**Advanced Mockito  
  
 1.Mocking databases and Repositories**

**Repository.java**public interface Repository {

String getData();

}  
 **Service.java**public class Service {

private final Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

String data = repository.getData();

return "Processed " + data;

}

}

**ServiceTest.java**import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class ServiceTest {

@Test

public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData()).thenReturn("Mock Data");

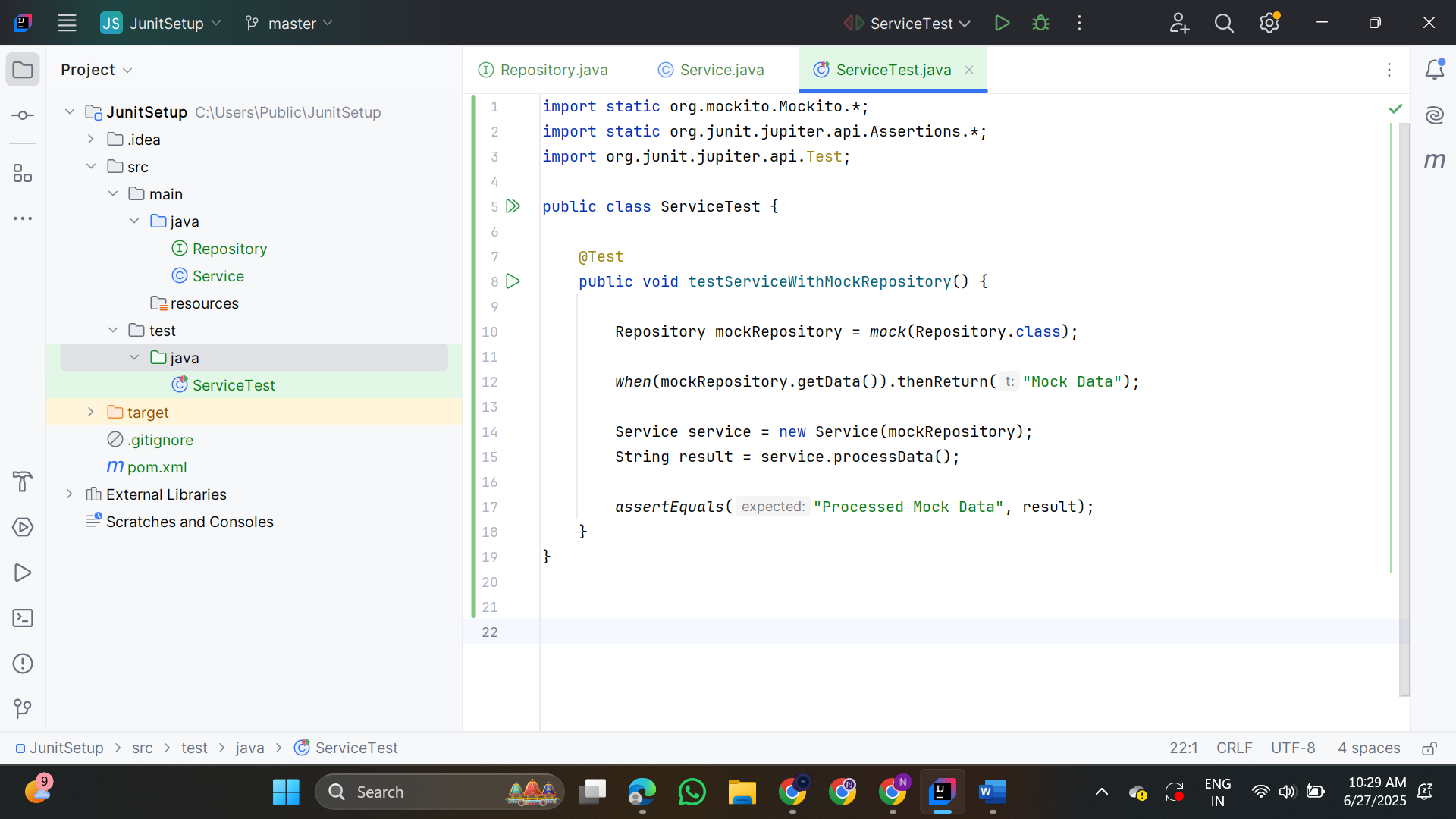
Service service = new Service(mockRepository);

