

INTRODUCTION TO DEVOPS

S2-22_SESAPZG515

DONE BY

Krishna, Simran
2022SP93047

TABLE OF CONTENT

01 GIT

02 JENKINS

03 SELENIUM

04 SONARQUBE

05 KUBERNETES

Devops Assignment

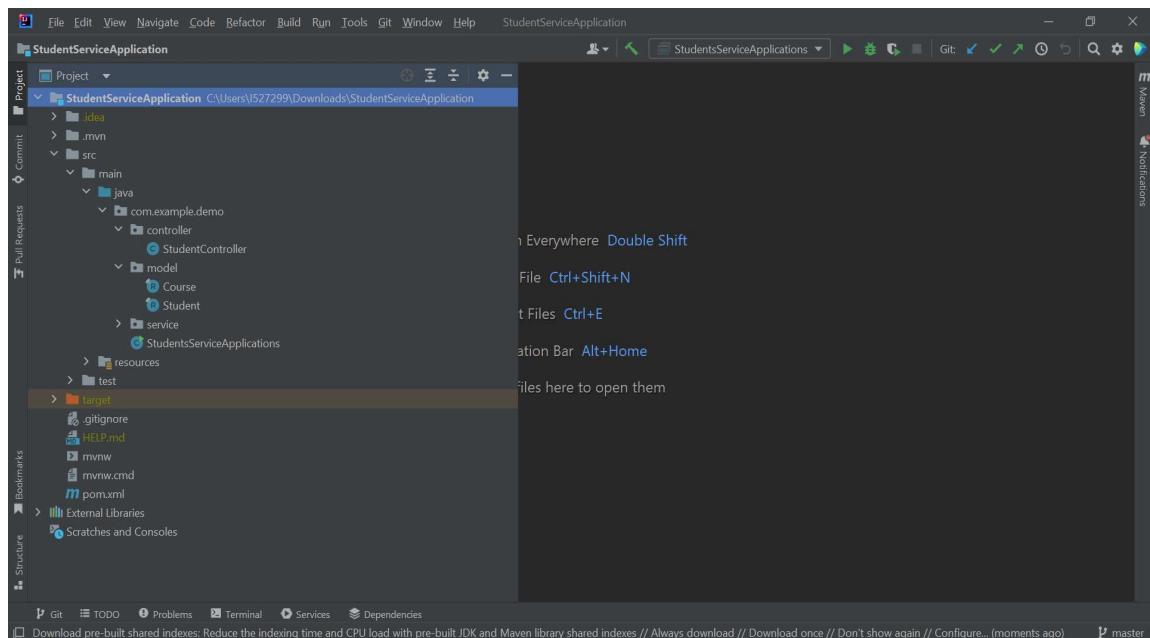
1. Git (3)

1. Get your project up and running locally

Project – Student Service Application

In this project, there are three services using proper URIs and HTTP methods:

- `@GetMapping("/students/{studentId}/courses")`: You can ask the courses a specific student has registered for using request method Get and example uri /students/Student1/courses.
- `@GetMapping("/students/{studentId}/courses/{courseId}")`: You can ask a specific course for a specific student using request method Get and example uri /students/Student1/courses/Course1.
- `@PostMapping("/students/{studentId}/courses")` : You can register a student for a course by sending a POST request to URI /students/Student1/courses



A few details:

- `StudentController.java` - Rest controller exposing all the three service methods discussed above.
- `Course.java, Student.java, StudentService.java` - Business Logic for the application. `StudentService` exposes a couple of methods we would consume from our Rest Controller.
- `StudentControllerIT.java` - Integration Tests for the Rest Services.
- `StudentControllerTest.java` - Unit Tests for the Rest Services.
- `StudentServicesApplication.java` - Launcher for the Spring Boot Application. To run the application, just launch this file as Java Application.
- `pom.xml` - Contains all the dependencies needed to build this project. We will use Spring Boot Starter Web.

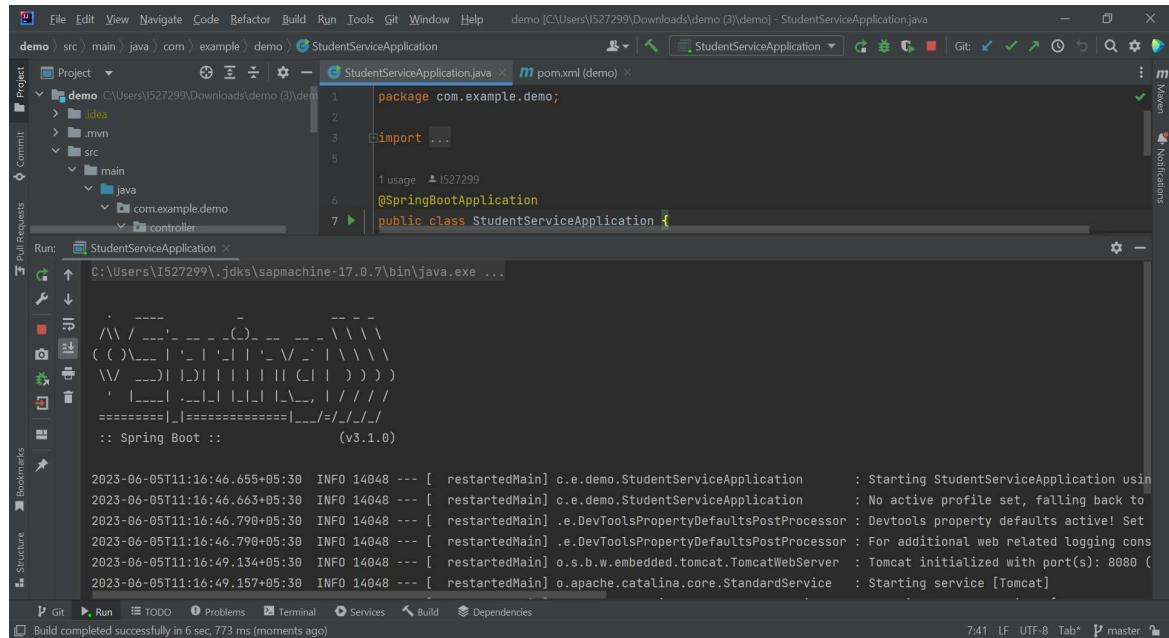
We have `StudentService` exposing methods to

- `public List<Student> retrieveAllStudents()` - Retrieve details for all students
- `public Student retrieveStudent(String studentId)` - Retrieve a specific student details
- `public List<Course> retrieveCourses(String studentId)` - Retrieve all courses a student is registered for
- `public Course retrieveCourse(String studentId, String courseId)` - Retrieve details of a specific course a student is registered for
- `public Course addCourse(String studentId, Course course)` - Add a course to an existing student

Executing the Http Get Operation Using Postman

We will access a request to `http://localhost:8080/students/Student1/courses/Course1` to test the service. And will receive the following response shown in the below.

Below picture shows how we can execute this Get Service from Postman

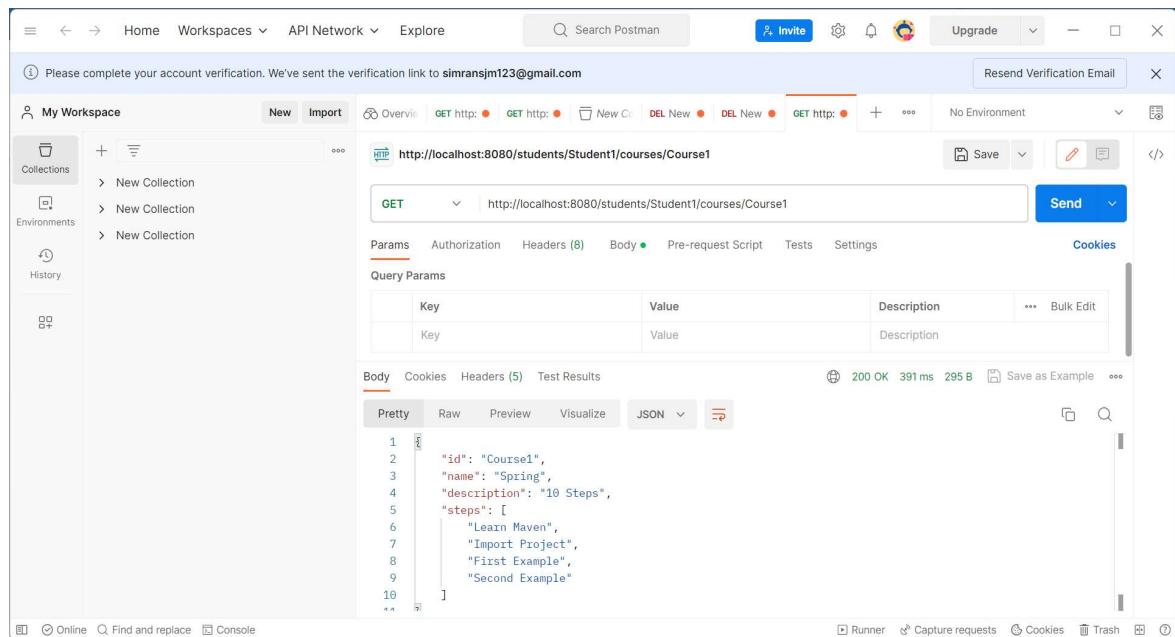


The screenshot shows the IntelliJ IDEA interface with the 'StudentServiceApplication.java' file open. The code defines a main class with a @SpringBootApplication annotation. Below the code editor is a terminal window displaying the application's startup logs. The logs show the application starting up, including the Spring Boot logo and various INFO messages about restarting the main application and initializing Tomcat.

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentServiceApplication {
```

```
2023-06-05T11:16:46.655+05:30 INFO 14048 --- [ restartedMain] c.e.demo.StudentServiceApplication      : Starting StudentServiceApplication using Java 17 on I527299 at 192.168.1.11:8080
2023-06-05T11:16:46.663+05:30 INFO 14048 --- [ restartedMain] c.e.demo.StudentServiceApplication      : No active profile set, falling back to 'auto-configuration'
2023-06-05T11:16:46.790+05:30 INFO 14048 --- [ restartedMain] o.e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'devtools.port' to override port
2023-06-05T11:16:46.790+05:30 INFO 14048 --- [ restartedMain] o.e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider adding 'logging.level.org.e...
```



The screenshot shows the Postman interface. A new collection named 'New Collection' is selected. A GET request is being prepared for the URL `http://localhost:8080/students/Student1/courses/Course1`. The 'Pretty' tab of the response body shows the JSON structure of the course data, which includes an ID, name, description, and a list of steps.

