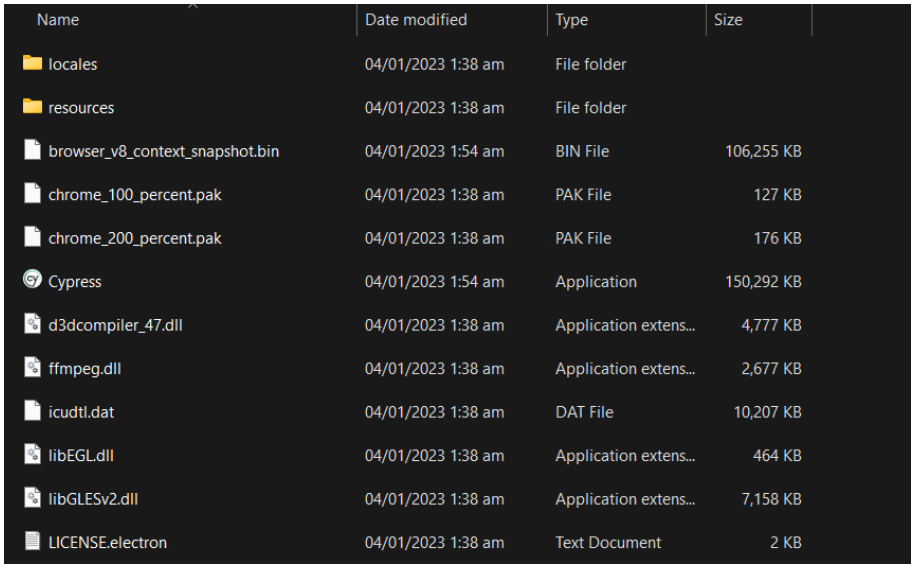
Cypress is test automation tool and we can test any thing that runs on a web browser. Cypress tests run on browser and doesn't use selenium. It's open source and we need node.js and it comes with npm module(Node package manager in which we will be able to get all the libraries we wants to for the cypress automation tool).

Installation(1st Way):

i. Go to the installation instructions link on cypress website, i.e., https://docs.cypress.io/guides/getting-started/installing-cypress

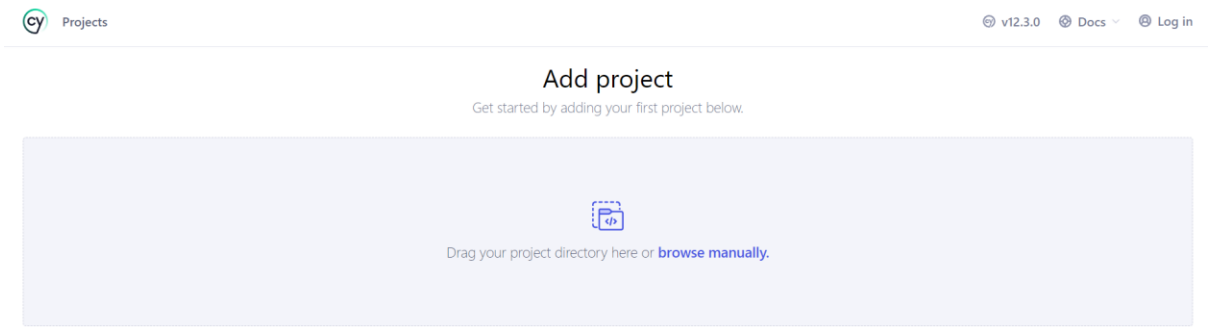


ii. Click the 'download Cypress directly from out CDN' link.
iii. The download will start, which is a zip file.
iv. After download is completed, unzip the file, which will contain the following content.



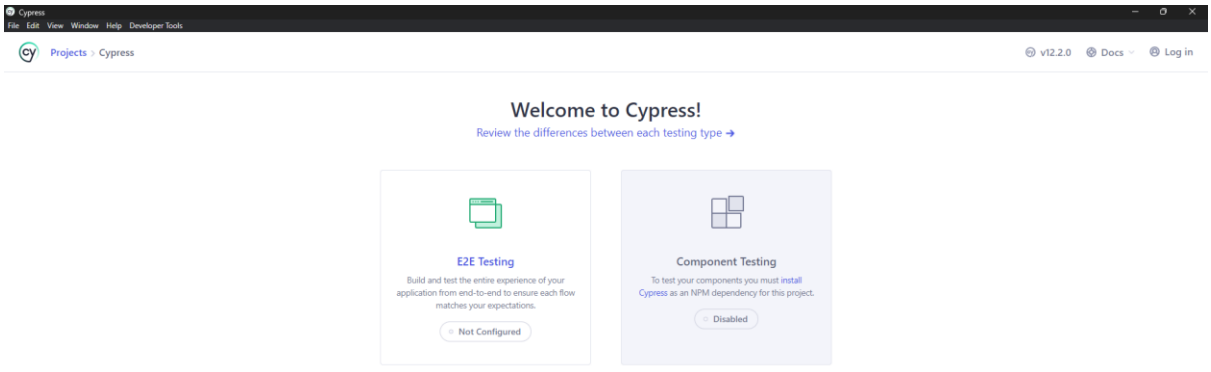
Name	Date modified	Type	Size
locales	04/01/2023 1:38 am	File folder	
resources	04/01/2023 1:38 am	File folder	
browser_v8_context_snapshot.bin	04/01/2023 1:54 am	BIN File	106,255 KB
chrome_100_percent.pak	04/01/2023 1:38 am	PAK File	127 KB
chrome_200_percent.pak	04/01/2023 1:38 am	PAK File	176 KB
Cypress	04/01/2023 1:54 am	Application	150,292 KB
d3dcompiler_47.dll	04/01/2023 1:38 am	Application extens...	4,777 KB
ffmpeg.dll	04/01/2023 1:38 am	Application extens...	2,677 KB
icudtl.dat	04/01/2023 1:38 am	DAT File	10,207 KB
libEGL.dll	04/01/2023 1:38 am	Application extens...	464 KB
libGLESv2.dll	04/01/2023 1:38 am	Application extens...	7,158 KB
LICENSE.electron	04/01/2023 1:38 am	Text Document	2 KB

v. Click the Cypress.exe file, after which you will see the following interface.



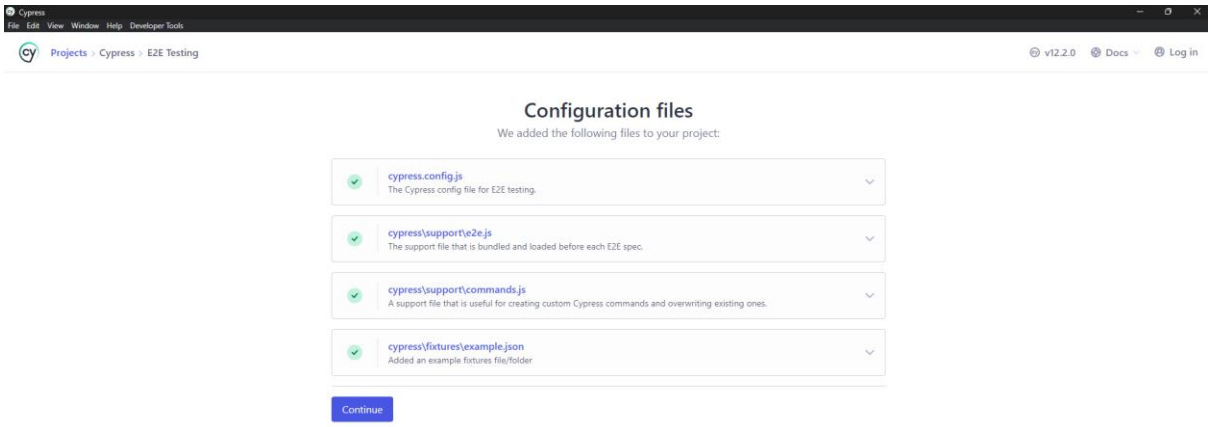
vi. Click on browse manually, and select an empty folder, where you want to save your cypress project.

vii. After you select the folder, the following screen will be displayed.

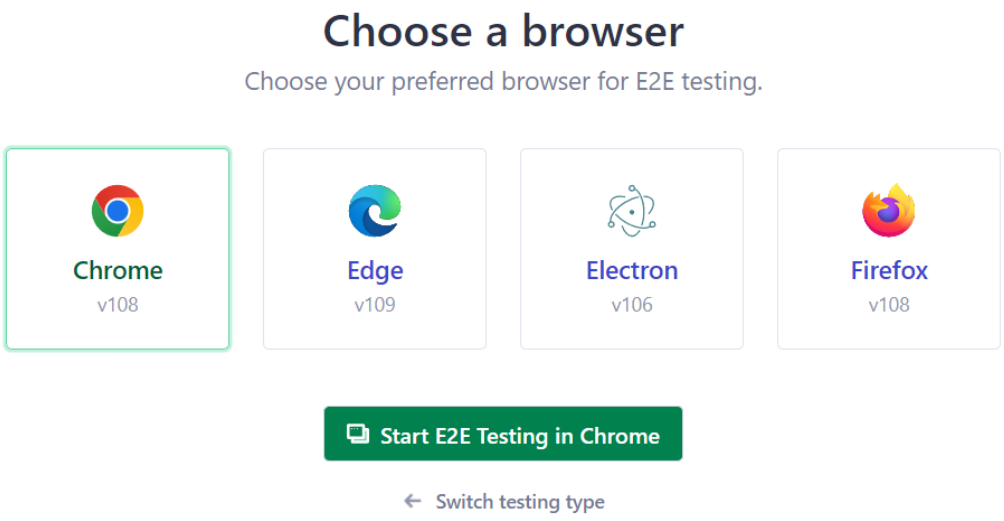


viii. Here you can either configure your project to be end-to-end testing based, or component testing based. In our case, we will click ‘E2E Testing’ option.