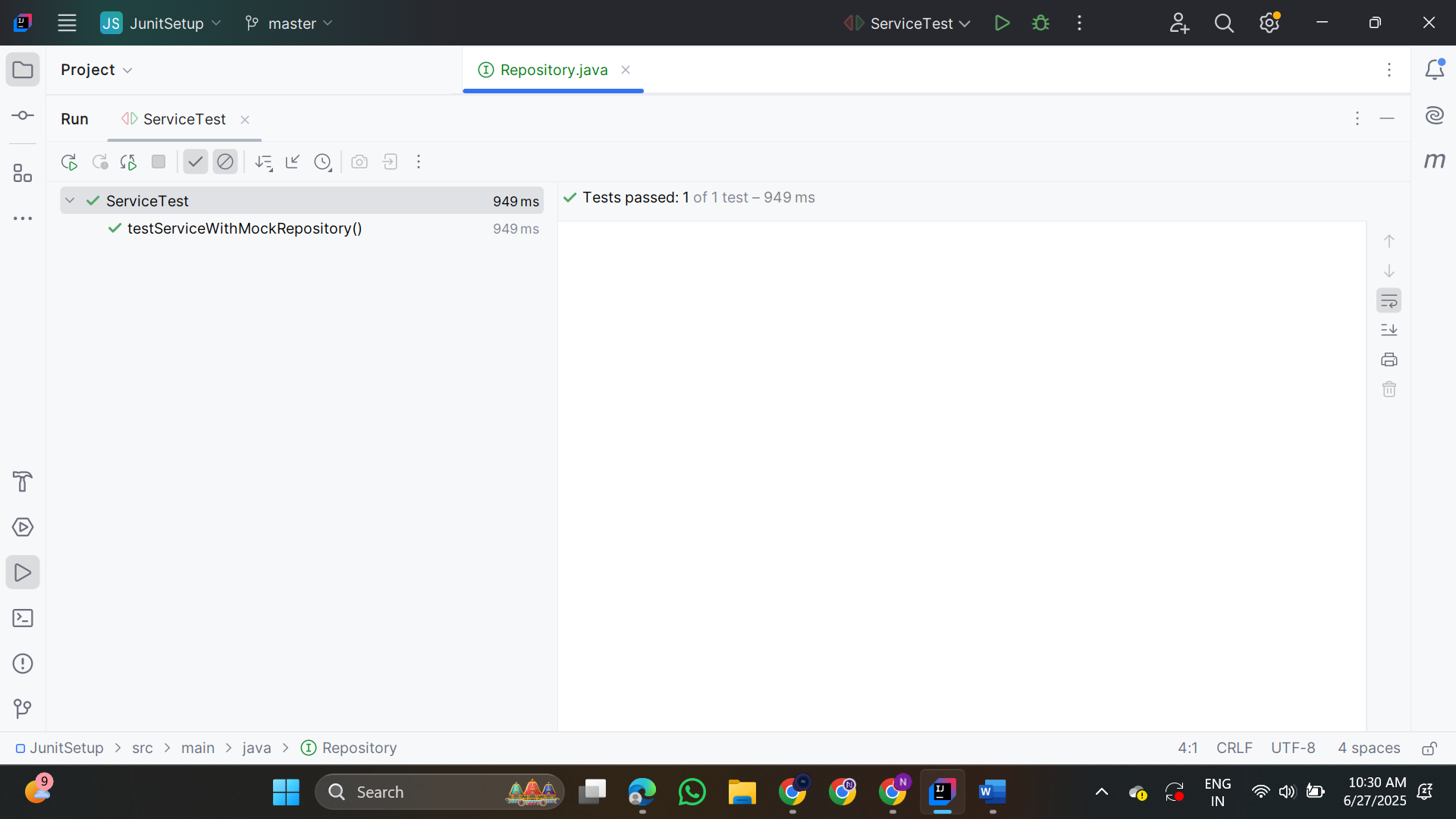
String result = service.processData();

assertEquals("Processed Mock Data", result);

}

}





**2.Mocking External Services (Rest API)**

**RestClient.java**

public class RestClient {

public String getResponse() {

return "Real Response";

}

}

**ApiService.java**

public class ApiService {

private final RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

return "Fetched " + restClient.getResponse();

}

}

**ApiServiceTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class ApiServiceTest {

@Test

public void testServiceWithMockRestClient() {

RestClient mockRestClient = mock(RestClient.class);

when(mockRestClient.getResponse()).thenReturn("Mock Response");