```
[{"id": "Course1", "name": "Spring", "description": "10 Steps", "steps": ["Learn Maven", "Import Project", "First Example", "Second Example"]}
```

Executing a Http POST Operation Rest Service

Example request is shown below. It contains all the details to register a course to a student.

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, API Network, and Explore. A search bar is present, along with options for Invite, Upgrade, and Resend Verification Email. The main workspace is titled 'New Collection / http://localhost:8080/api/scholars'. A POST request is selected, with the URL set to 'http://localhost:8080/students/Student1/courses'. The 'Body' tab is active, showing a JSON payload:

```
1 "name": "Microservices",
2 "description": "10 Steps",
3 "steps": [
4   "Learn How to Break Things Up",
5   "Automate the hell out of everything",
6   "Have fun"
7 ]
```

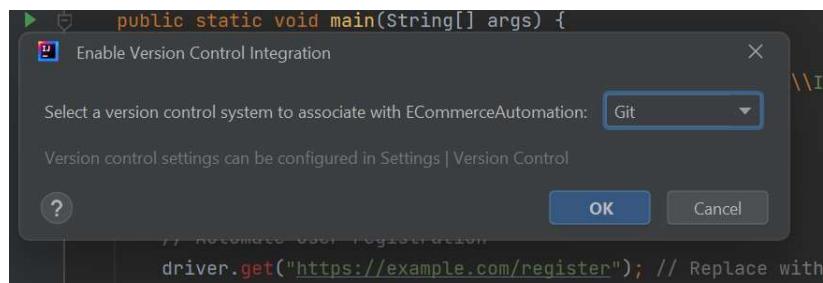
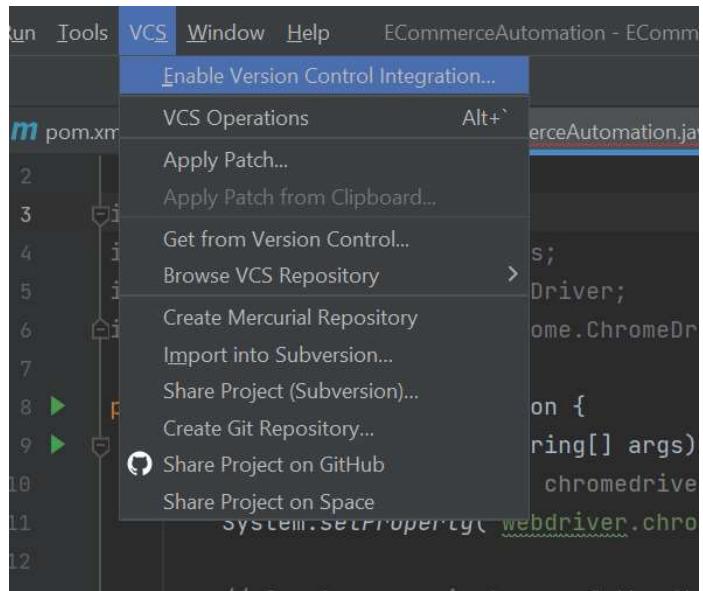
Below the body, the response status is 201 Created, with a duration of 139 ms and a size of 188 B. There are tabs for Pretty, Raw, Preview, Visualize, and Text. At the bottom, there are buttons for Runner, Capture requests, Cookies, and Trash.

PUSHED THE PROJECT TO GITHUB THROUGH INTELIJ

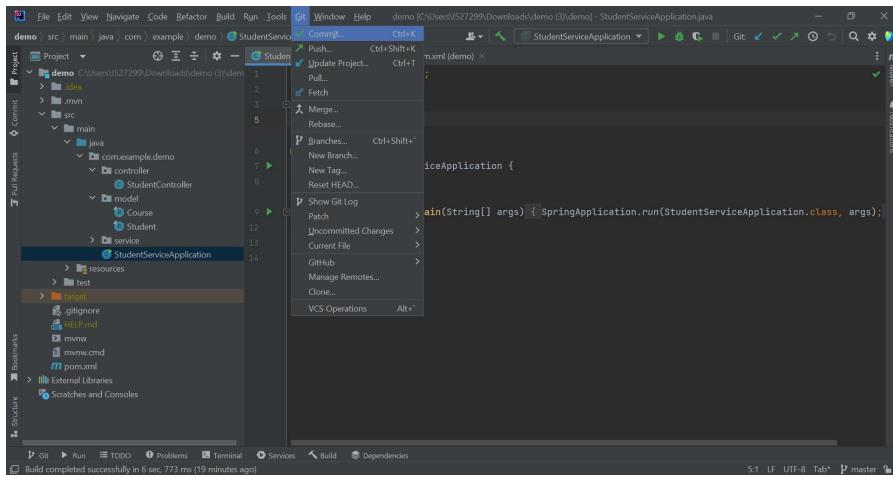
STEPS

To push your project to GitHub using IntelliJ without using the terminal, you can follow these steps:

1. Make sure you have a Git repository initialized in your project. You can do this by going to "VCS" in the top menu, selecting "Enable Version Control Integration," and choosing Git.

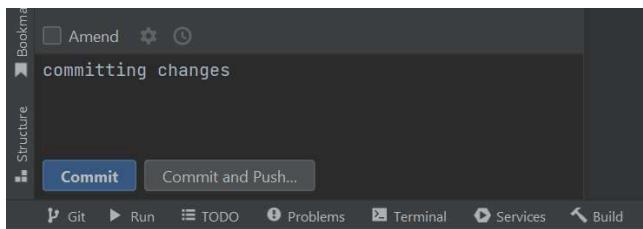


2. Once your project is under version control, go to the "Version Control" panel in IntelliJ. You can find it by going to "View" in the top menu, selecting "Tool Windows," and choosing "Version Control."
3. In the "Version Control" panel, you should see your changed files under the "Local Changes" tab. Select the files you want to commit and push to GitHub.
4. Right-click on the selected files and choose "Commit" from the context menu. Alternatively, you can click on the "Commit" button in the top toolbar.

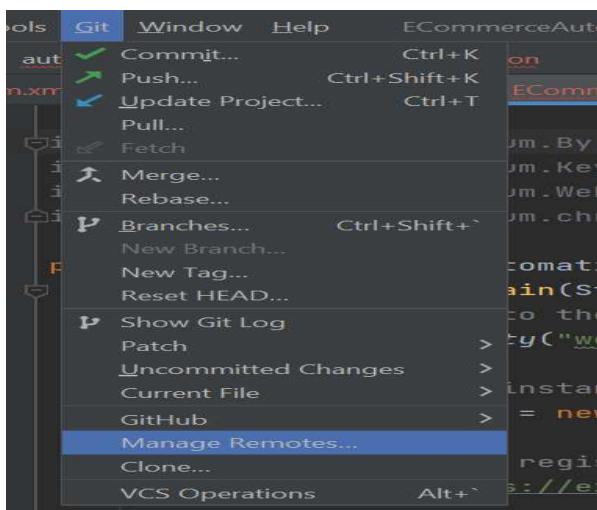


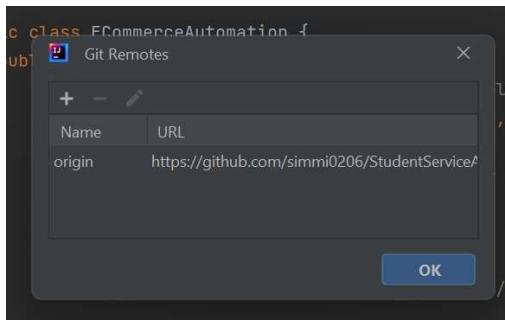
5. In the commit dialog, enter a commit message describing your changes. Make sure to provide a meaningful and concise message.

6. After entering the commit message, click on the "Commit and Push" button at the bottom of the dialog.

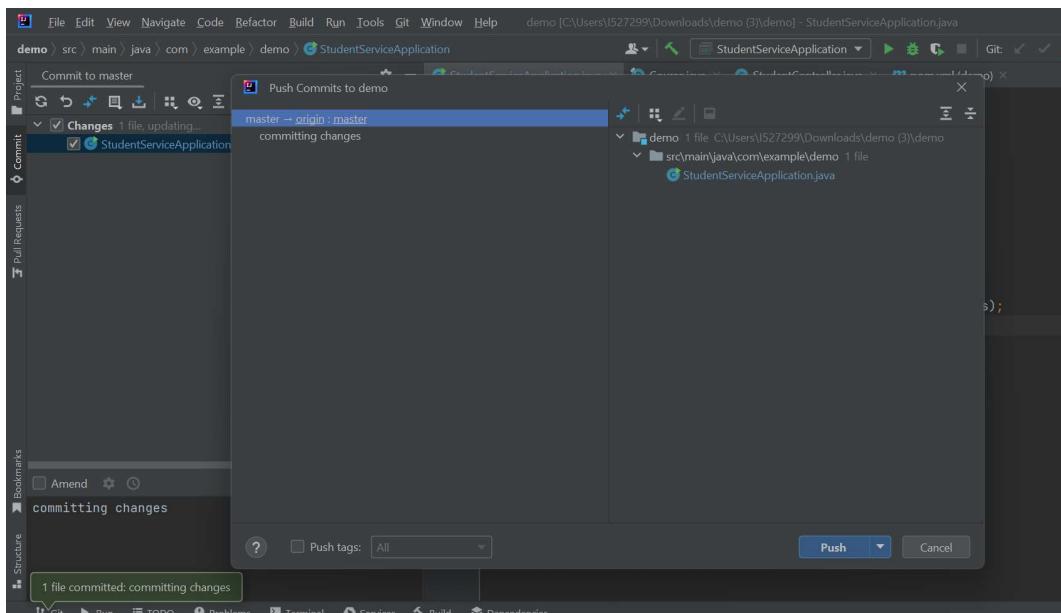


7. In the next dialog, select the remote repository where you want to push your changes. If you haven't added your GitHub account as a remote, click on the "+" button and enter the repository URL and your GitHub credentials.

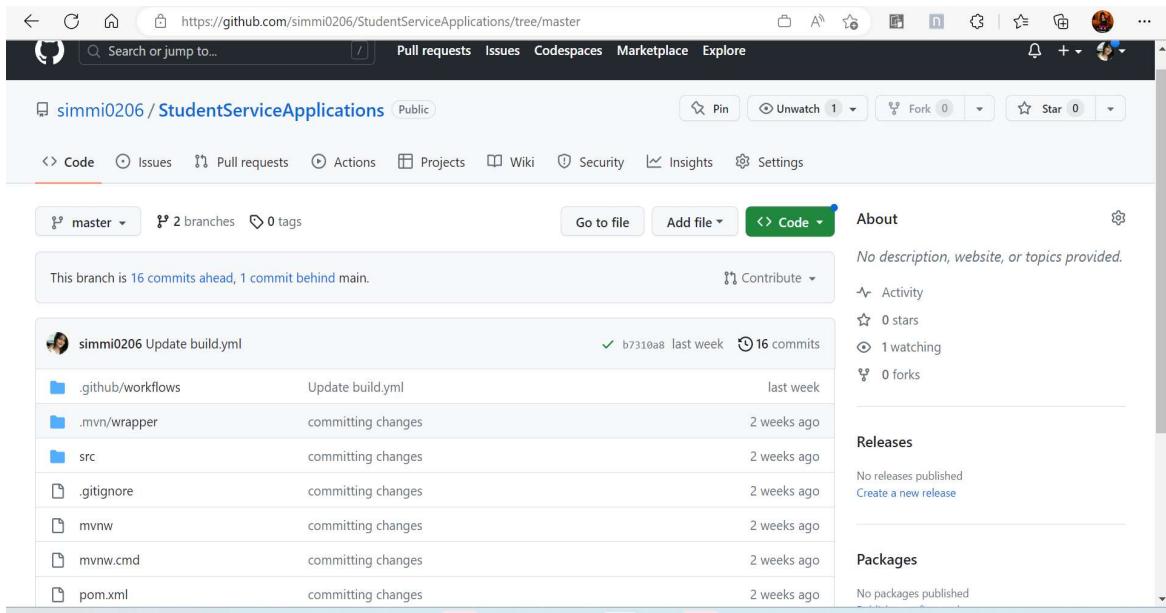




8. Once you have selected the remote repository, click on the "Push" button.



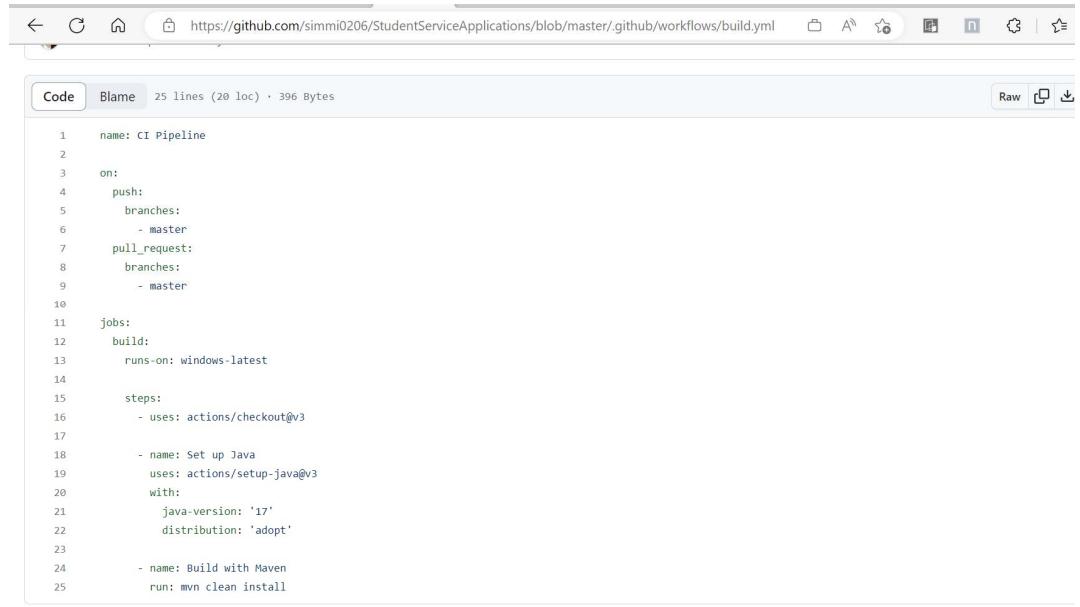
9. IntelliJ will now push your changes to the selected remote repository (in this case, GitHub).



Build an initial CI pipeline on Gitlab

To build an initial CI (Continuous Integration) pipeline on GitHub, following are the steps:

- 1. Create a GitHub repository:** Start by creating a new repository on GitHub, push code.
- 2. Choose GitHub Actions:** GitHub Actions is an integrated CI/CD platform provided by GitHub. It supports Windows environments and can be used to build CI pipelines.
- 3. Define the CI workflow:** In your repository, create a .github/workflows directory if it doesn't exist. Inside this directory, create a new YAML file (e.g., ci.yml) to define your CI workflow.
- 4. Configure workflow triggers:** Within the YAML file, define the triggers that will activate the CI pipeline. For example, you can set it to trigger on every push to the repository or only on specific branches



The screenshot shows a GitHub browser interface with the URL <https://github.com/simmi0206/StudentServiceApplications/blob/master/.github/workflows/build.yml>. The page displays a GitHub Actions workflow file named 'build.yml'. The code editor shows the following YAML configuration:

```
name: CI Pipeline
on:
  push:
    branches:
      - master
  pull_request:
    branches:
      - master
jobs:
  build:
    runs-on: windows-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Java
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'adopt'
      - name: Build with Maven
        run: mvn clean install
```

5. Specify the workflow steps: Define the steps required to build and test your Java Maven project. Here's an example workflow that builds the project using Maven. In this example, the workflow checks out the code, sets up Java 19 using the actions/setup-java action, and then builds and tests the project using Maven.

6. Commit and push the YAML file: Once you've defined the workflow, commit and push the YAML file to the .github/workflows directory in your repository. GitHub Actions will automatically detect the new workflow and start executing it based on the defined triggers.

7. Monitor the workflow: You can monitor the progress of your workflow by visiting the "Actions" tab in your GitHub repository. It will show you the status of each workflow run, including any errors or failures.

https://github.com/simmi0206/StudentServiceApplications/actions

simmi0206 / StudentServiceApplications Public

Actions

All workflows

Showing runs from all workflows

2 workflow runs

Event Status Branch Actor

Update build.yml CI Pipeline #16: Commit b7310a8 pushed by simmi0206 master last week 1m 48s

Update build.yml CI Pipeline #12: Commit 4162e54 pushed by simmi0206 master 2 weeks ago 1m 15s

simmi0206 / StudentServiceApplications Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

CI Pipeline

Update build.yml #16

Re-run all jobs ...

Summary

Triggered via push last week simmi0206 pushed -> b7310a8 master Status Success Total duration 1m 48s Artifacts -

Jobs

build

Run details

Usage

Workflow file

build.yml
on: push

build 1m 35s

- +

2. Maven and Jenkins

a. Setup a Jenkins Job with Apache Maven

To set up a Jenkins job with Apache Maven, steps are Following:

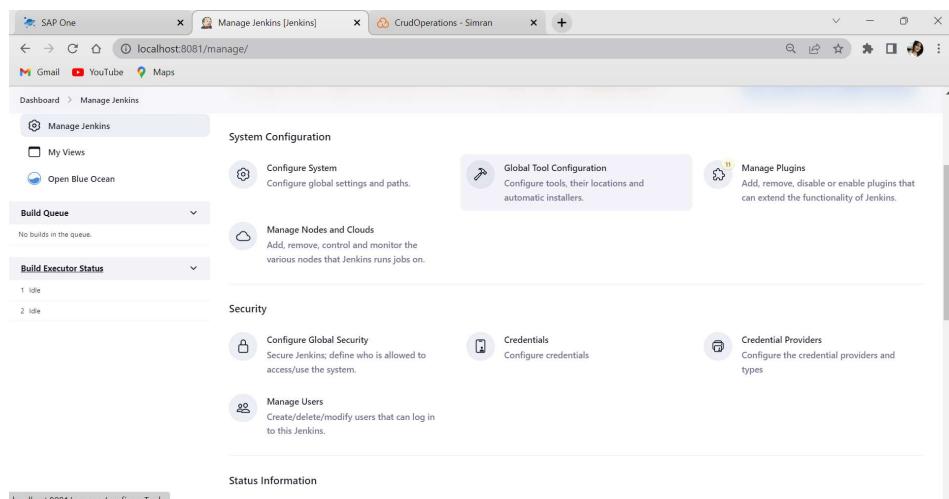
1. Install Jenkins: Download and install Jenkins on your server or local machine. Visit the official Jenkins website for installation instructions: <https://www.jenkins.io/download/>

2. Configure Jenkins:

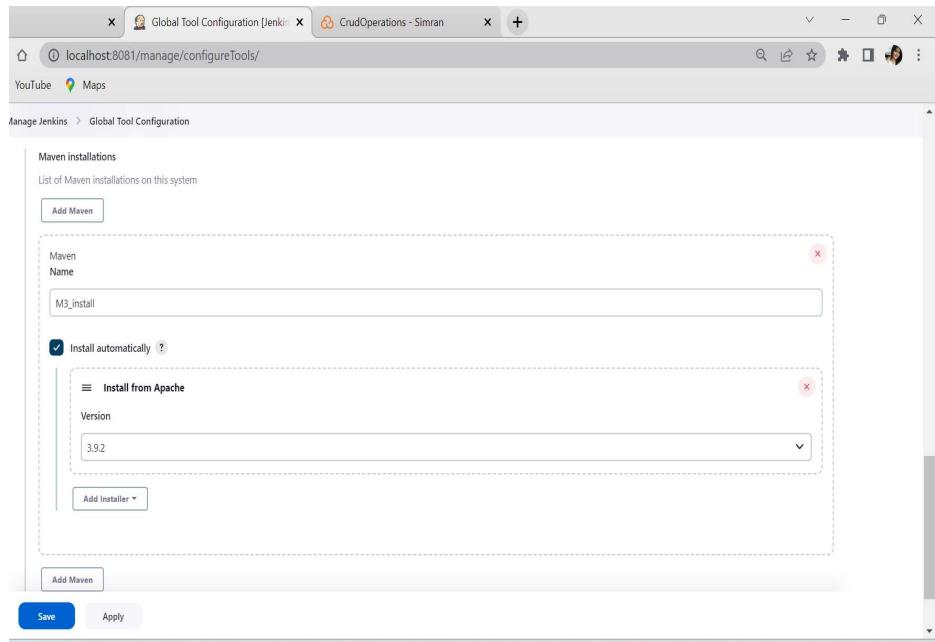
- Access Jenkins through your web browser using the URL: `http://localhost:8080` (if running locally) or the appropriate server address.
- Complete the initial setup by following the on-screen instructions, including creating an admin user and installing recommended plugins.

3. Install Maven on Jenkins: Open Jenkins and navigate to "Manage Jenkins" from the main dashboard.

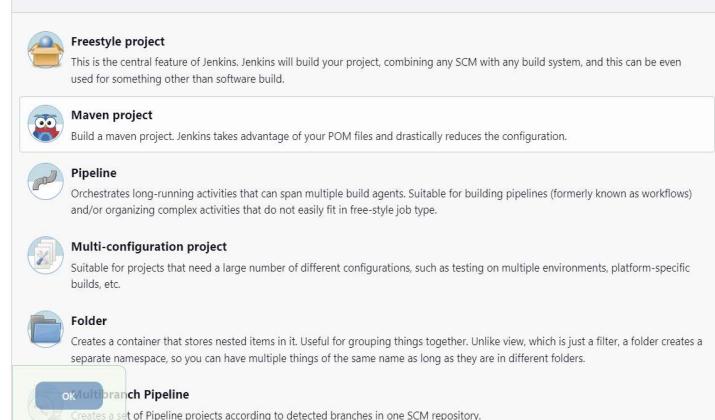
- Select "Global Tool Configuration" to manage Jenkins tools.



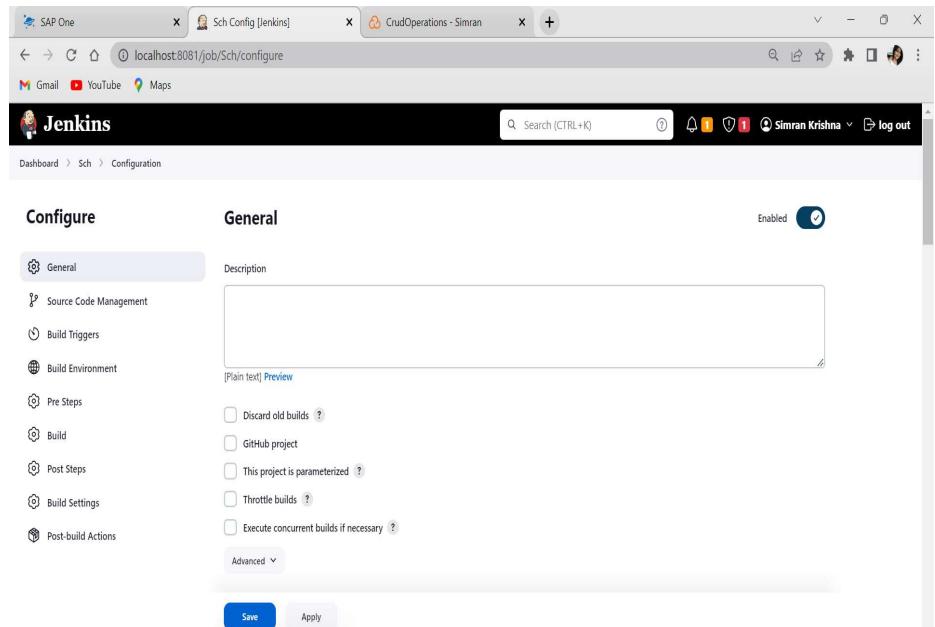
- Scroll down to the "Maven" section and click on "Add Maven."
- Provide a name for the Maven installation and select the Maven version to install.



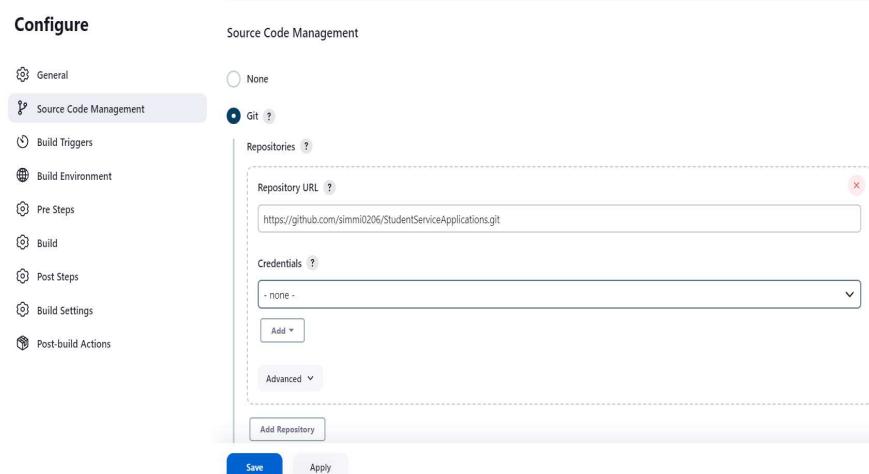
- Save the configuration.
4. Create a new Jenkins job:
- From the Jenkins main dashboard, click on "New Item" to create a new job.
 - Enter a name for your job and select the "Maven Project" option.



- From the Jenkins main dashboard, click on "New Item" to create a new job.
- Enter a name for your job and select the "Maven Project" option.
- Click on "OK" to proceed.
- Configure the Jenkins job:

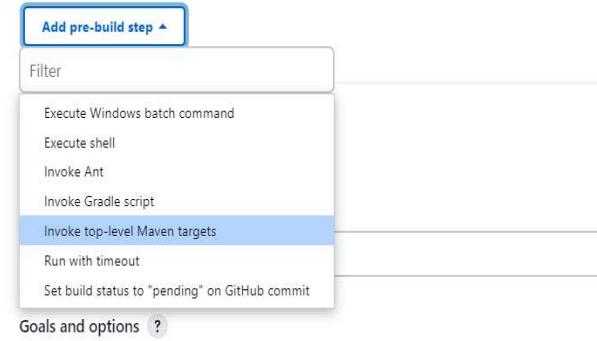


5. Job configuration page → Source Code Management → Add git repository https or SSH link.



6. Under the "Build" section, click on "Add build step" and select "Invoke top-level Maven targets."

Pre Steps



A screenshot of the Jenkins configuration interface showing the "Pre Steps" section. A dropdown menu is open under the "Add pre-build step" button, listing several options: Execute Windows batch command, Execute shell, Invoke Ant, Invoke Gradle script, Invoke top-level Maven targets, Run with timeout, and Set build status to "pending" on GitHub commit. The "Invoke top-level Maven targets" option is highlighted with a blue selection bar.

7. In the "Goals" field, specify the Maven goals you want to execute (e.g., clean install, test, package, etc.).



A screenshot of the Jenkins configuration interface showing the "Goals and options" section. A text input field contains the value "clean install".

8. Set additional Maven options or specify the path to the Maven installation if different from the default.

9. Save the configuration.

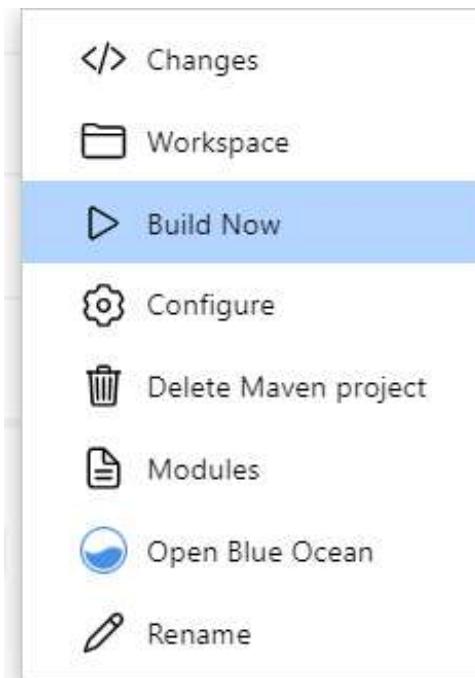
Add post-build action ▾

Save

App

10. Build the Jenkins job:

11. On the main job dashboard, click on "Build Now" to initiate a build. (here Application is Build)



The image shows the Jenkins Dashboard at localhost:8081. The dashboard includes:

- A sidebar with links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, and My Views.
- A main area showing the build history for two projects:

S	W	Name	Last Success	Last Failure	Last Duration
Green checkmark	Cloud icon	StudentServiceApplication	8 days 21 hr #7	8 days 21 hr #6	2 min 24 sec
Green checkmark	Cloud icon	StudentServiceApplications	8 days 13 hr #9	8 days 13 hr #5	0.21 sec
- Build Queue: No builds in the queue.
- Build Executor Status: 1 Idle, 2 Idle.
- Icons for S, W, M, L.
- Atom feed links: Atom feed for all, Atom feed for failures, Atom feed for just latest builds.
- REST API and Jenkins 2.387.3 links.

12. The console output will display the build progress, including any errors or warnings encountered