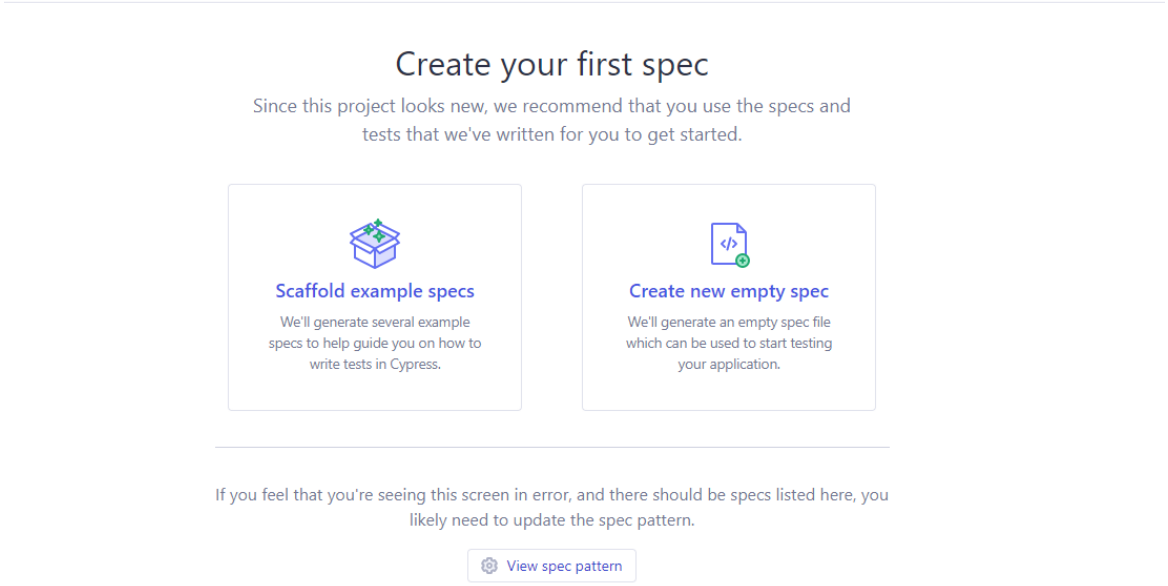
ix. Click the ‘Continue’ button from the next screen.



x. From the next screen, select the browser you want your test to be launched in.

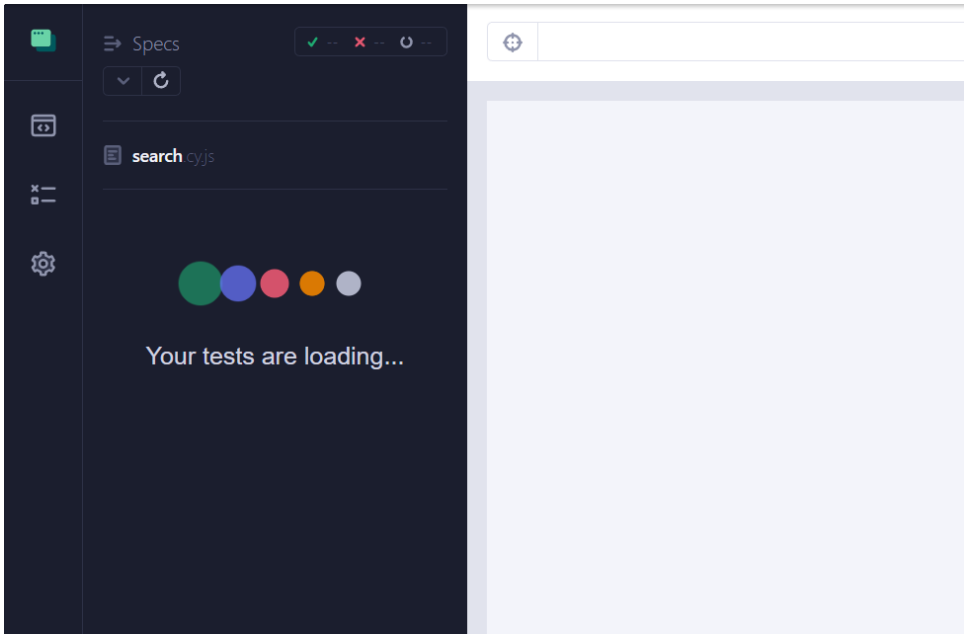


xi. Click the ‘Start E2E Testing’ button, after which the following screen will be displayed in browser.



xii. Click ‘Create new empty spec’ option, then click ‘Okay run the spec’ option from the popup.

xiii. Then your test will be run in the browser and the results will be displayed in the following manner.

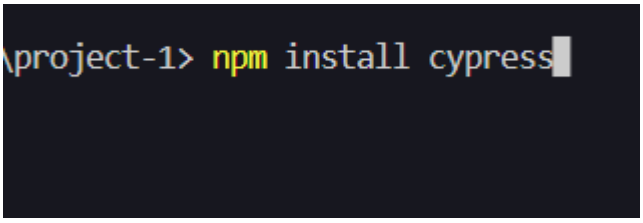


Installation (2nd way)

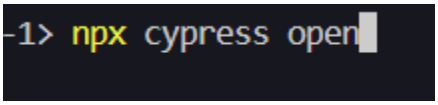
Cypress is a web application testing tool that is used to test web applications. The framework uses JavaScript and can be used to test various kinds of web apps irrespective of the platform or framework that was used to develop them. Cypress can be installed in a project or an empty folder or can be downloaded as an application. All you need is the latest version of Nodejs installed on your computer, which you can install from the following link.

https://nodejs.org/en/

To install cypress in the project you need to run the following command in the folder of the project you want to test.



A file with cypress configuration will be created name as cypress.json and a folder with cypress will be created on the first run, to run cypress from the project terminal run the given below command:



You can follow detailed instructions to install Cypress on your system from this link.

https://docs.cypress.io/guides/getting-started/installing-cypress

Setup:

Once downloaded extract the zip file which you just downloaded. The extracted contents of the zip file represent the cypress app as whole. Once you run the cypress you can add a project just by selecting a folder and configuring it.

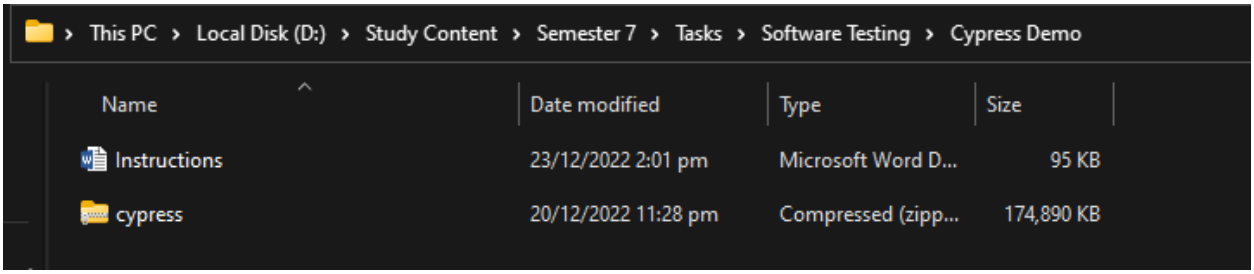


Figure 1 Downloaded Cypress

After installation go in the extracted folder and run the cypress.exe file.

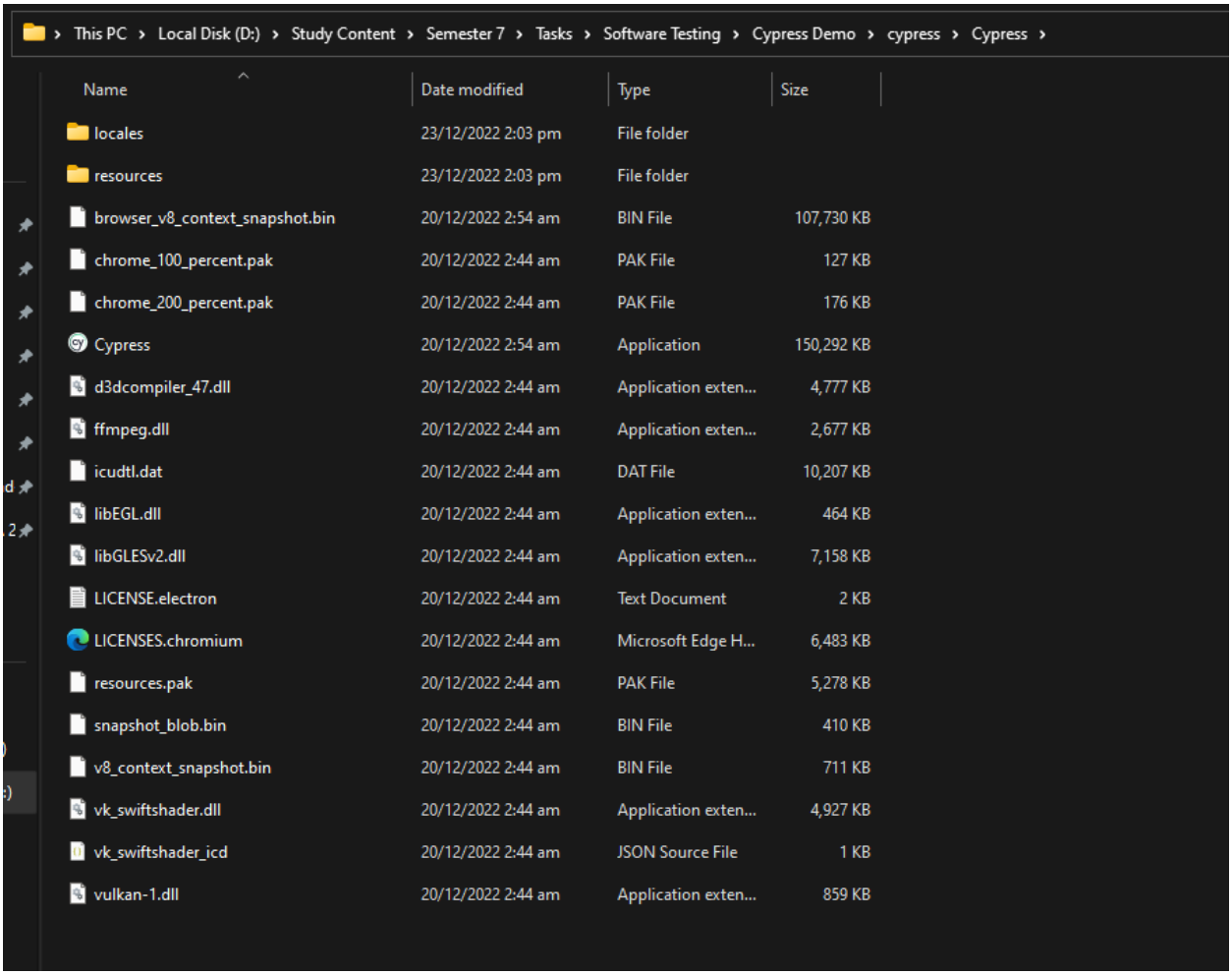


Figure 2 Extracted Folder

After double clicking the exe file the cypress application will run.

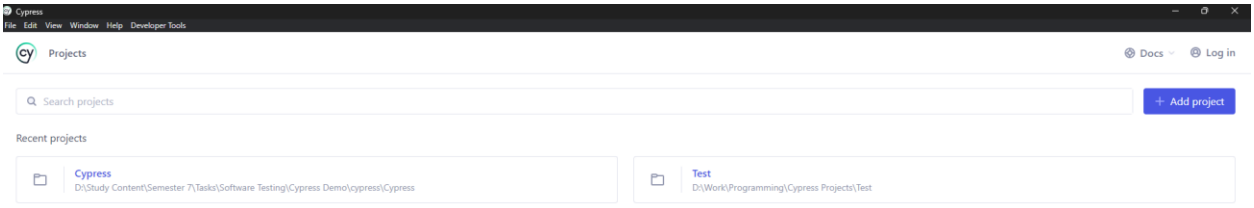


Figure 3 Cypress Initial Screen

Click on the Add Project and select the any folder where you want to configure the cypress.