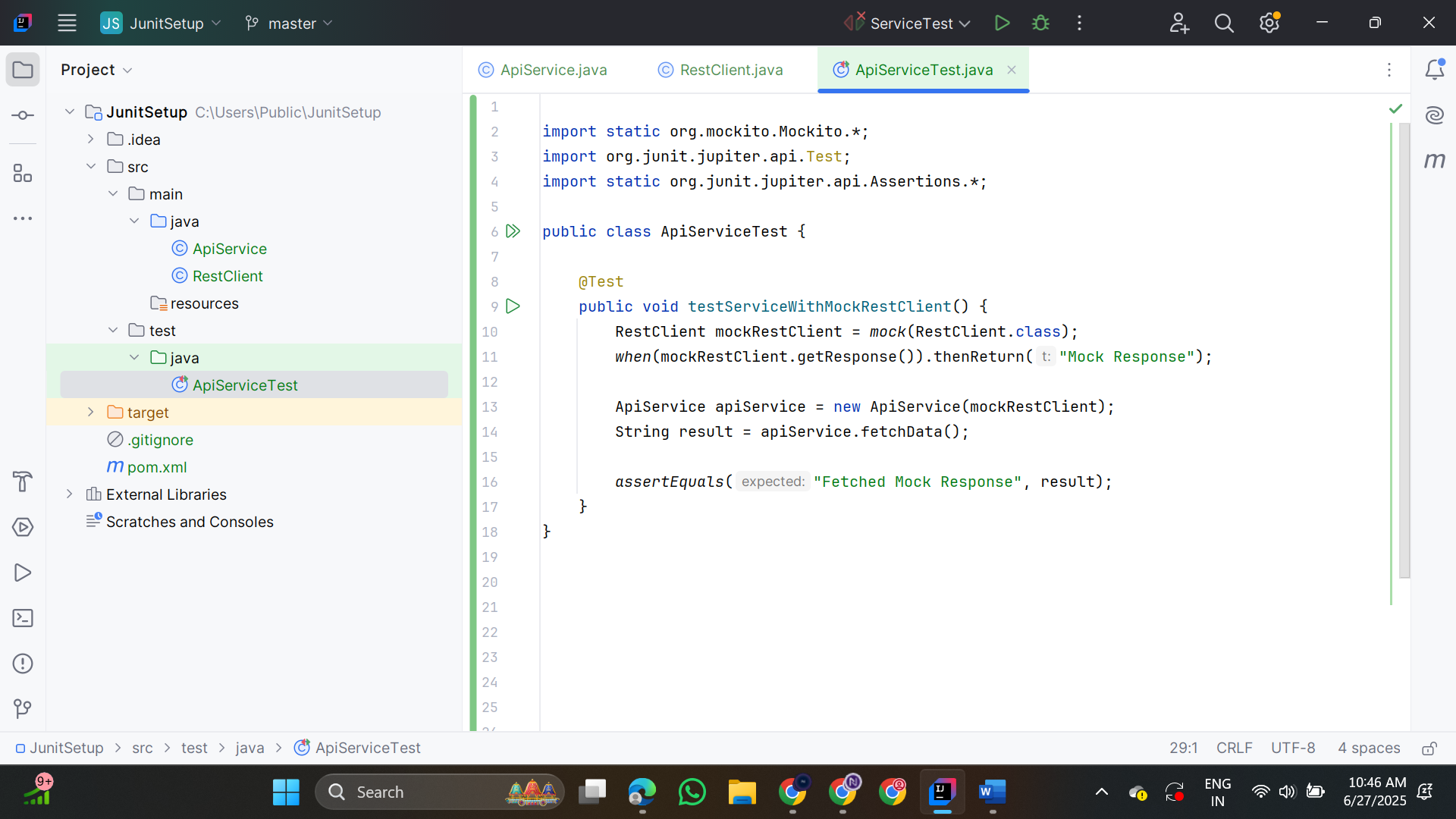
ApiService apiService = new ApiService(mockRestClient);