```

t:8081/job/StudentServiceApplication/7/console

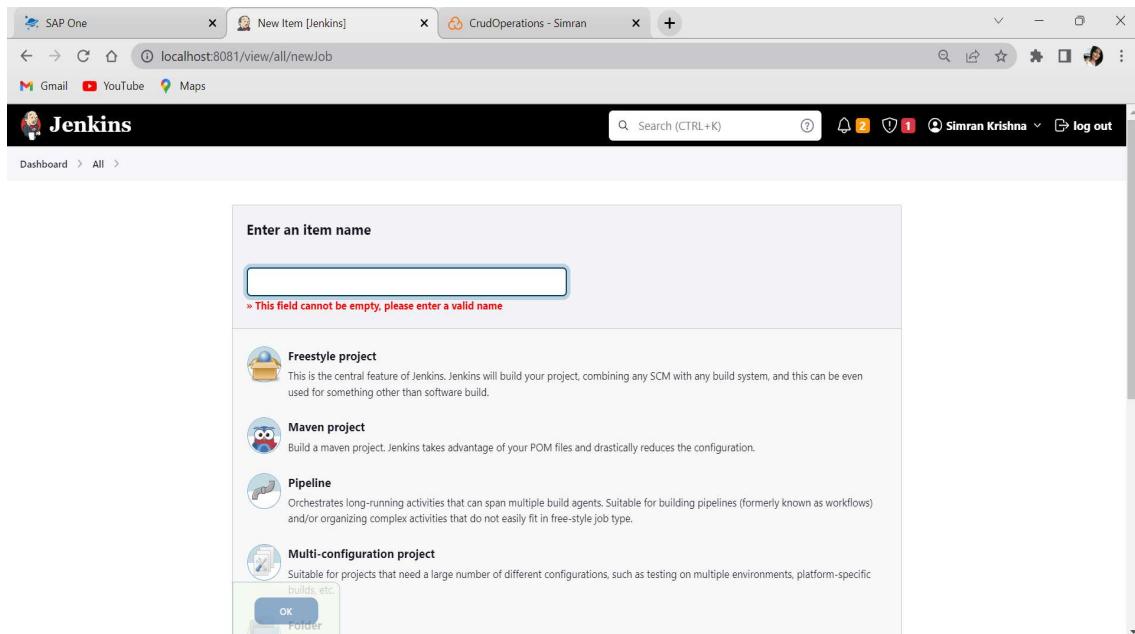
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-lang3/3.7/commons-lang3-3.7.jar (500 kB at 139 kB/s)
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/28.2-android/guava-28.2-android.jar (2.6 MB at 624 kB/s)
[INFO] Replacing main artifact C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\target\demo-0.0.1-SNAPSHOT.jar with repackaged archive, adding nested dependencies in BOOT-INF/
[INFO] The original artifact has been renamed to C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\target\demo-0.0.1-SNAPSHOT.jar.original
[INFO]
[INFO] --- install:3.1.1:install (default-install) @ demo ---
[INFO] Installing C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\pom.xml to C:\Windows\system32\config\systemprofile\.m2\repository\com\example\demo\0.0.1-SNAPSHOT\demo-0.0.1-SNAPSHOT.pom
[INFO] Installing C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\target\demo-0.0.1-SNAPSHOT.jar to C:\Windows\system32\config\systemprofile\.m2\repository\com\example\demo\0.0.1-SNAPSHOT\demo-0.0.1-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:43 min
[INFO] Finished at: 2023-05-27T15:36:43+05:30
[INFO] -----
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\pom.xml to com.example.demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.pom
[JENKINS] Archiving C:\ProgramData\Jenkins\jenkins\workspace\StudentServiceApplication\target\demo-0.0.1-SNAPSHOT.jar to com.example.demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.jar
channel stopped
Finished: SUCCESS

```

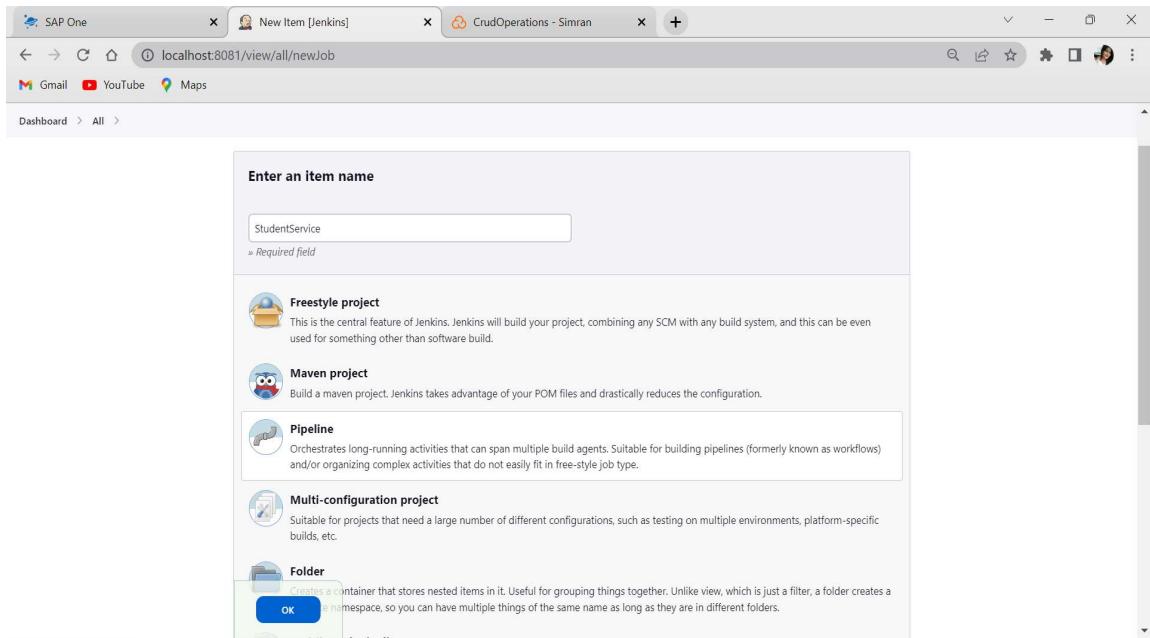
b. Setup a Jenkins build Pipeline with Build, Test, Package, Deploy

Step 1: Create a new Jenkins Pipeline job:

- From the Jenkins main dashboard, click on "New Item" to create a new job.



- Enter a name for your pipeline job and select the "Pipeline" option.



- Click on "OK" to proceed.

Step 2: Configure the Jenkins Pipeline job:

The screenshot shows the Jenkins pipeline configuration page for the 'StudentService' job. The 'General' tab is selected. In the 'Description' section, there is a text input field with a placeholder '[Plain text] Preview'. Below the description, there is a list of checkboxes for pipeline options:

- Discard old builds
- Do not allow concurrent builds
- Do not allow the pipeline to resume if the controller restarts
- GitHub project
- Pipeline speed/durability override
- This project is parameterized
- Throttle builds

At the bottom of the configuration page are two buttons: 'Save' and 'Apply'.

- In the pipeline configuration page, scroll down to the "Pipeline" section.
- In the "Definition" dropdown, select "Pipeline script"

The screenshot shows the Jenkins Pipeline configuration page for a job named "StudentService". The "Pipeline" tab is selected. In the "Script" area, a Groovy pipeline script is displayed:

```
1 pipeline {  
2     agent any  
3     stages {  
4         stage('Clone Repository') {  
5             steps {  
6                 git branch: 'master', url: 'https://github.com/simmi0206/StudentServiceApplications.git'  
7             }  
8         }  
9         stage('Build') {  
10            steps {  
11                // Maven build command  
12                bat 'mvn clean install'  
13            }  
14        }  
15        stage('Test') {  
16            steps {  
17                // Maven test command  
18            }  
19        }  
20    }  
21}
```

Below the script, there is a checkbox labeled "Use Groovy Sandbox" which is checked. At the bottom of the page are "Save" and "Apply" buttons.

- In the script area, you can define your pipeline stages and steps. Use the following example script as a starting point

Step 3: Save the configuration.

Step 4: Run the Jenkins Pipeline job:

- On the main job dashboard, click on "Build Now" to start the pipeline.

The screenshot shows the Jenkins job dashboard for "StudentService". The "Build Now" button is highlighted. Other options visible include "Status", "Changes", "Configure", "Delete Pipeline", "Full Stage View", "Rename", and "Pipeline Syntax".

Status

</> Changes

▷ Build Now

⚙ Configure

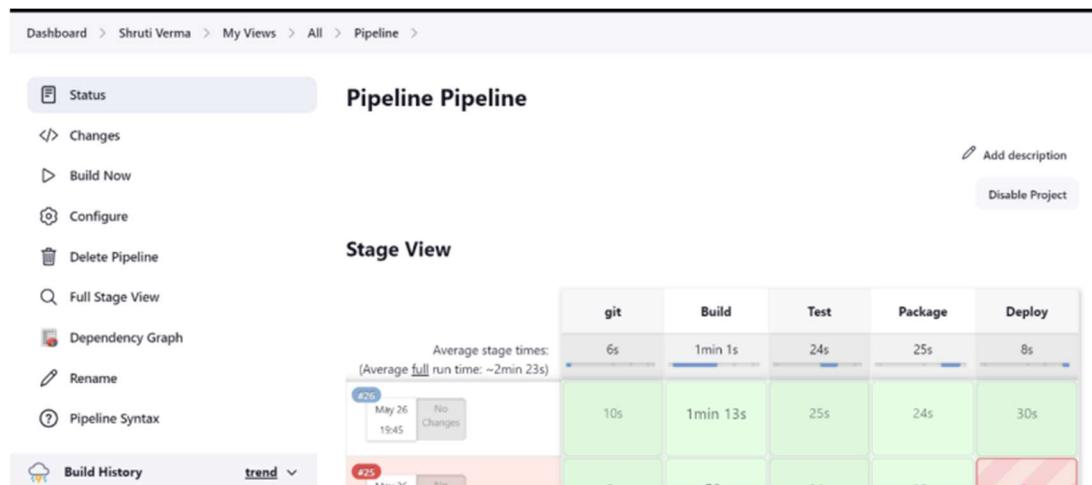
Delete Pipeline

🔍 Full Stage View

✍ Rename

ⓘ Pipeline Syntax

- Jenkins will execute each stage sequentially, running the defined steps within each stage.
- The console output will display the progress of each stage and any errors or warnings encountered



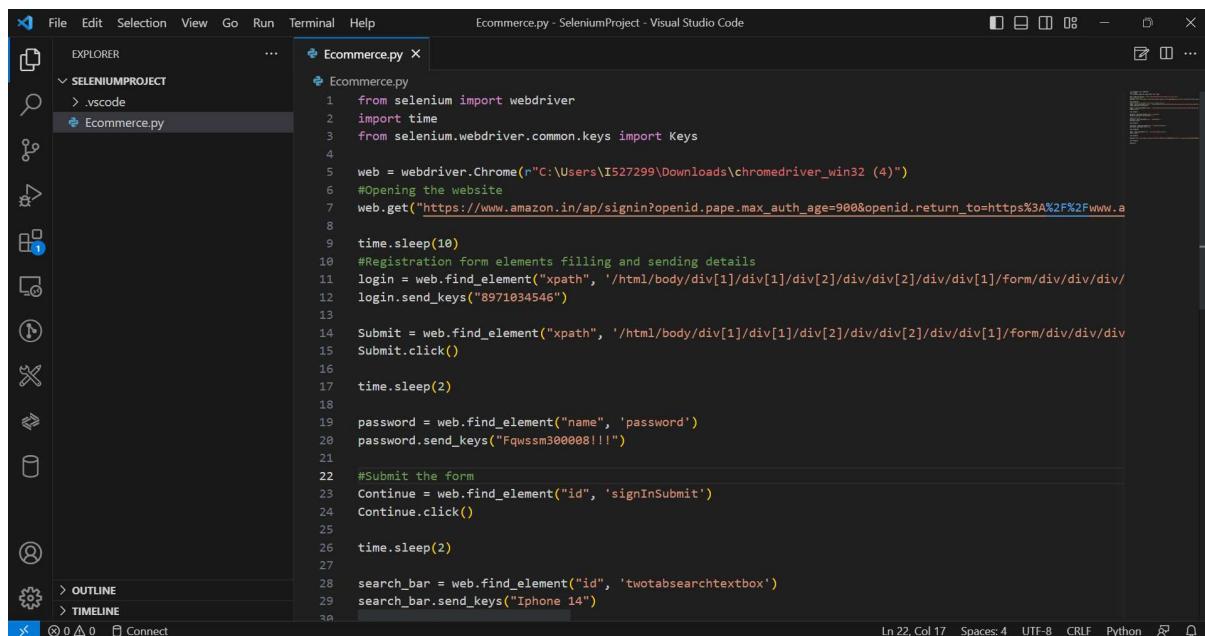
3.Selenium, SonarQube (3)

1. Automate user registration & search product feature of e-commerce website with selenium

Automating user registration and search product features of an e-commerce website using Selenium can be done by following these steps:

1. Install Selenium WebDriver: Download the latest version of Selenium WebDriver for your preferred programming language and install it.
2. Install selenium in CMD: pip install selenium
3. Set up the development environment: Create a new project in your preferred integrated development environment (IDE) and import the Selenium WebDriver library.
4. Open the website: Use the WebDriver to open the e-commerce website in a browser.
5. Automate user registration: Find the registration form elements using the WebDriver and fill them out with user information. Then submit the form to complete the registration process.

Here is an example code snippet for registering a new user:



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:** The file "Ecommerce.py" is open, showing Python code for automating user registration. The code imports webdriver, time, and Keys from selenium, initializes a Chrome driver, opens the Amazon sign-in page, fills in login and password fields, and submits the form.
- Explorer Panel:** Shows a project structure with "SELENIUMPROJECT" containing ".vscode" and "Ecommerce.py".
- Bottom Status Bar:** Shows "Ln 22, Col 17" and other status indicators like "Spaces: 4", "UTF-8", "CRLF", "Python".

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a project named 'SELENIUMPROJECT' containing files '.vscode' and 'Ecommerce.py'. The 'Ecommerce.py' file is open in the main editor area, displaying Python code for a Selenium script. Below the editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'SQL CONSOLE'. The 'TERMINAL' tab is active, showing a command prompt window with the command 'PS C:\Users\I527299\Downloads\SeleniumProject> python Ecommerce.py'. To the right of the terminal, there is a 'Terminal' sidebar with three entries: 'powershell', 'powershell', and 'powershell'. The status bar at the bottom shows 'Ln 22, Col 17' and other system information.

```
File Edit Selection View Go Run Terminal Help Ecommerce.py - SeleniumProject - Visual Studio Code
EXPLORER ... Ecommerce.py
SELENIUMPROJECT .vscode Ecommerce.py
from selenium import webdriver
import time
from selenium.webdriver.common.keys import Keys
web = webdriver.Chrome(r"C:\Users\I527299\Downloads\chromedriver_win32 (4)")
#Opening the website
web.get("https://www.amazon.in/ap/signin?openid.pape.max_auth_age=900&openid.return_to=https%3A%2F%2Fwww.a
time.sleep(10)
#Registration form elements filling and sending details
login = web.find_element("xpath", '/html/body/div[1]/div[1]/div[2]/div/div[1]/form/div/div/div[1]
login.send_keys("8971034546")
Submit = web.find_element("xpath", '/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div[1]/form/div/div/div[1]
Submit.click()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE

PS C:\Users\I527299\Downloads\SeleniumProject> python Ecommerce.py

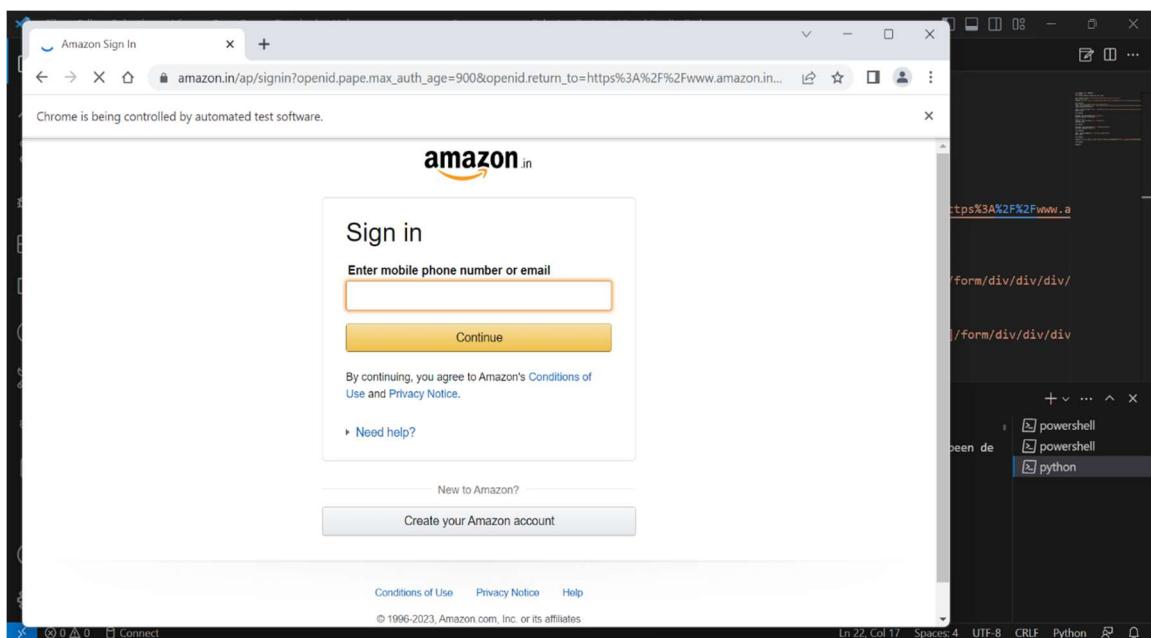
+ ... ^ x

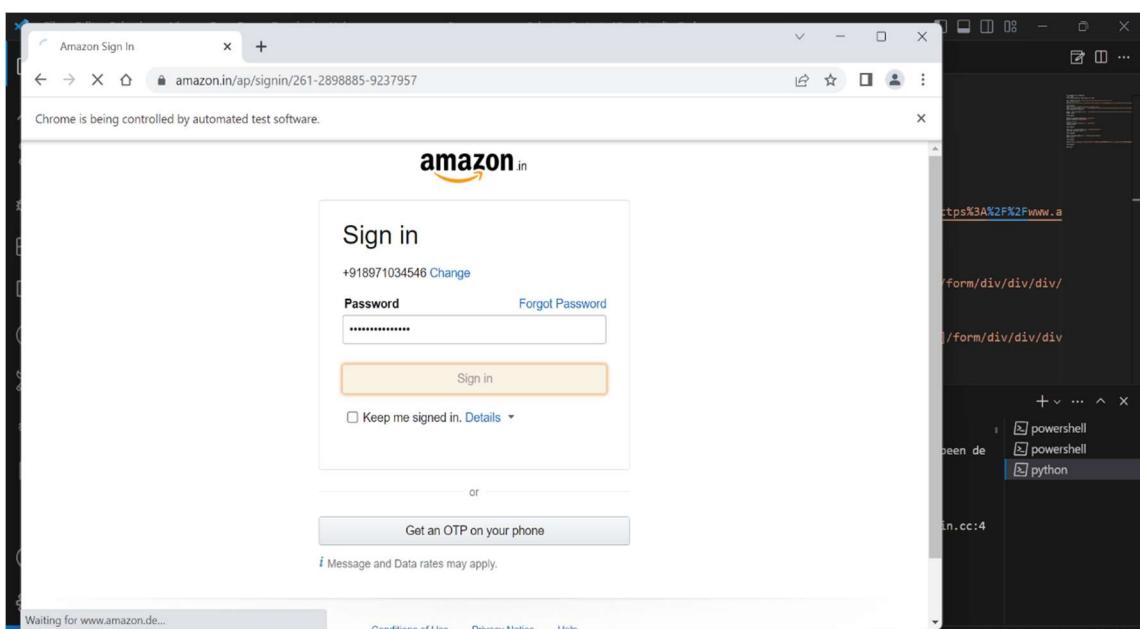
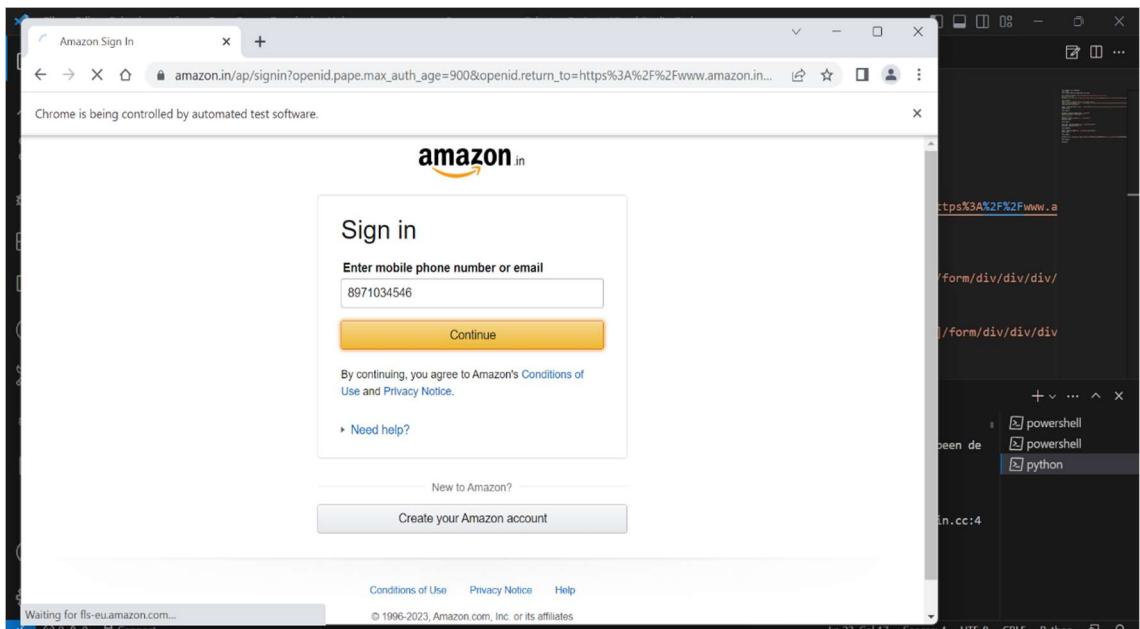
powershell
powershell
powershell

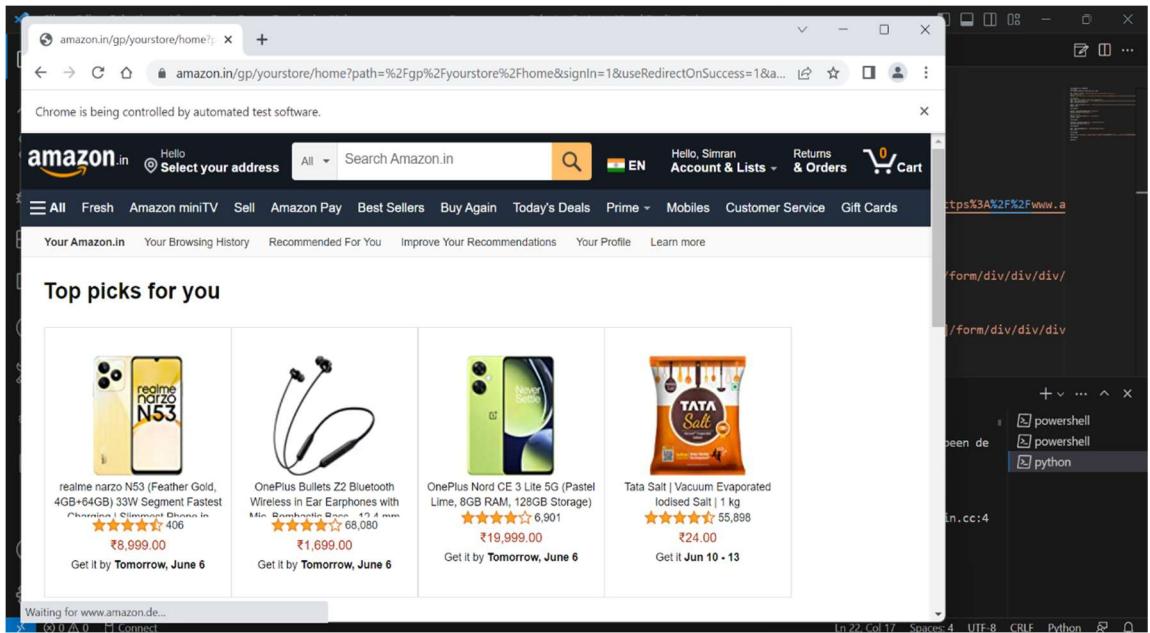
Ln 22, Col 17 Spaces: 4 UTF-8 CRLF Python

Output:

1. Registering on Amazon, automatically details are filled.





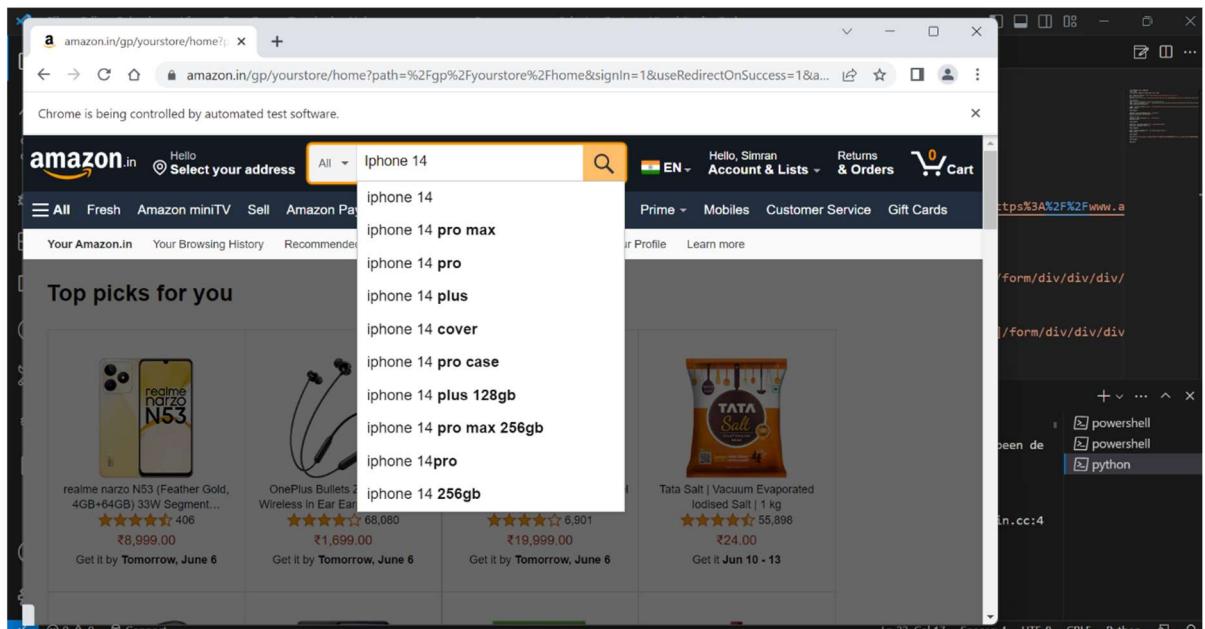


Automate product search: Find the search bar element using the WebDriver and enter the product name or keywords. Then, submit the search query to display the search results.