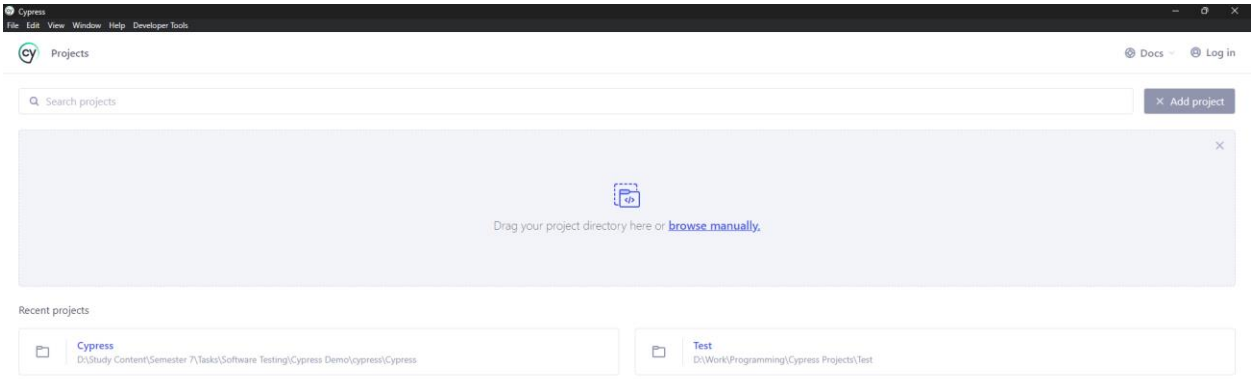


Figure 4 After Clicking Add Project Button

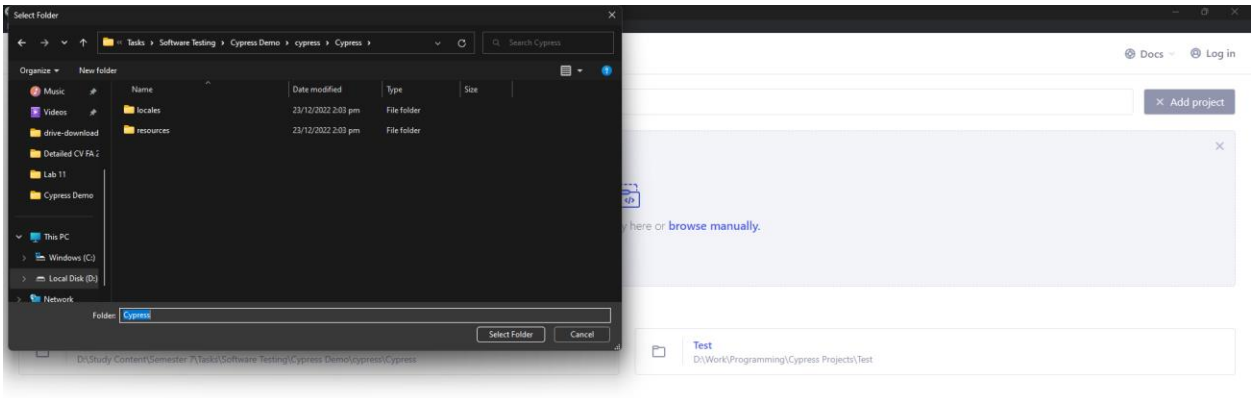
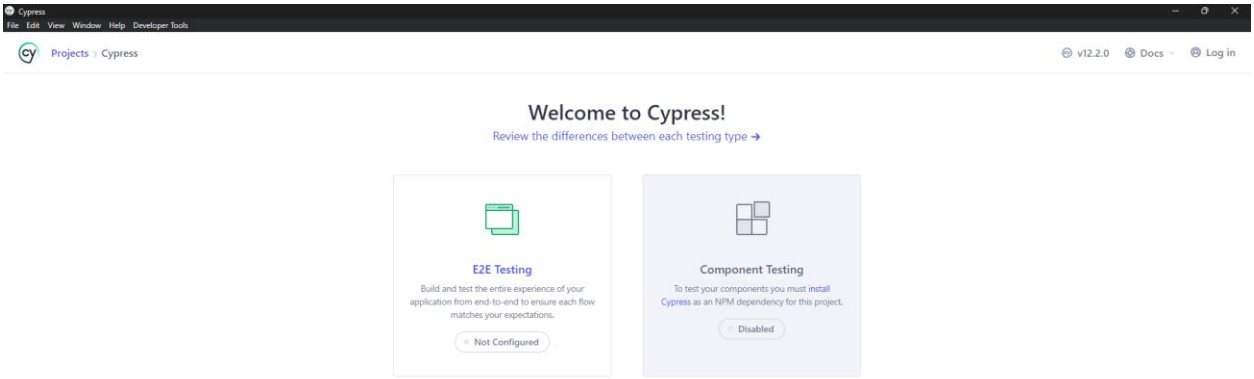
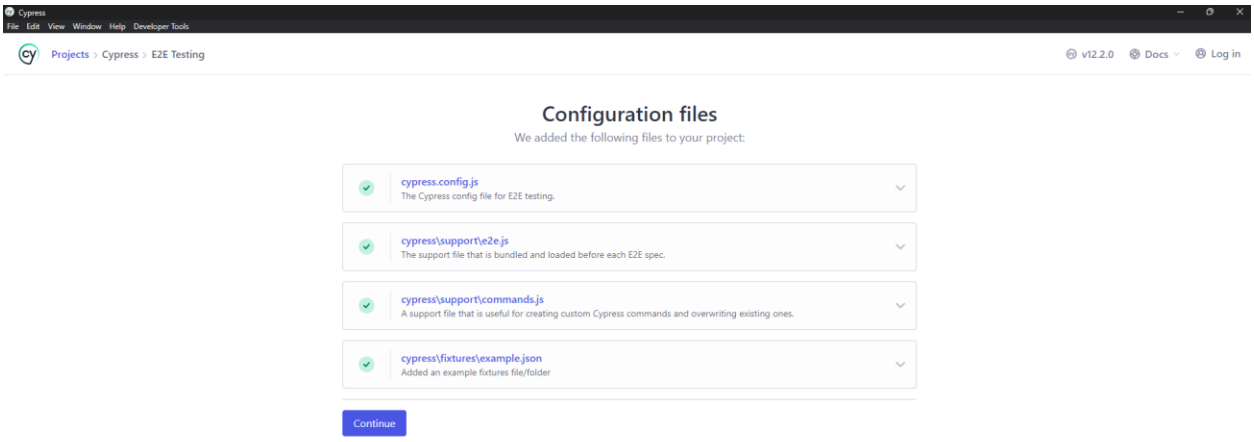


Figure 5 Manually Selecting the extracted folder

Once the project folder has been selected the below screen will appear.

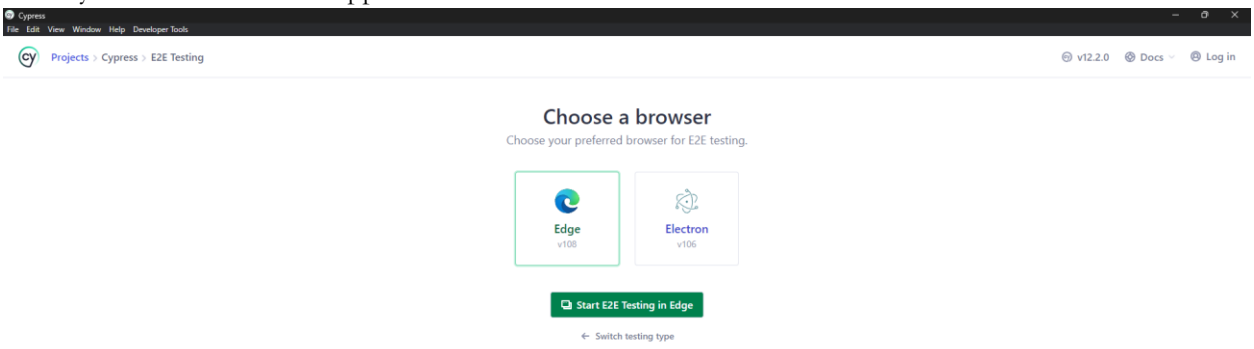


On this screen select E2E testing also known as End to End testing. This testing will allow you to test functionality of the website from the user perspective. After clicking the E2E Testing for the first time the below screen will appear.

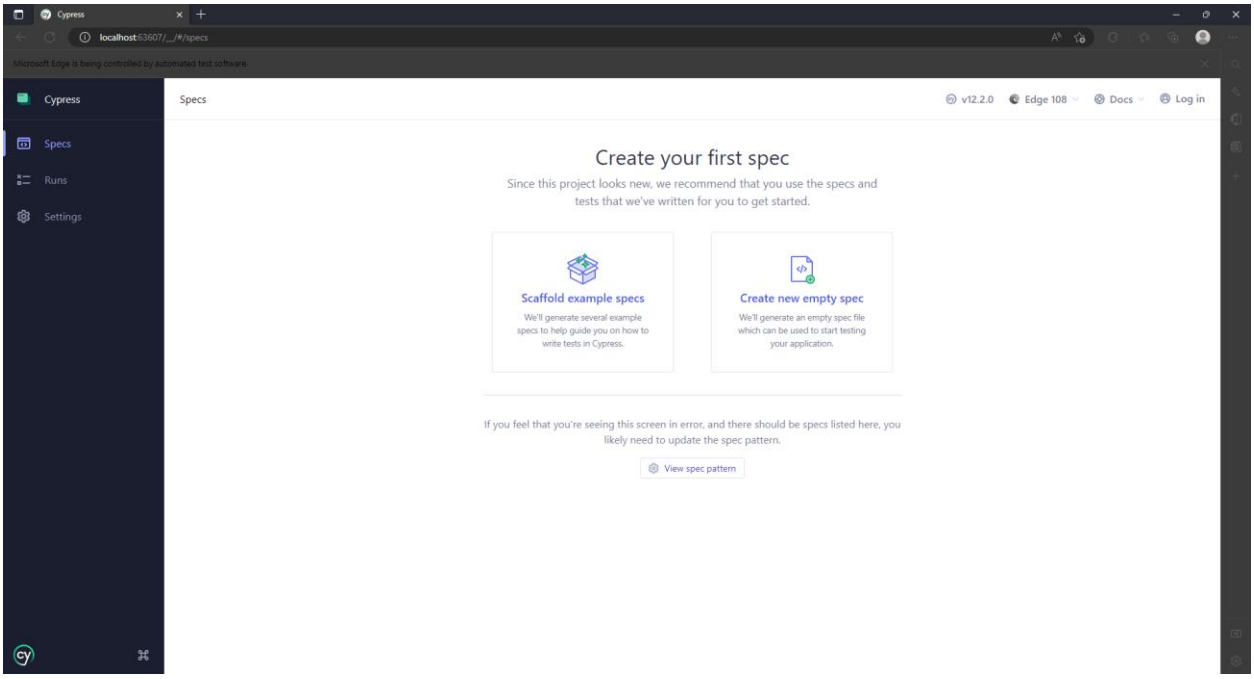


On this screen click on the continue option which will configure the project for End to End testing.