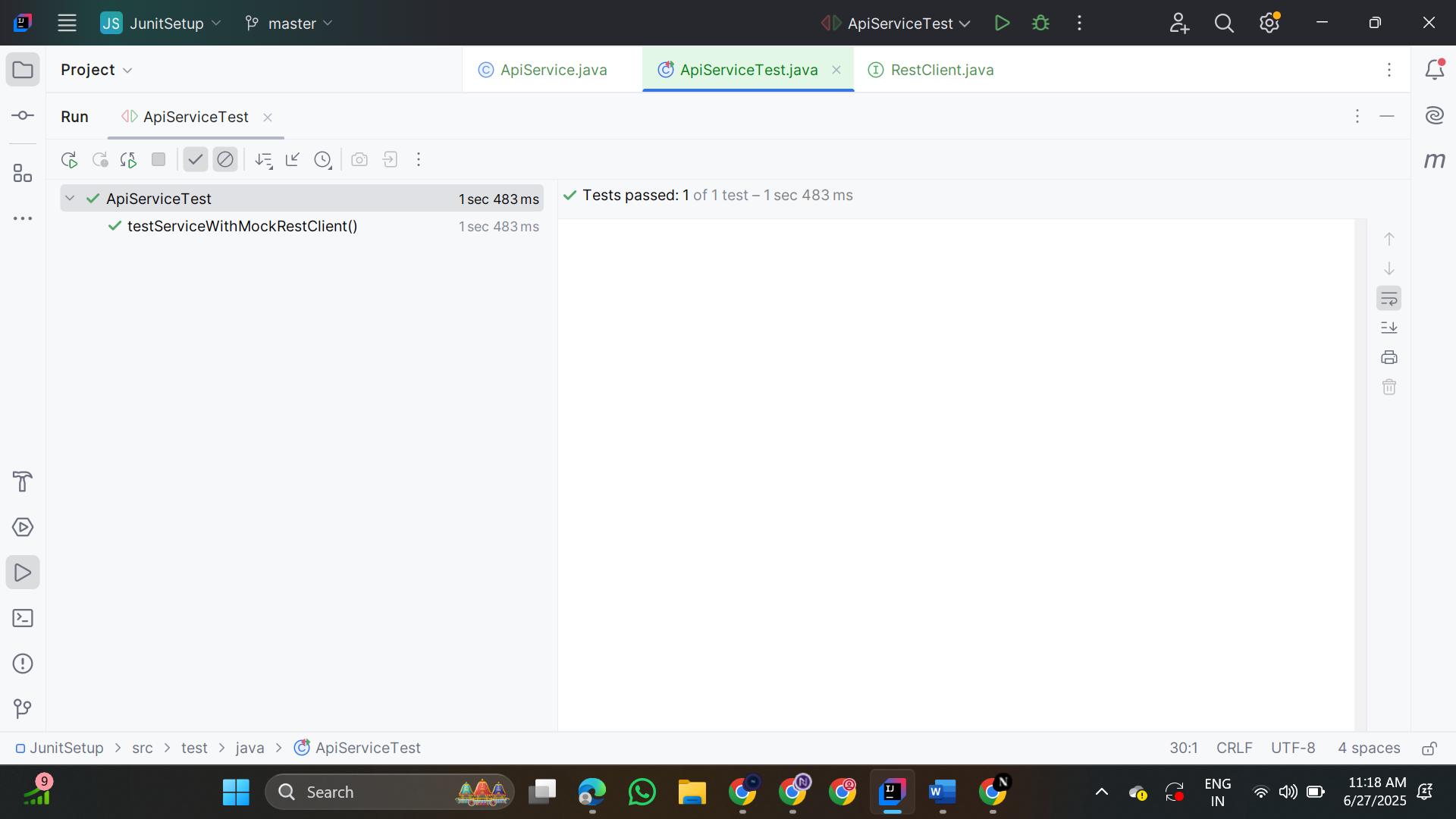
String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result);

}

}





**3.Mocking File I/O**

**FileReaderInterface.java**

public interface FileReaderInterface {

String read();

}

**FileWriterInterface.java**

public interface FileWriterInterface {

void write(String content);

}

**MyFileReader.java**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class MyFileReader implements FileReaderInterface {

private final String filePath;

public MyFileReader(String filePath) {

this.filePath = filePath;

}

@Override

public String read() {

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

return reader.readLine();

} catch (IOException e) {

throw new RuntimeException("Error reading file", e);

}

}

}

**MyFileWriter.java**

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

public class MyFileWriter implements FileWriterInterface {

private final String filePath;

public MyFileWriter(String filePath) {

this.filePath = filePath;

}

@Override

public void write(String content) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

writer.write(content);

} catch (IOException e) {

throw new RuntimeException("Error writing to file", e);

}

}

}

**FileService.java**public class FileService {

private final FileReaderInterface reader;

private final FileWriterInterface writer;

public FileService(FileReaderInterface reader, FileWriterInterface writer) {

this.reader = reader;

this.writer = writer;

}

public String processFile() {

String content = reader.read();

writer.write(content);

return "Processed " + content;