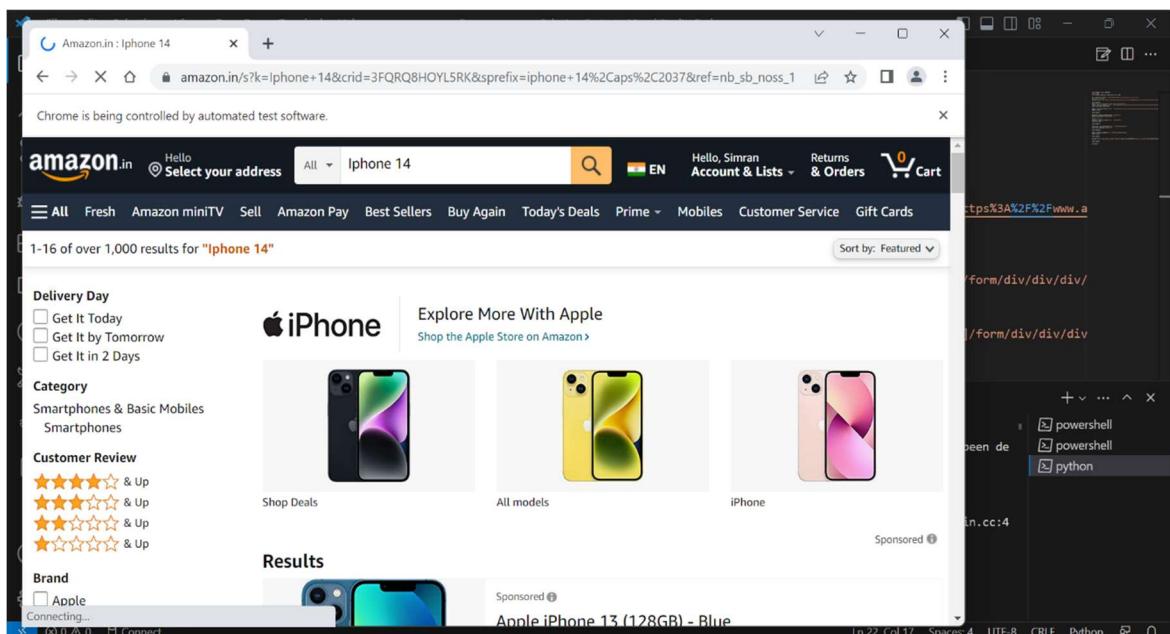
Here is a code snippet for searching a product

Output:

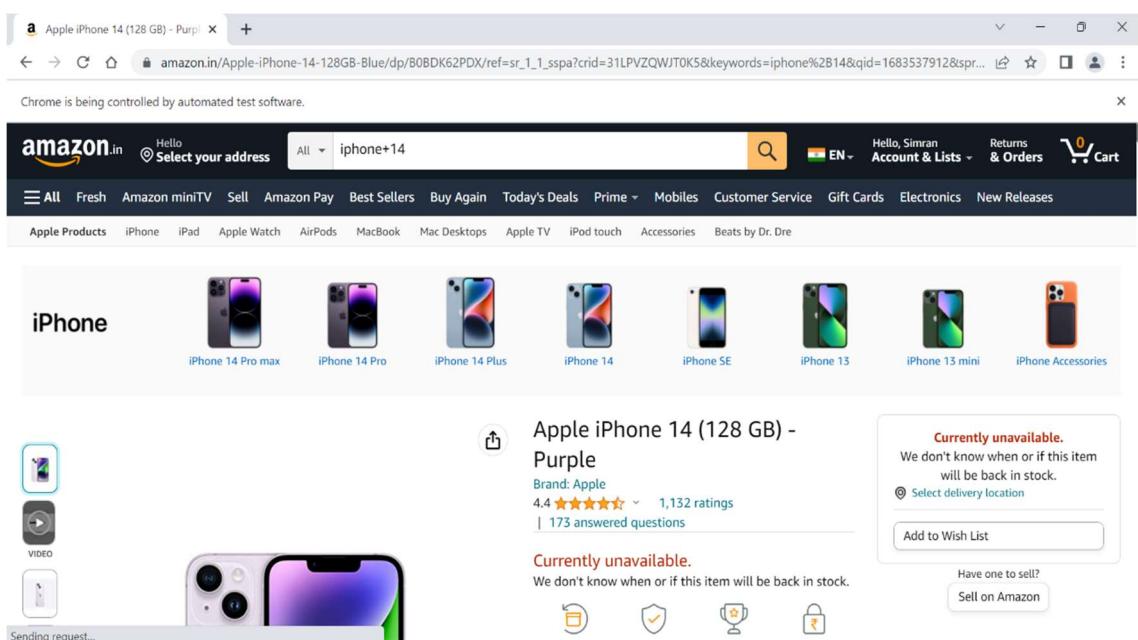
1. Automated Searching.



2. Automated Web-Results.



3. Product Features.



SonarQube(3)

Run a basic SonarQube scan and show dashboard with 3 types of issues ie Bug, Vulnerability and Code Smell

For this project, we use Sonarcloud and we connect it with our GitHub organization and scan a existing repository. The following steps are followed: -

- Create a SonarCloud Account: We sign up for a SonarCloud account at <https://sonarcloud.io> using our GitHub credentials.
- Connect SonarCloud with GitHub Organization: In SonarCloud, we go to our account settings and select "Organizations" from the sidebar. We click on "Create Organization" and choose the option to connect with our GitHub organization. We follow the prompts to authorize SonarCloud to access our GitHub organization.

The screenshot shows the SonarCloud interface. At the top, there is a navigation bar with links for 'My Projects', 'My Issues', 'Explore', and a search icon. Below the navigation bar, the title 'Create an organization' is displayed in bold. A descriptive text states, 'An organization is a space where a team or a whole company can collaborate across many projects.' Below this text is a button labeled 'Import an organization from GitHub'. At the bottom of the page, there is a note: 'Just testing? You can [create an organization manually](#)'.

- Create a New Project in SonarCloud: After connecting with our GitHub organization, we create a new project in SonarCloud.
- Select the organization we connected with and we choose the repository that we want to analyse. Lastly, we configure any additional project settings as needed.

SAP One Analyze projects +

sonarcloud.io/projects/create

Gmail YouTube Maps

sonarcloud My Projects My Issues Explore Q

Analyze projects

Select repositories from one of your GitHub organization.

Organization Simran

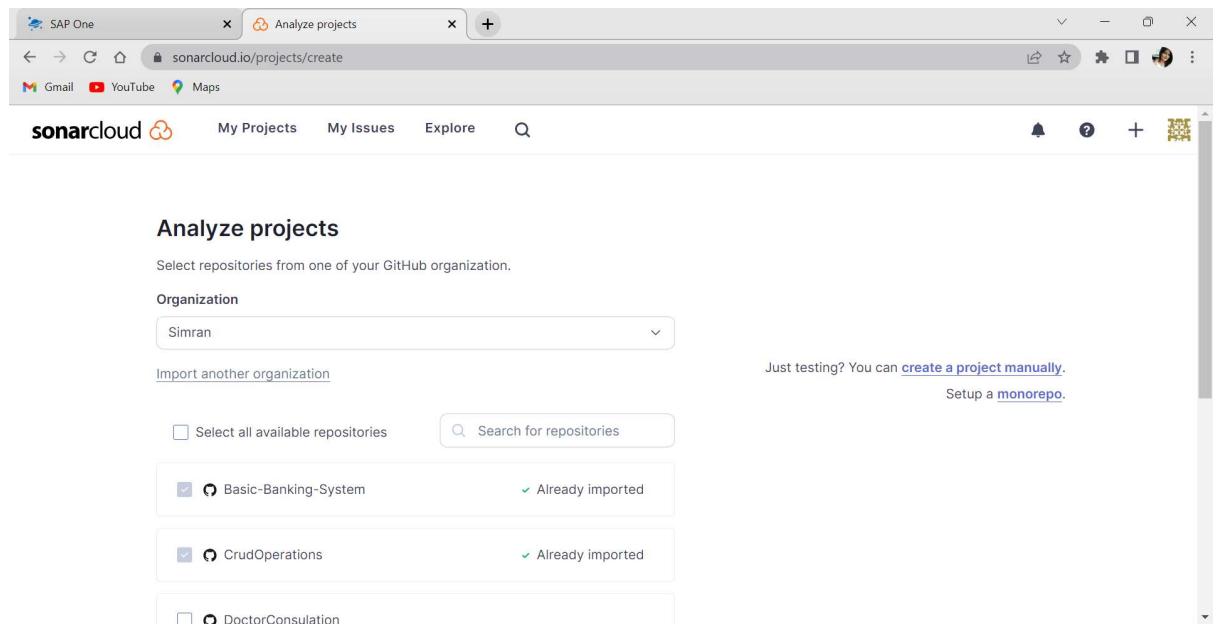
Import another organization Just testing? You can [create a project manually](#).
Setup a [monorepo](#).

Select all available repositories Search for repositories

Basic-Banking-System ✓ Already imported

CrudOperations ✓ Already imported

DoctorConsultation



SAP One Resume-using-HTML5 - Simran +

sonarcloud.io/project/configuration/AutoScan?id=simmi0206_Resume-using-HTML5

Gmail YouTube Maps

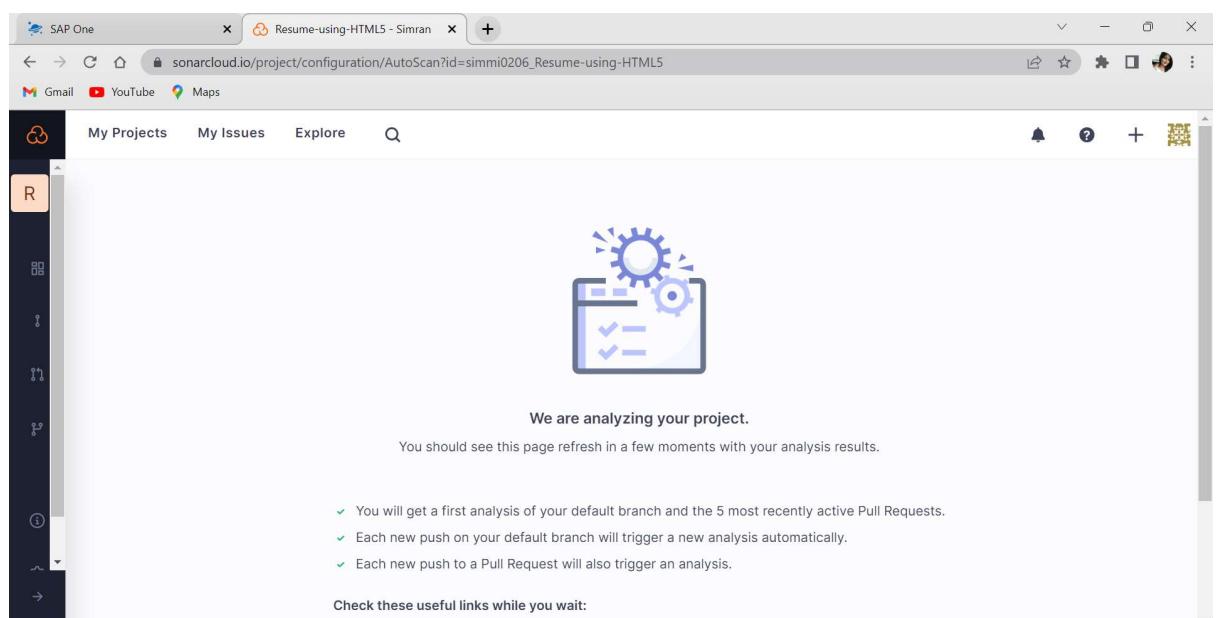
My Projects My Issues Explore Q

R

We are analyzing your project.
You should see this page refresh in a few moments with your analysis results.

- ✓ You will get a first analysis of your default branch and the 5 most recently active Pull Requests.
- ✓ Each new push on your default branch will trigger a new analysis automatically.
- ✓ Each new push to a Pull Request will also trigger an analysis.