After clicking the continue button you will be presented by the option to select the browser. All the browsers which you have installed will appear here with and addition of electron browser.

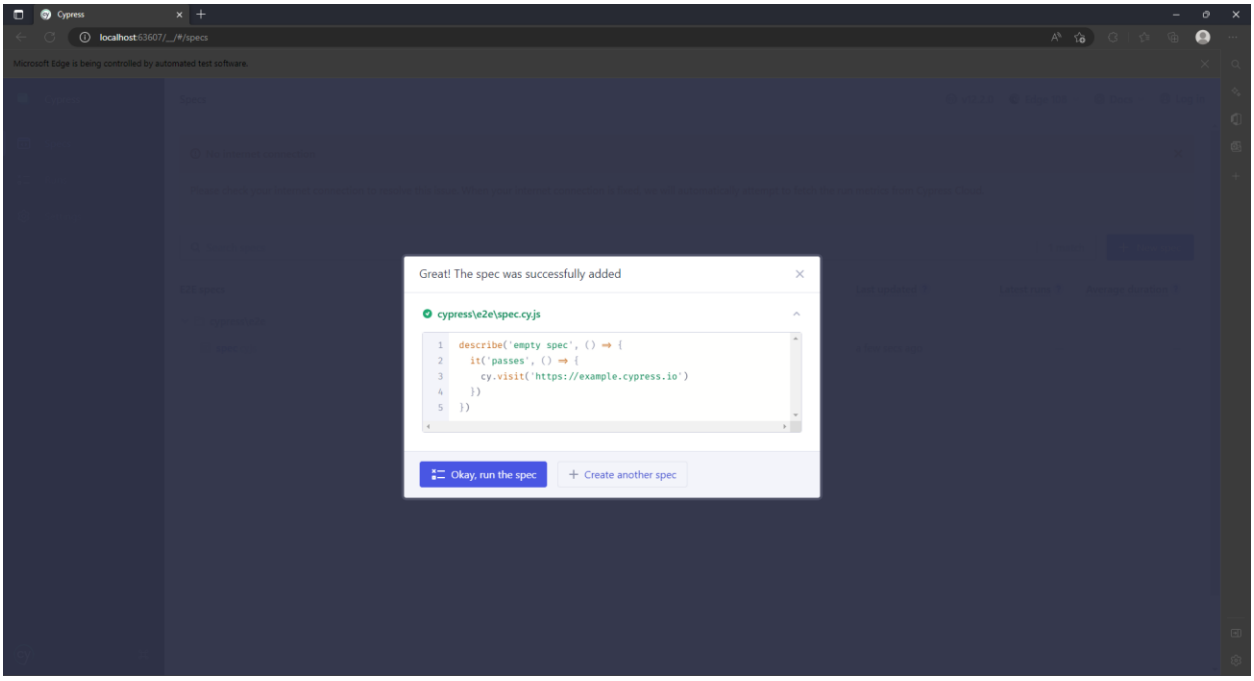


Once you have selected the browser and clicked on Start E2E Testing the cypress dashboard will open in a new browser instance.



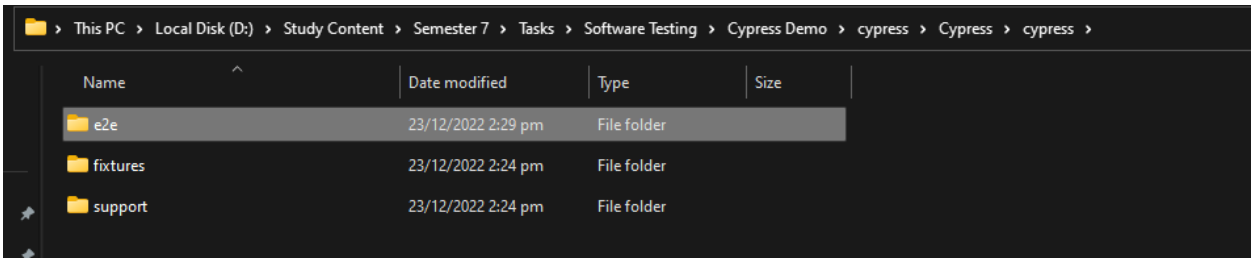
On this page there will be two options. 1st is to use the Scaffolded examples which you can run and learn from. 2nd is to create a new empty spec in which you can type your own test cases.

Click on create a new empty spec. This will show a popup to enter the name of the file. After adding the name if will show you the below screen to add a new spec or run the new spec which you just added.



Once you have created the spec now go to the folder where you have extracted the cypress files.

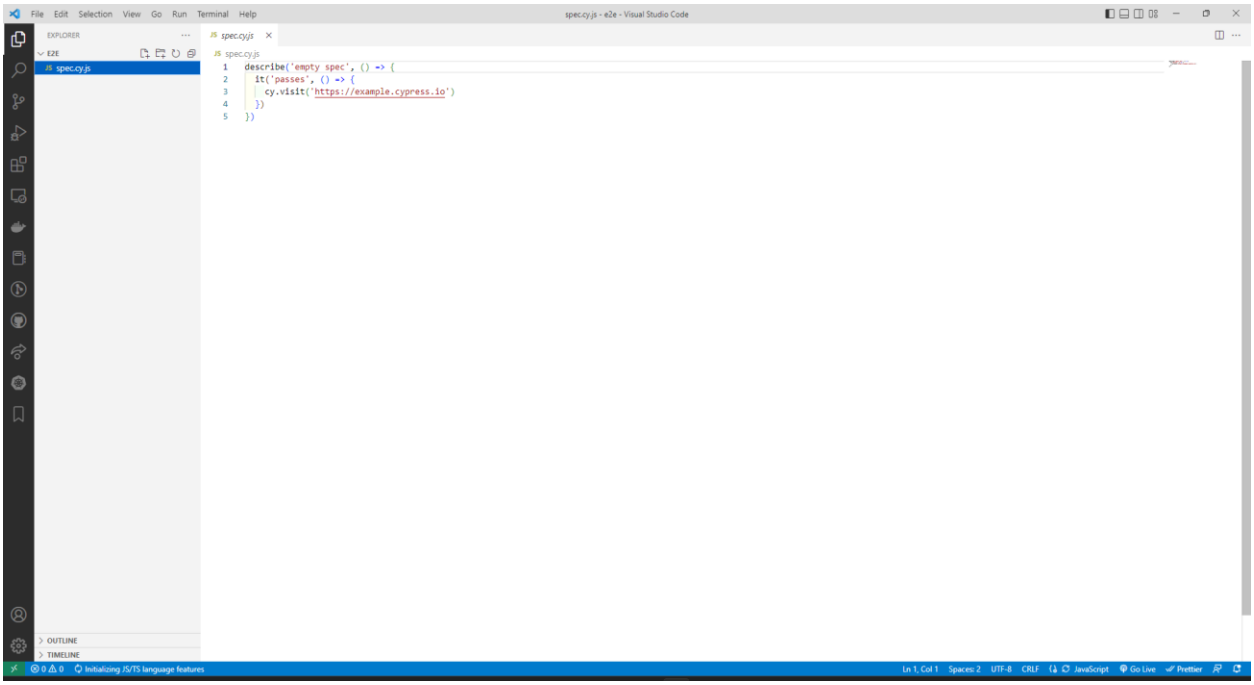
Go in the cypress folder.



In that folder you will see 3 different folders.

1) e2e: This folder will contain all of your scripts.
2) fixtures: This folder will be used to store data in json format that can be used to simulate data retrieval from API.
3) support: This folder will be used to write common commands which you can use in all of your scripts.

Go to the e2e folder and open the folder in visual studio code or any other IDE which you prefer. Once you have opened the visual studio code you will see the initial script which you have created.



In cypress **describe** function is used to organize your test cases in test suites. Where are **it** function is used to write individual test cases.

Now we are going to test **Daraz.pk** site. [Online Shopping in Pakistan: Fashion, Electronics & Books - Daraz.pk]

Describe Method:

Describe method is used to define the test suite. Multiple test suites can be written in a single file but 1 test suite per file is recommended for test case management and navigation. A describe method can contain list of test cases.

```
describe("name", () => {

});
```

It Method:

Its method contains the code for a specific test case. Each test case requires a name. If you have multiple test cases, then to run a single test case you can use **only** method with the it method.

```
it("name", () => {});

it.only("name", () => {});
```

Before, Before Each, After and After Each Methods:

```
before(() => {
   cy.viewport({width}, {height});

   cy.viewport('name')
   cy.visit("{url}");
});



beforeEach(() => {});



after(() => {});



afterEach(() => {});
```

Cypress commands

Cypress contains numerous different commands, that can be used for writing test scripts. Some of the most frequently used commands are listed below, while a complete list of these commands can be found in cypress docs at https://docs.cypress.io/api/

i. describe

This method is used to define a test suite, in which multiple test scripts are written. It takes two arguments, first is a string defining the name of the test suite, while the second is the function that contains all the test scripts.

ii. its

This method is used to define a test script, and is usually written within the test suites created by describe method. It also takes two arguments, first is a string defining the name of the test case, while the second is the function that contains the code of the test script.

iii. before

This method is defined inside the describe method, and is executed only once, that is, before execution of the test scripts of the test suite. This method has two arguments, first is the name of the method as a string, and the second argument is the function which is to be executed before test cases.

iv. beforeEach

This method is defined inside the describe method, and is executed before execution of every test script of the test suite. This method has two arguments, first is the name of the method as a string, and the second argument is the function which is to be executed before the test case.

v. get

This method is used to search for a particular web element from HTML DOM. It takes a string argument, which is the query selector for selecting the particular element. Cypress uses jQuery's query selection engine, so you can write any selector string that is relevant in jQuery. Some examples for query selector are given below.

a. Select element(s) with class heading
   get('.heading')
b. Select element(s) with attribute href
   get('[href]')
c. Select element(s) with id of title, class of text and alt attribute of 'image'
   get('#title .text [src="image"]')

vi. find

This method is almost same as get, but the main difference is that it looks for element within a particular html element, like a div. For example, get('div').find('h1') returns the h1 elements within div tags only.

vii. contains

This method selects and returns the element, which contains the text passed to this method as argument.

viii. click

This method is used to perform a click on the element, which has been selected by the get() method.

ix. type

This method is used to type in a text field, which has been selected by the get() method. The text to be typed is passed as an argument to this method.

x. should

This method is used to assert something to an element. It takes two arguments. The first is the assertion, and the second is the value to be asserted. For example, get('.price').should('be.lt', 1000). This statement asserts that the price in the selected element should be less than 1000. There are various assertions that could be made using this method.

xi. wait

This method is used to pause execution of test script and wait for particular number of milliseconds, which are passed to it as the argument.

xii. trigger

This method is used to trigger a particular event on an element. The event to be triggered on the element is passed as a string to this method. For example, get('element').trigger('mouseover').