}

}

**FileServiceTest.java**

public class FileServiceTest {

@Test

public void testServiceWithMockFileIO() {

FileReaderInterface mockReader = mock(FileReaderInterface.class);

FileWriterInterface mockWriter = mock(FileWriterInterface.class);

when(mockReader.read()).thenReturn("Mock File Content");

FileService service = new FileService(mockReader, mockWriter);

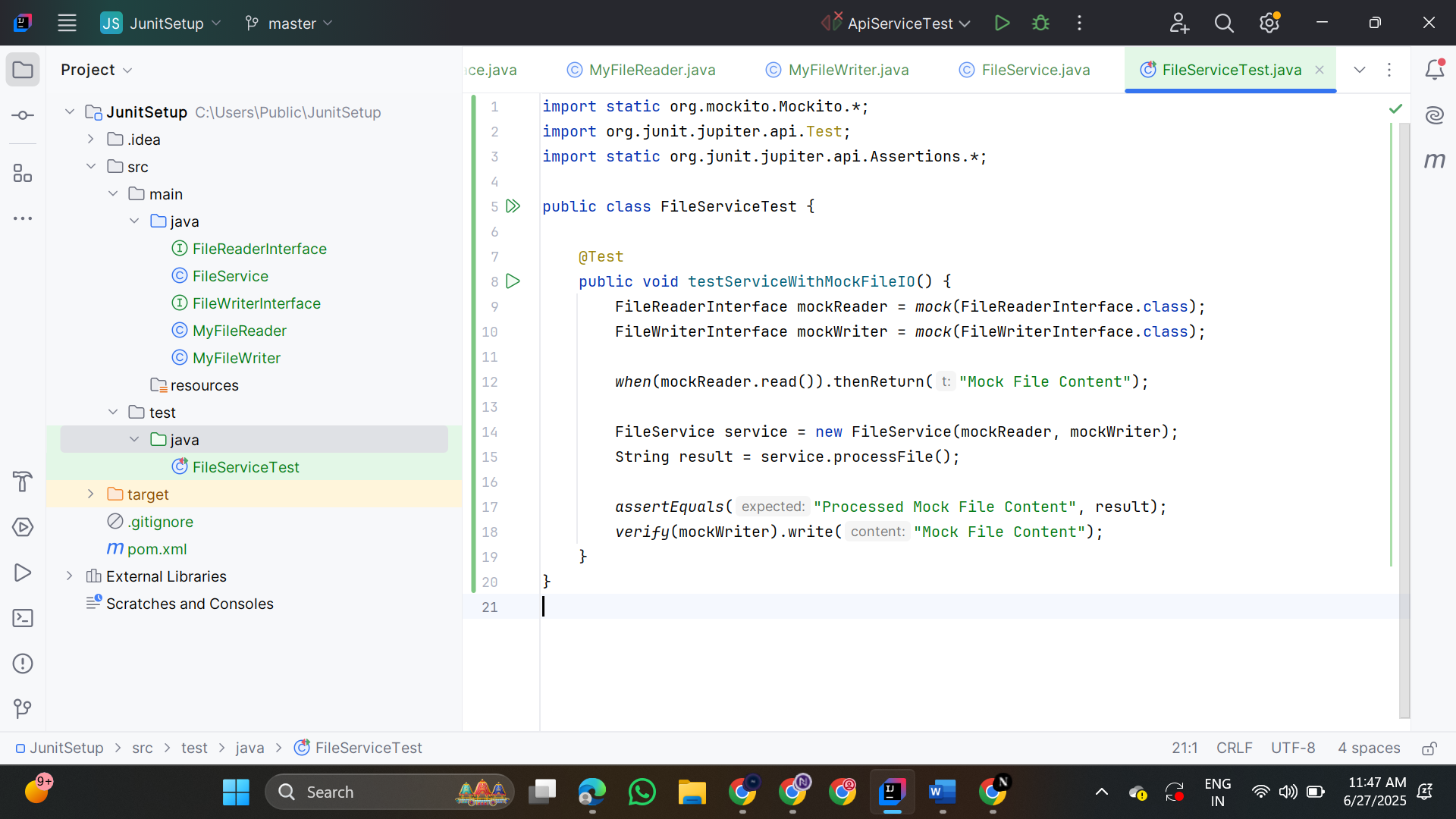
String result = service.processFile();