Check these useful links while you wait:



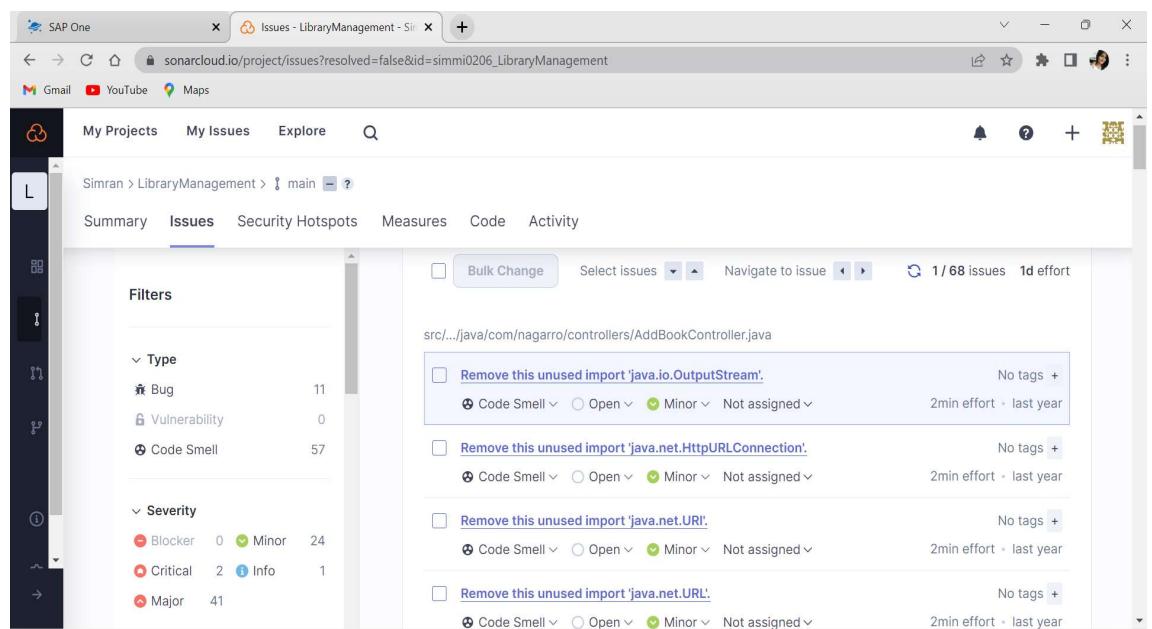
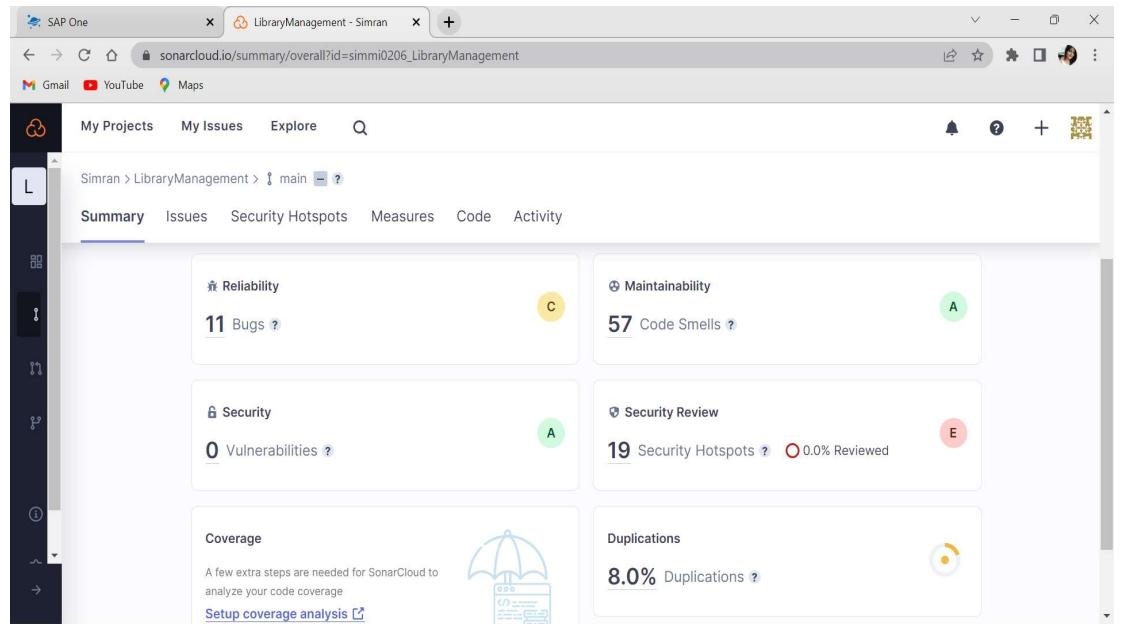
The screenshot shows the SonarCloud interface with the following details:

- Project Overview:** My Projects tab selected.
- Filters:** Quality Gate (Passed: 0, Failed: 0), Reliability (A: 0, B: 1, C: 3, D: 0, E: 0), Security (A: 2).
- Search:** Search for projects, Perspective (Overall Status), Sort by (Name), 4 projects found.
- Project 1: Simran / Resume-using-HTML5**
 - Last analysis: 6/5/2023, 4:54 PM · 101 Lines of Code · HTML
 - Bugs: 1, Vulnerabilities: 0, Hotspots Reviewed: 0.0%, Code Smells: 0, Duplications: 0.0%
- Project 2: Simran / LibraryManagement**
 - Last analysis: 5/26/2023, 4:54 PM · 967 Lines of Code · Java, JSP, ...
 - Bugs: 11, Vulnerabilities: 0, Hotspots Reviewed: 0.0%, Code Smells: 57, Duplications: 8.0%

The screenshot shows the SonarCloud interface with the following details:

- Project Overview:** My Projects tab selected.
- Filters:** Quality Gate (Passed: 0, Failed: 0), Reliability (A: 0, B: 1, C: 3, D: 0, E: 0), Security (A: 2).
- Project 1: Simran / CrudOperations**
 - Last analysis: 5/26/2023, 4:54 PM · 261 Lines of Code · Java, XML, ...
 - Bugs: 1, Vulnerabilities: 2, Hotspots Reviewed: 100%, Code Smells: 6, Duplications: 0.0%
- Project 2: Simran / Basic-Banking-System**
 - Last analysis: 5/26/2023, 4:54 PM · 508 Lines of Code · CSS, PHP
 - Bugs: 34, Vulnerabilities: 10, Hotspots Reviewed: 0.0%, Code Smells: 129, Duplications: 0.0%

We can view the detailed analysis of our projects by clicking on them.



- It shows the part/line where vulnerabilities/bugs are present.
- This helps us to maintain high code quality, enhance security, and improve maintainability, ultimately leading to a more reliable and efficient software development process.

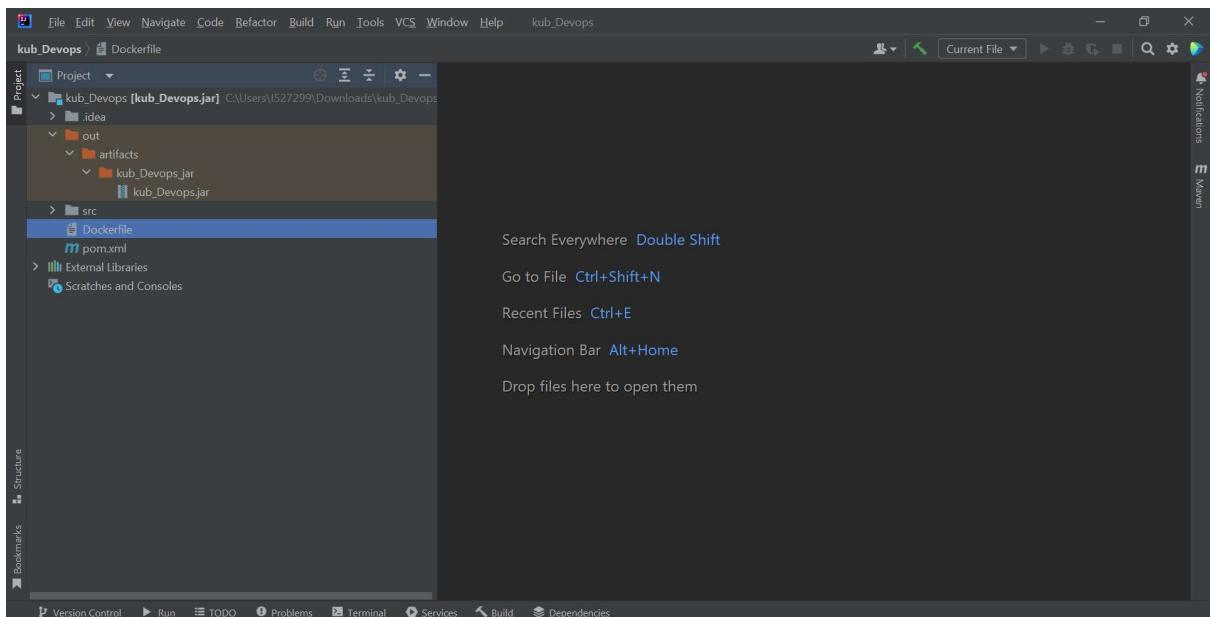
Kubernetes (3)

- 1. Take a Kubernetes image and deploy with high availability, fault tolerance, scalability provided by Kubernetes.**

Pre-requisites

1. Docker should be installed on the system and should be in running state
2. Docker Hub free account is created
3. Git Bash is installed to run the docker commands To create a Kubernetes image, you'll first need to create a Docker image since Kubernetes works with container images.

Step 1: Create Dockerfile in application root directory.



Step 2: Create docker image (make sure your docker desktop is running and in terminal you are in application directory)

In cmd : docker build -t dockerimage .

Step 3: Login to docker hub (make sure you have docker hub account)

In cmd : docker login

Step 4: Give tag to your image as username/repository name

In cmd : Docker tag dockerimage simran111111/repodocker

Step 5: Push docker image to docker hub

In cmd: Docker push simran111111/repodocke

Step 6: Install kubectl with link: <https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>

Step 7: Add path to environment variable.

Step 8: Create a Kubernetes deployment manifest file, that describe application's state. configure the file with image details and options to enable high availability, fault tolerance, scalability

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: dockerimage
          ports:
            - containerPort: 8500
          resources:
            limits:
              cpu: 0.5
              memory: "512Mi"
            requests:
              cpu: 0.2
              memory: "256Mi"
```

Step 9: Create Kubernetes cluster: Enable Kubernetes in docker desktop

Step 10: Deploy the image to created cluster

In cmd: kubectl apply -f demo.yaml

Step 11: Get deployments:

cmd: kubectl get deployments

Step 12. Get pods:

cmd: kubectl get pods

Step 13. Get info :

cmd: kubectl cluster-inf

```
Command Prompt      + - X

C:\Users\I527261>cd Downloads

C:\Users\I527261\Downloads>kubectl apply -f demo.yaml
deployment.apps/my-app-deployment created

C:\Users\I527261\Downloads>kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
my-app-deployment   0/3       3           0          22s

C:\Users\I527261\Downloads>kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
my-app-deployment-f5758bd8-56r7w   0/1     ImagePullBackOff   0    92s
my-app-deployment-f5758bd8-l7swg   0/1     ImagePullBackOff   0    92s
my-app-deployment-f5758bd8-vnwxs   0/1     ErrImagePull      0    92s

C:\Users\I527261\Downloads>kubectl cluster -info
error: unknown command "cluster" for "kubectl"

Did you mean this?
      cluster-info

C:\Users\I527261\Downloads>
C:\Users\I527261\Downloads>kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```