Lab 13

Miscellaneous Features in Cypress

Basic Methods

- **visit** method is used to navigate to a specific link
- **contains** method will find the element containing the specified text.
- **get** method will find number of elements based on the given css selector.
- **trigger** method will be used to trigger any css action like mouse over, mouse move, mouse down, mouse up etc.
- **click** method will be used to click on a clickable item which can be retrieved using get or contains method.
- **first** method retrieves the first element from the list.
- **eq** method can be used to get a specific element from the list
- **children** method will get all the children of the currently selected item.
- **last** method will get last element of the elements.
- **then** method can be used to get the element and perform other operations like storing value etc. Each cypress method is like a promise but not exactly a promise. You cannot await but you can use then method.
- **its** method can be used to get a property of an item like length, href, etc.
- **should** method is an assertion method which can be used to compare values. Different type of assertions can be made using should method like
    - **have.length** to compare length of items
    - **exists** to check if an item exists
    - **be.visible** to check if an item is visible on the screen
    - similarly, other type of assertions can be made using should method
- **expect** method can be used to put an assertion in a then statement.
- **fixture** method to use data from the fixtures.

```
it("{Name}", () => {
    cy.visit("{link}")
    cy.contains("{text}");
    cy.contains("{text}")
        .trigger("{action}");
    cy.contains("{text}")
        .click();
    cy.get("{css-selector}");

    cy.get("{css-selector}")
    .first()
    .children()
    .last()
    .then(el => {});
```

```
    cy.get("{css-selector}")

    .eq(number);

    cy.get("{css-selector}")
    .its('{property}')
    .should("{method}", {value});


});
```

Test Cases for Daraz Site:

Signup Test Cases:

```
describe("Signup Test Cases", () => {

    beforeEach(() => {
        cy.viewport(1280, 786);
        cy.visit("https://www.daraz.pk");
    })

    it("Check Signup Link Exists", () => {
      cy.contains("Signup")
          .should("exist")
          .click();
      cy.contains("Create your Daraz Account")
          .should("exist");
   });

    it.only("Check Fields on Signup Interactable", () => {
       cy.contains("Signup")
            .should("exist")
            .click();
       cy.get(".mod-login-input-phone input")
           .type("03117995371");
       cy.get(".mod-login-input-password input")
           .type("Asimwattoo123$");
       cy.get("#month")
           .click();
       cy.contains("March")
           .click();
       cy.get("#day")
           .click();
       cy.contains("13")
           .click();
       cy.get("#year")
```

```
            .click();
        cy.contains("2003")
            .click();
        cy.get("#gender")
            .click();
        cy.contains("male")
            .click();
        cy.get(".mod-login-input-name")
            .type("Muhammad Asim");
        cy.get("#nc_2_n1z")
            .trigger("mousedown")
            .trigger("mousemove", {clientX: 800, clientY: 100})
            .trigger("mouseup");
        cy.pause();
    });
});
```

Sign in Test Cases:

```
/// <reference types='../cypress' />

describe('Sign In', () => {

    beforeEach(() => {
        cy.viewport(1280, 786);
        cy.visit('https://www.daraz.pk/');
    });

    it('Check Sign In Link on Home Page', () => {
         cy.contains("login")
             .should("exist")
    });

    it("Check Login Fields", () => {
        cy.contains("login")
            .click();
         cy.contains("Error")
             .should("not.exist");
        cy.get("div.mod-login-input-loginName input")
            .type("asimwattoo6@gmail.com");
        cy.get("div.mod-input-password input")
            .type("asimwattoo123");
    });

    it("Check Login Error Shows on Invalid Data", () => {
         cy.contains("login")
```

```
            .click();
        cy.contains("Error")
            .should("not.exist");
        cy.get("div.mod-login-input-loginName input")
            .type("asimwattoo6@gmail.com");
        cy.get("div.mod-input-password input")
            .type("asimwattoo123");
        cy.get("div.mod-login-btn > button")
            .click();
        cy.contains("Error")
            .should("exist");
    });

    it.only("Check Login With Number and Password", () => {
        cy.contains("login")
            .click();
        cy.contains("Error")
            .should("not.exist");
        cy.fixture("credentials.json").then(data => {
            cy.get("div.mod-login-input-loginName input")
                .type(data.number);
            cy.get("div.mod-input-password input")
                .type(data.password);
            cy.get("div.mod-login-btn > button")
                .click();
            cy.url().should("eq", "https://www.daraz.pk/");
        });
    });
});
```

Searching Test Cases

```
describe("Search and Filter Products", () => {

    beforeEach(() => {
        cy.viewport(1280, 786);
        cy.visit("https://www.daraz.pk");
    })

    it("Searching via TextBox", () => {
        cy.get("input#q")
            .should("exist")
            .type("Drones");
        cy.get("button.search-box__button--1oH7")
            .click({force: true});
        cy.get("div.tips--QRnmZ")
```

```
            .should("exist")
            .then(element => {
                expect(element.text()).to.include("Drones")
            })
    });

    it("Category Wise Filtering", () => {
        cy.visit("https://www.daraz.pk")
        cy.contains("Electronic Devices")
            .should("exist")
            .click();
        cy.contains("Smart Phones")
            .trigger("mouseover");
        cy.contains("Apple iPhones")
            .click({force: true});
        cy.get(".title--Xj2oo")
            .should("exist")
            .should("include.text", "Apple iPhones")
    });

    it("Product Filtering", () => {
        let brand = "";
        cy.get("input#q")
            .should("exist")
            .type("Drones");
        cy.get("button.search-box__button--1oH7")
            .click();
        cy.get(".checkbox--tqPns.ant-checkbox-wrapper input")
            .first()
            .check();
        cy.get(".checkbox--tqPns.ant-checkbox-wrapper")
            .first()
            .children()
            .last()
            .then(el => {
                brand = el.text();
                cy.get(".ant-tag")
                    .should("include.text", brand);
            })
    });

});
```

Cart Test Cases:

```
describe('Cart Test Cases', function () {
```

```
    before(() => {
       cy.viewport(1280, 786);
       cy.visit("https://www.daraz.pk/");
       cy.login();
    });

    beforeEach(() => {
        cy.viewport(1280, 786);
        cy.visit("https://www.daraz.pk");
        cy.on('uncaught:exception', (err, runnable) =>
        {
            return false;
        });
    });

    it("Check Product Adding to cart", () => {
        cy.get("input#q")
            .should("exist")
            .type("Keyboards");
        cy.get("button.search-box__button--1oH7")
            .click({force: true});
        cy.get(".gridItem--Yd0sa a")
            .first()
            .click();
        cy.contains("Add to Cart")
            .click();
    });

    it.only("Select all items in the cart", () => {
        let total = 0;
        let shipping = 0;
       cy.get(".lzd-nav-cart")
           .click();
       cy.url().should("include", "cart");
       cy.get(".list-header-container input[type='checkbox']")
           .click();
        cy.get(".cart-item")
            .its('length')
            .should("gt", 0);
       cy.get(".cart-item")
           .each(item => total += parseInt(item.find(".current-
price").text().split(' ')[1].replace(',', '')));
       cy.wait(2000);
       cy.get(".checkout-summary-noline-value")
           .last()
           .then(t => {
              shipping = parseInt(t.text().split(' ')[1].replace(',', ''));
           });
```

```
        cy.get(".checkout-order-total-fee")
            .then(t => {
                cy.log(`Shipping: ${shipping}`)
               let actualTotal = parseInt(t.text().split(' ')[1].replace(',',
''));
               expect(total + shipping).to.eq(actualTotal);
            });
    });

});
```

Fixtures:

```
{
  "number": "",
  "password": ""
}
```

Commands

```
Cypress.Commands.add("login", () => {
    cy.contains("login")
        .click();
    cy.contains("Error")
        .should("not.exist");
    cy.fixture("credentials.json").then(data => {
        cy.get("div.mod-login-input-loginName input")
            .type(data.number);
        cy.get("div.mod-input-password input")
            .type(data.password);
        cy.get("div.mod-login-btn > button")
            .click();
        cy.url().should("eq", "https://www.daraz.pk/", {timeout: 10 * 1000});
    });
});
```

Custom commands:

```
describe('Amazon Complete Operations', () => {
  before(() => {
    cy.visit('https://www.amazon.com');
  });

  it('logs in', () => {
```

```
    cy.login('theayeshamehmood@gmail.com', 'ayesha@123');
  });

  it('adds a product to the cart and checks the count', () => {
    cy.visit('https://www.amazon.com');
    cy.searchForProduct('Echo Dot');
    cy.addProductToCart();
    cy.wait(2000);
    cy.checkCartCount('1');
  });

  it('navigates to Deals and changes delivery location', () => {
    cy.visit('https://www.amazon.com');
    cy.navigateToDeals();
    cy.changeDeliveryLocation('10001');
  });

  it('applies a filter and verifies product details', () => {
    cy.visit('https://www.amazon.com');
    cy.searchForProduct('laptop');
    cy.applyFilter();
  });

  it('checks footer link', () => {
    cy.visit('https://www.amazon.com');
    cy.checkFooterLink('Careers');
  });

  it('navigates to categories',()=>{
    cy.visit('https://www.amazon.com');
    cy.navigateToCategory('Shop By Interest');
  })

  after(() => {
    //cy.signOut();
  });

});
```

Data Driven Tests:

```
// Data Driven Tests
  describe("Amazon Data Driven Tests", () => {
    beforeEach(() => {
      cy.visit(Cypress.env("amazonUrl"));
    });

    it("Searches for multiple products", () => {
      const products = [
        { product: "Echo Dot", expectedResult: "Echo Dot" },
        { product: "Kindle", expectedResult: "Kindle" },
      ];
      products.forEach((product) => {
        cy.searchForProduct(product.product);
        cy.get(".s-main-slot").should("contain",
product.expectedResult);
      });
    });
  });
```

Set Evirnoment Variables:

```
describe('Amazon Login and Search', () => {
  it('logs into Amazon and searches for a product', () => {
    cy.visit('https://www.amazon.com');

    // Logging in
    cy.get('#nav-link-accountList').click();
    cy.get('#ap_email').type(Cypress.env('amazonEmail'));
    cy.get('.a-button-inner > #continue').click();
    cy.get('#ap_password').type(Cypress.env('amazonPassword'));
    cy.get('#signInSubmit').click();

    // Searching for a product
    cy.visit('https://www.amazon.com')
    cy.get('#twotabsearchtextbox').type(`${Cypress.env('searchQue
ry')}{enter}`);
  });
});
```

ENV variables:

```
 env: {
    amazonEmail: "theayeshamehmood@gmail.com",
    amazonPassword: "ayesha@123",
    searchQuery: "Echo Dot",
  }
```

Lab 14

API Mocking, Cypress Studio, Visual Regression Testing

API mocking in Cypress:

Most web applications have dependencies with external APIs when integrating third-party services. Sometimes they may be unavailable during development because of maintenance, or development is still in progress. Mock APIs are useful when we have such external dependencies in our systems.

API mocking decouples the frontend from the backend so frontend developers can speed up the development by building against a well-defined specification.

API mocking can be performed in cypress through the 'intercept' method. It can be used to spy and stub network requests and responses. This method takes three arguments:

i. First argument is the type of request as string, e.g., GET, POST, etc.
ii. Second argument is the API request URL which we want to mock
iii. Third argument is the JSON response we want to provide as a mock to this API request. As a convention, this JSON response is stored inside fixtures.

Example;

```
cy.intercept("GET", `${Cypress.env("apiUrl")}/tags`, {
      fixture: "tags.json",
      }).as("alias_here");
```

Now whenever a request is sent to the tags API URL, then the response stored in tags.json file is returned.

Sending API requests in Cypress:

Sometimes we need to send real API requests, in order to get some real data required for our system. For such purposes, we can send API requests within out cypress test scripts.

For sending API requests, we use 'request' method of cypress. This method takes one argument, which is an object containing fields like;

i. URL – This field contains the API URL to which we want to send the request
ii. method – This field contains the HTTP method for the request, e.g., GET, POST, PUT, etc.
iii. headers – This field contains HTTP header, like content-type and authorization.

iv. body – This field contains the request body

Example;

```
cy.request({
        url: `${Cypress.env("apiUrl")}/articles?limit=10`,
        headers: { Authorization: "Token " + token },
        method: "GET",
      }).then((response) => {
      });
```

Cypress Studio:

By recording interactions against the application under test, Cypress Studio provides a visual way to generate tests within Cypress

Cypress Studio is an **experimental feature** of Cypress that can be enabled by set by setting following property in “cypress.config.js” file:

```
const { defineConfig } = require('cypress')

module.exports = defineConfig({
  e2e: {
    experimentalStudio: true
  }
})
```

Following Cypress commands are supported in Cypress Studio:

- Click ()
- Type ()
- Check ()
- Uncheck ()
- Select ()

When Cypress Studio is activated, Cypress will generate test code when interacting with the DOM inside of the Cypress Studio. We can also generate assertions by right clicking on an element that we would like to assert on.

i. Generating an example test with Cypress Studio:

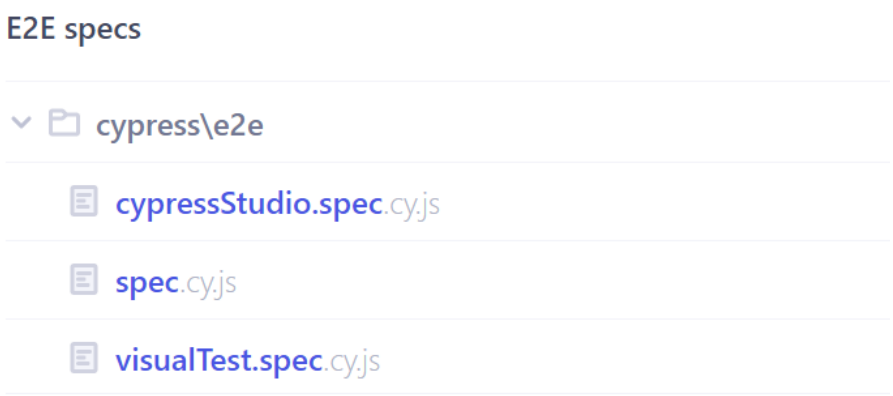
Cypress spec file before running Cypress Studio:

```
describe("My First Test", () => {
  beforeEach("Load website", () => {
    cy.visit("https://www.pakwheels.com/");
  });

  Open Cypress | Set ".only"
  it("Search a car in Islamabad within price range of 10 to 20 Lacs", () => {});
});
```

As we can see in above screenshot that test body is empty which we will now generate through Cypress Studio.

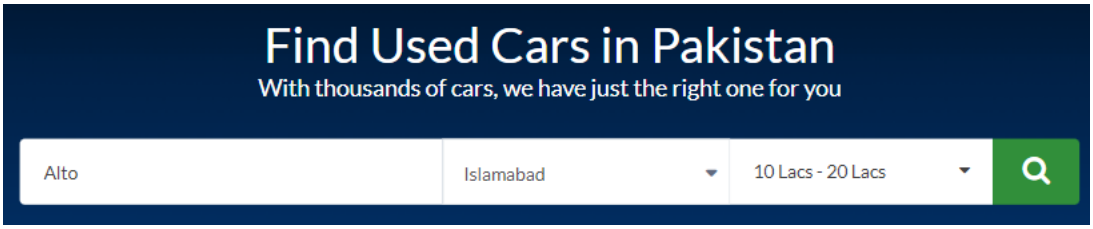
ii. Steps for generating test case:

a. Select desired spec file in which we want to generate test

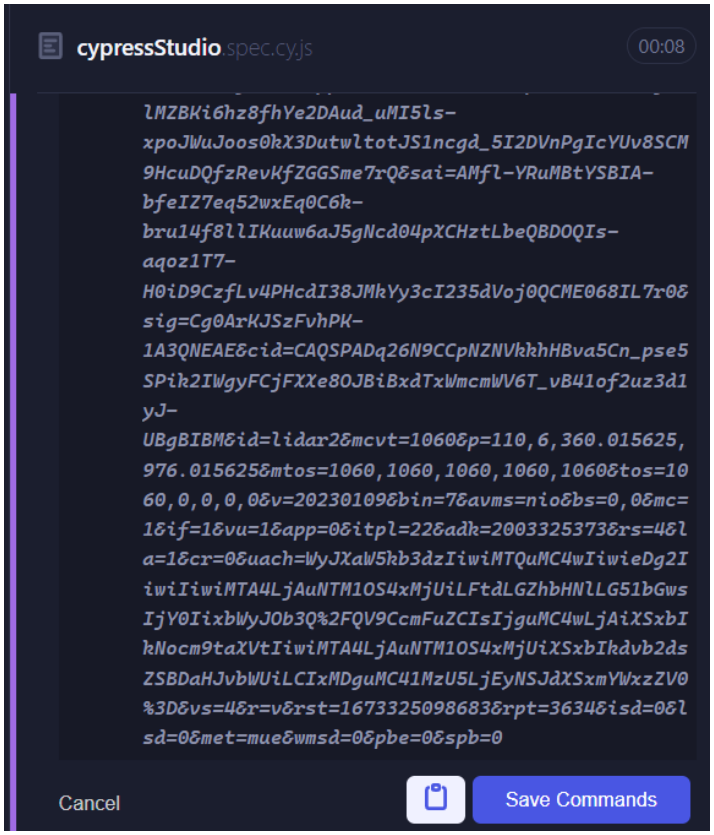


b. Once the tests complete their run, hover over a test in the Command Log to reveal an "Add Commands to Test" button. Clicking on "Add Commands to Test" will launch the Cypress Studio.

b. Start interacting with the application under test like a real user would.



c. After we are done interacting with the application, save the generated commands by clicking on “Save Commands” in bottom of Cypress Test Runner

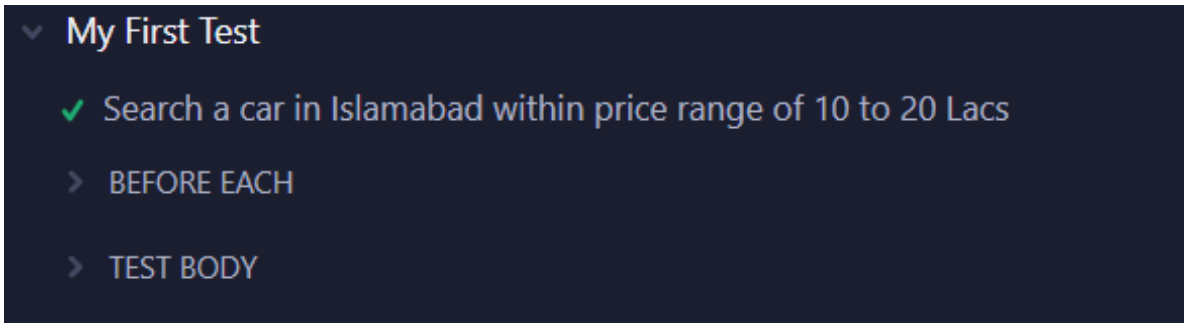


After saving commands, Cypress will run the generated test in the same manner as we interacted with the application, and we can see generated test case in our code editor.

Cypress spec file after running Cypress Studio:

```
it("Search a car in Islamabad within price range of 10 to 20 Lacs", () => {
  /* ==== Generated with Cypress Studio ==== */
  cy.get('#onesignal-slidedown-cancel-button').click();
  cy.get('#home-query').clear('A');
  cy.get('#home-query').type('Alto');
  cy.get('.chzn-single > span').click();
  cy.get('.chzn-search > input').clear();
  cy.get('.chzn-search > input').type('Islamabad{enter}');
  cy.get('.chzn-search > input').clear('Islamaba');
  cy.get('.chzn-search > input').type('Islamabad{enter}');
  cy.get('#pr-range-filter').click();
  cy.get('.tt-dataset > :nth-child(2)').click();
  cy.get(':nth-child(2) > .twitter-typeahead > .tt-menu > .tt-dataset > :nth-child(2)').click
  ();
  cy.get('#home-search-btn').click();
  /* ==== End Cypress Studio ==== */
});
```

Test Result for generated test case:



Visual Regression Testing:

A visual regression test checks what the user will see after any code changes have been executed. This is done by comparing screenshots taken before and after the code changes. Visual test highlights any visual changes that occur after the code update.

Visual tests generate, analyze, and compare browser snapshots to detect if any pixels have changed. We can set the threshold values, which give us the flexibility to compare images with small differences or ignore small differences.

There are a lot of plugins available in Cypress that can be used to capture visual images and compare them. In this demonstration, cypress cypress-visual-regression is used to explain Cypress Visual Testing.

i. Installation instructions for the plugin:

a. *cypress-visual-regression* is available as an NPM dependency. Type the following command in your terminal to install the plugin.

```
npm i cypress-visual-regression
```

b. Add the following config to your cypress.config.js file:

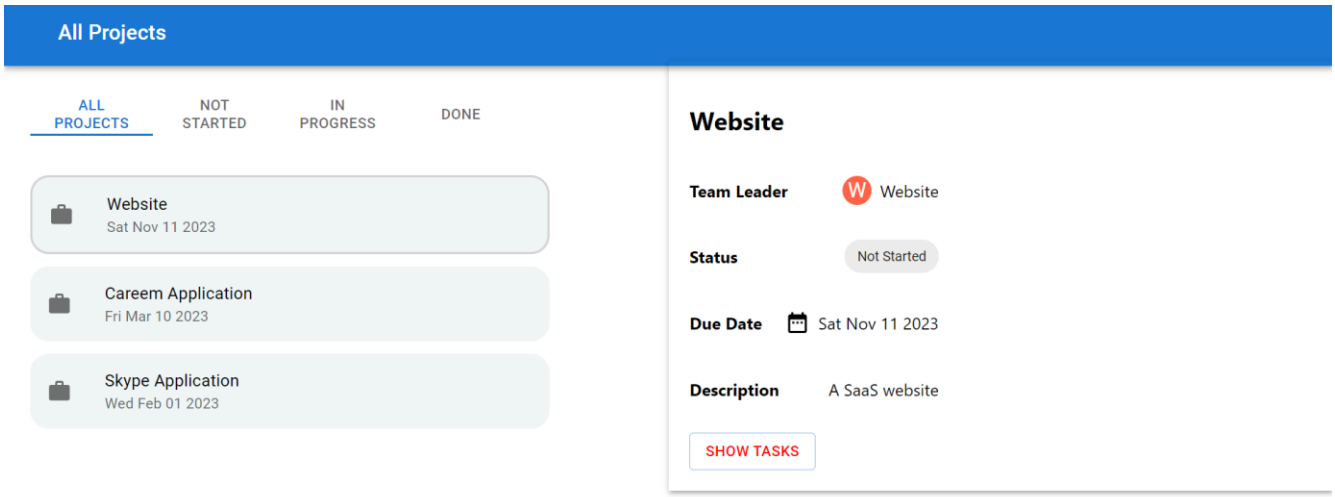
```
const { defineConfig } = require("cypress");
const getCompareSnapshotsPlugin = require('cypress-visual-regression/dist/plugin');

module.exports = defineConfig({
  env: {
    screenshotsFolder: './cypress/snapshots/actual',
    trashAssetsBeforeRuns: true,
    video: false
  },
  e2e: {
    setupNodeEvents(on, config) {
      getCompareSnapshotsPlugin(on, config);
    },
  },
});
```

c. Add the following to cypress/support/commands.js:

```
const compareSnapshotCommand = require('cypress-visual-regression/dist/command');
compareSnapshotCommand();
```

ii. Visual Testing of a Webpage:

This demonstration uses the following webpage to explain visual testing using cypress-visual-regression plugin.



iii. Test Cases

TC-1: Compare snapshots of the complete webpage

```
it.only("Should match previous screenshot of Page", () => {
  Cypress.Screenshot.defaults({
    onBeforeScreenshot($el) {
      const $foundElement = $el.find('[accesskey="detailsContainer"] div');
      if ($foundElement) {
        $foundElement.css("visibility", "hidden");
      }
    },
    onAfterScreenshot($el, props) {
      const $foundElement = $el.find('[accesskey="detailsContainer"] div');
      if ($foundElement) {
        $foundElement.css("visibility", "visible");
      }
    },
  });
  cy.wait(5000);
  cy.document().compareSnapshot("All Projects Page", {
    capture: "viewport",
    errorThreshold: 0.0,
  });
});
```

iv. Cypress Screenshot API:

In the above test case, Cypress Screenshot API is used to set defaults for how screenshots are captured during test execution. Using Screenshot API, we can set defaults for taking screenshots for example which parts of the screen to capture or whether to disable JavaScript timers and CSS animations when taking the screenshot etc.

"onBeforeScreenShot" function is automatically invoked before capturing a screenshot during a test run and "onAfterScreenShot" is automatically invoked after a screenshot is captured. In the above test case, these functions are used to demonstrate how to toggle the visibility of specific sections (dynamic content, counters, and animations etc.) of the webpage before and after capturing the screenshot.

v. Test Case Description:

The above test case selects the webpage as a DOM element using "document" function. Then using the "compareSnapshot" function provided by cypress-visual-regression plugin, the snapshot of the selected element is captured.

Note: In Cypress, screenshots are automatically captured only in "run" (headless) mode. In run/headless mode, tests are executed through command line rather than Cypress Launchpad.

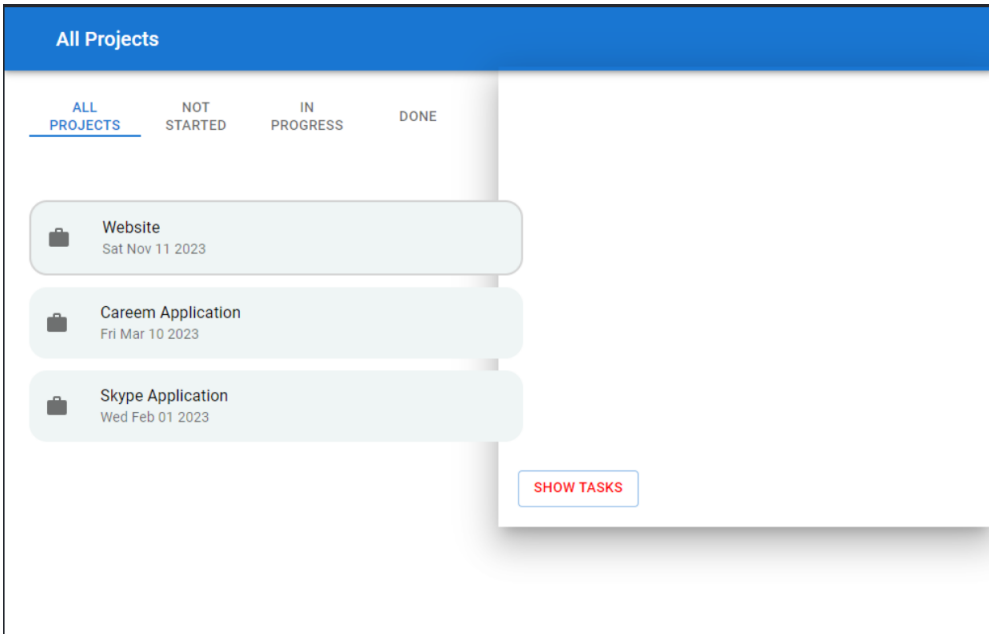
vi. Running Visual Tests:

a. Initial/baseline snapshot:

To take the baseline snapshot, type the following command in terminal:

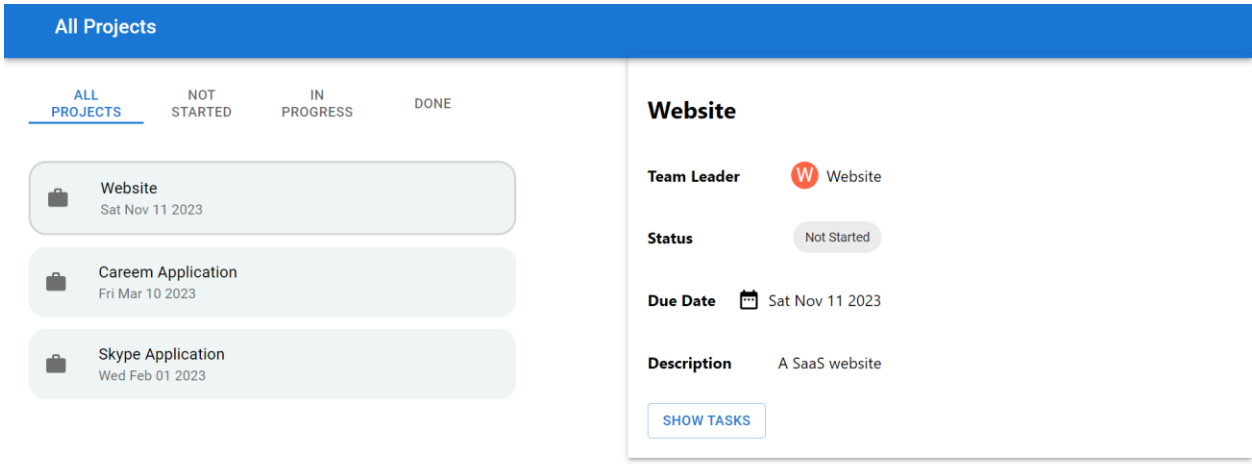
npx cypress run --spec "path to file" --env type=base

After test run is complete, baseline snapshot is saved in snapshots/base directory of the project.



b. Snapshot after code changes:

For demonstration purpose, the color of "Show Tasks" button is change from "red" to blue as shown in the image below:



To take the snapshot after code changes, type the following command in terminal:
npx cypress run --spec "path to file" --env type=actual

After test run is complete, new snapshot is saved in snapshots/diff directory of the project. This screenshot highlights the differences between the baseline snapshot and the snapshot after code changes.

TC-2: Compare snapshots of the specific element on the webpage:

Snapshots of specific elements on the webpage can also be compared to highlight any visual differences.

```
it("Should match previous screenshot of Button", () => {
  cy.visit("http://localhost:3000/")
    .get(
      "#root > div > div.MuiGrid2-root.MuiGrid2-container.MuiGrid2-direction-xs-row.
      MuiGrid2-grid-sm-12.css-1ohtbti-MuiGrid2-root > div.MuiGrid2-root.
      MuiGrid2-direction-xs-row.MuiGrid2-grid-sm-6.css-1n4wed4-MuiGrid2-root > a"
    )
    .compareSnapshot("Show_Tasks Button");
});
```

Baseline snapshot of selected element

Updated snapshot of selected element:

SHOW TASKS

Cypress Dashboard:

Cypress Dashboard is a cloud-based service that enables you to view and manage your Cypress test runs, including video recordings of your tests, network traffic, and test results. With the Cypress Dashboard, you can view the status of your test runs in real-time, share test results with your team, and analyze the results to identify and fix issues in your code.

i. Login in and Project Creation:

Create an account in https://cloud.cypress.io/login and create a new project. After creating or login an account, the project will be created by providing the name of project and project access i.e., public or private as shown below.

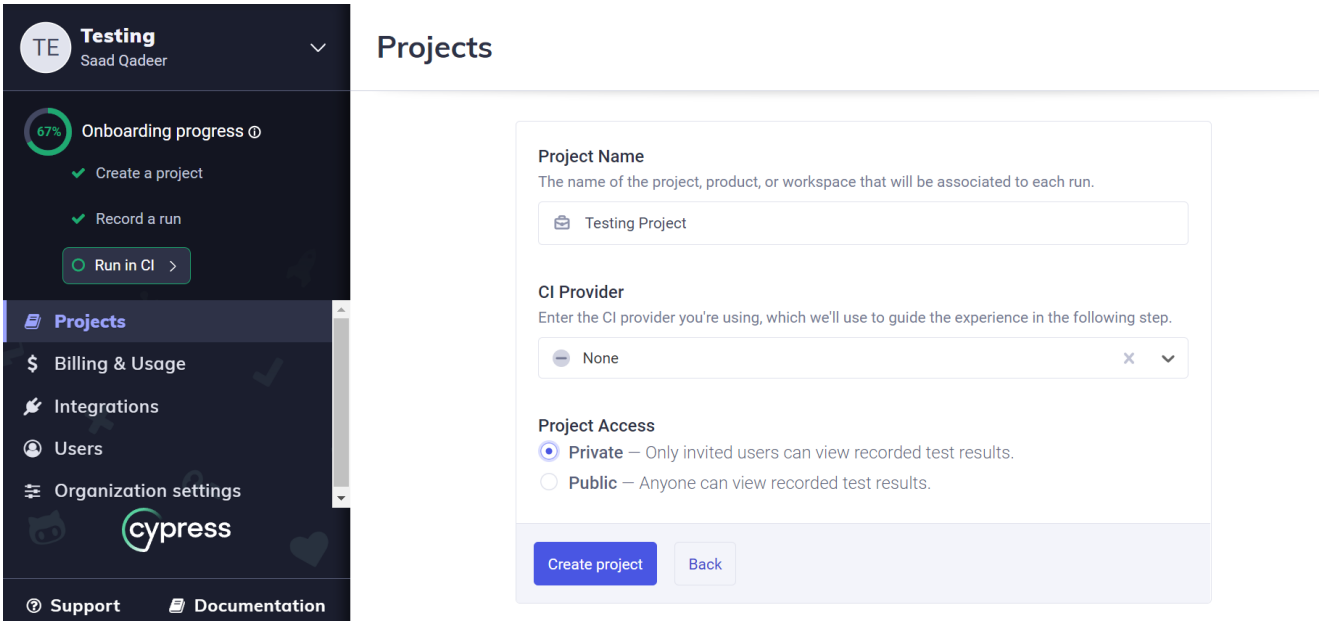


Figure 1: Shows the screen to create of new project

ii. Project Credentials: (Project Id and response Key)

After creation of project, the project id and response key will be generated which will be used in project to connect the project with cypress dashboard.

Figure 2: Shows the Project id and record key of project

The project id will be entered in cypress.config.js file in cypress project and then run the generated command in terminal as shown below.

Figure 3: Shows the pasting of projectId in cypress.config.js file

iii. Executing Test Cases:

Run the command on the terminal to execute the cypress test cases on cypress dashboard. The command is as follows:

npx cypress run –record –key <recordKey>

Record key is given to you by cypress dashboard on the result of creation of new project or you can simply copy the command and paste it in your terminal to execute the test cases on dashboard.

After running the command, the test file will execute and result of test suite will be displayed on terminal. It contains different information such as name of file containing set of test suits or test cases, total number of test cases, number of test cases which are passed, failed, pending and skipped and describes the time required to run the test suit.

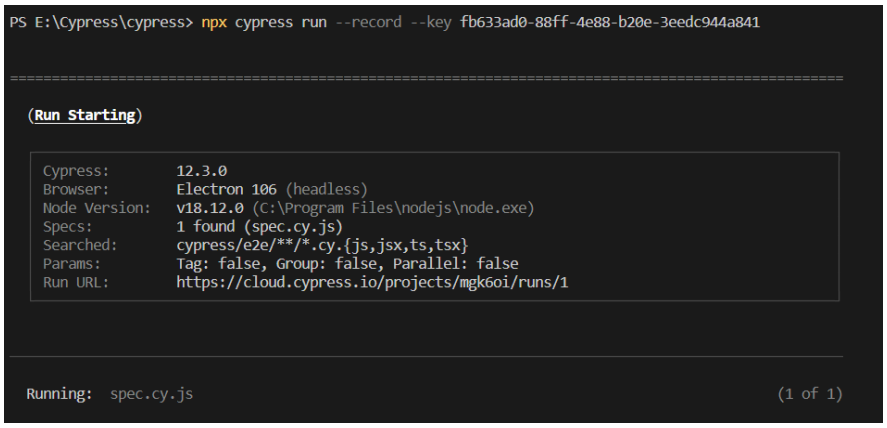


Figure 4: Shows the command to execute test cases on cypress cloud

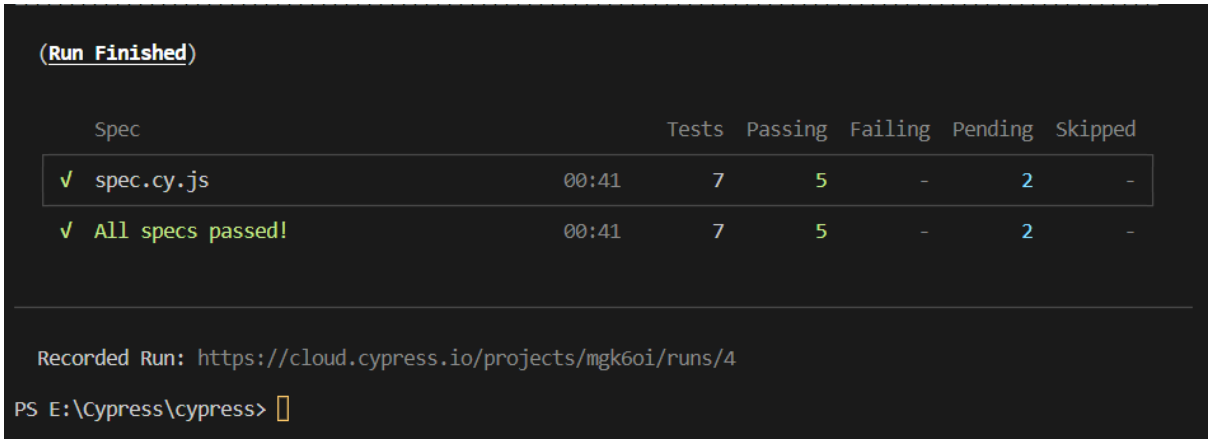


Figure 5: Shows the result of test cases on terminal

iv. Overview of Test Results in Cypress Dashboard:

Along with terminal, the test cases also run parallel on cypress dashboard and displays the overview of executed test cases such that which test cases are passed, failed, skipped running, timeout, over limit, errored etc. and filter the test cases on the basis of their status.

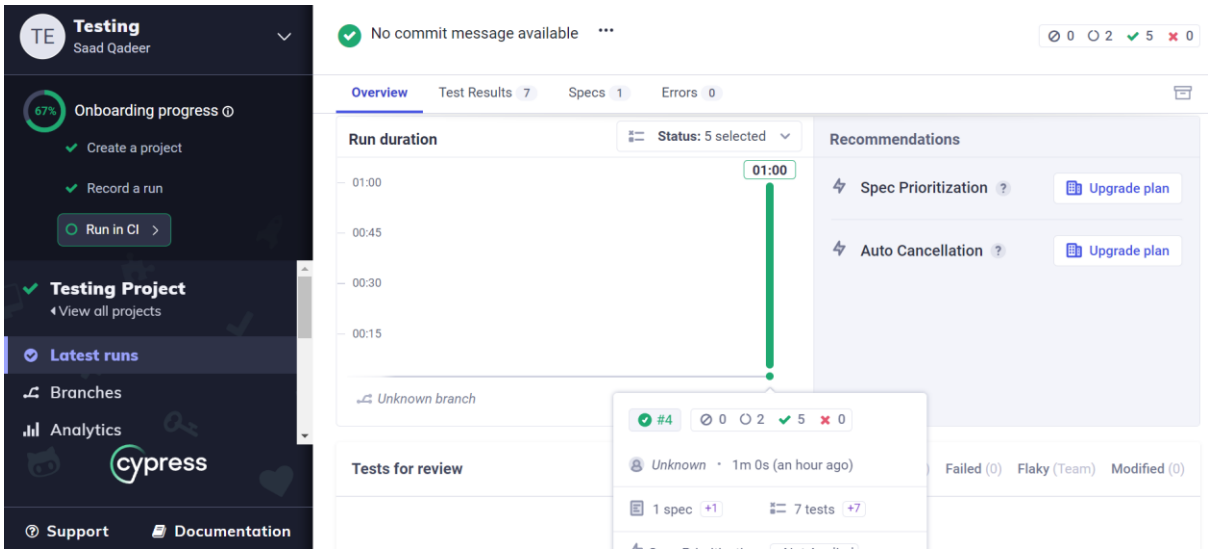


Figure 6: Shows the overview of test cases

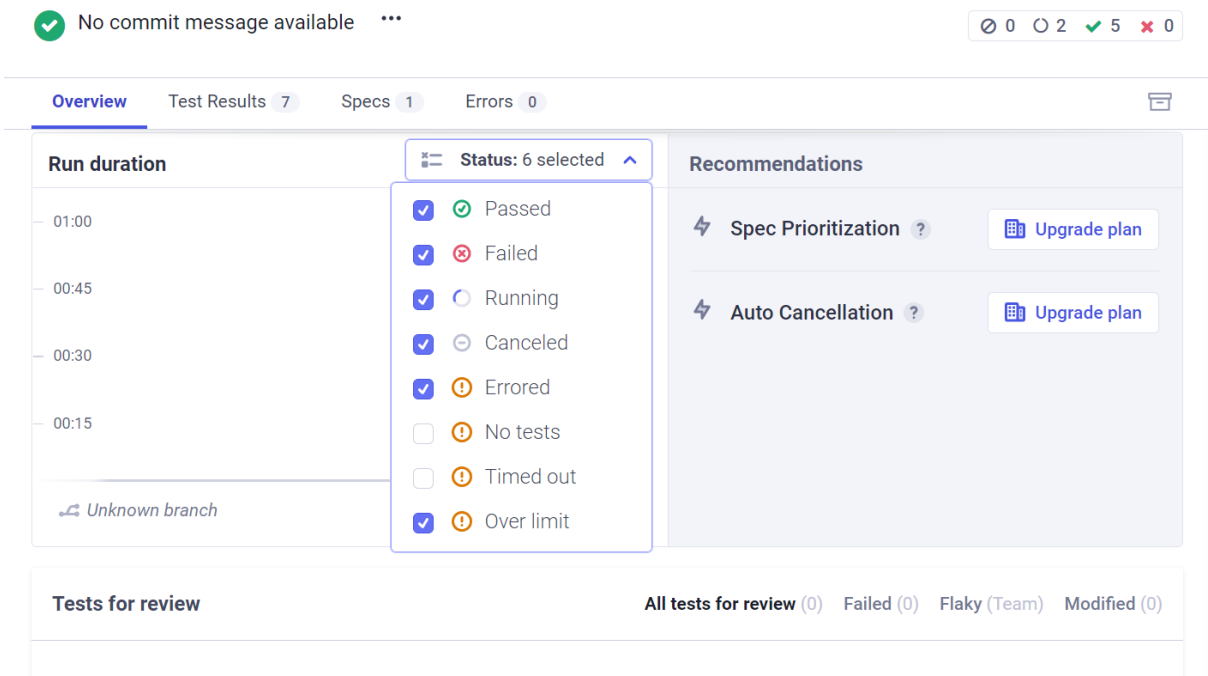


Figure 7: Shows the filtration of test cases on the basis of their status