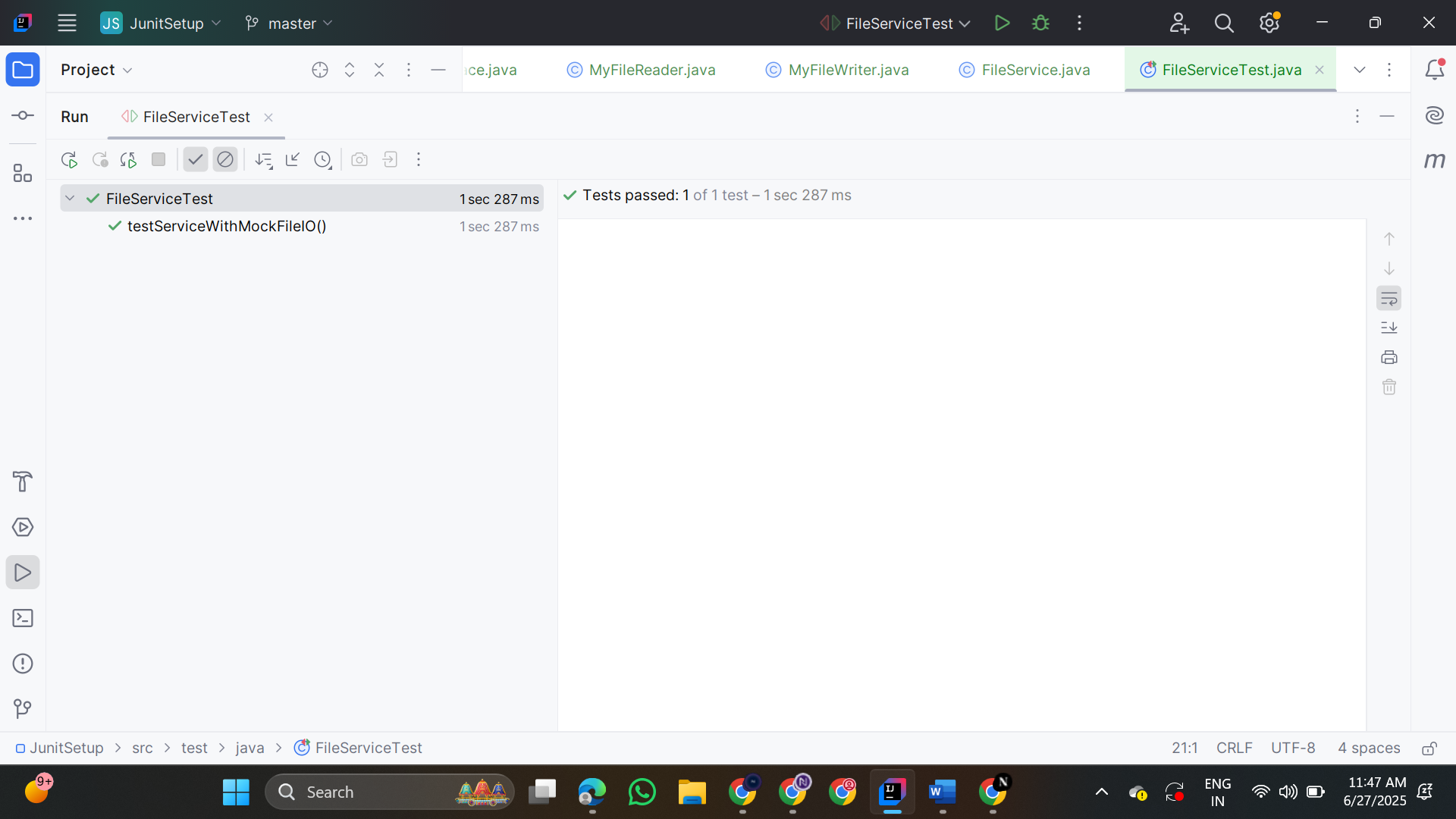
assertEquals("Processed Mock File Content", result);

verify(mockWriter).write("Mock File Content");

}

}





**4.Mocking Network Interactions**

**NetworkClient.java**

public interface NetworkClient {

String connect();

}

**NetworkService.java**

public class NetworkService {

private final NetworkClient networkClient;

public NetworkService(NetworkClient networkClient) {

this.networkClient = networkClient;

}

public String connectToServer() {

return "Connected to " + networkClient.connect();

}

}

**NetworkServiceTest.java**

public class NetworkServiceTest {

@Test

public void testServiceWithMockNetworkClient() {

NetworkClient mockNetworkClient = mock(NetworkClient.class);

when(mockNetworkClient.connect()).thenReturn("Mock Connection");

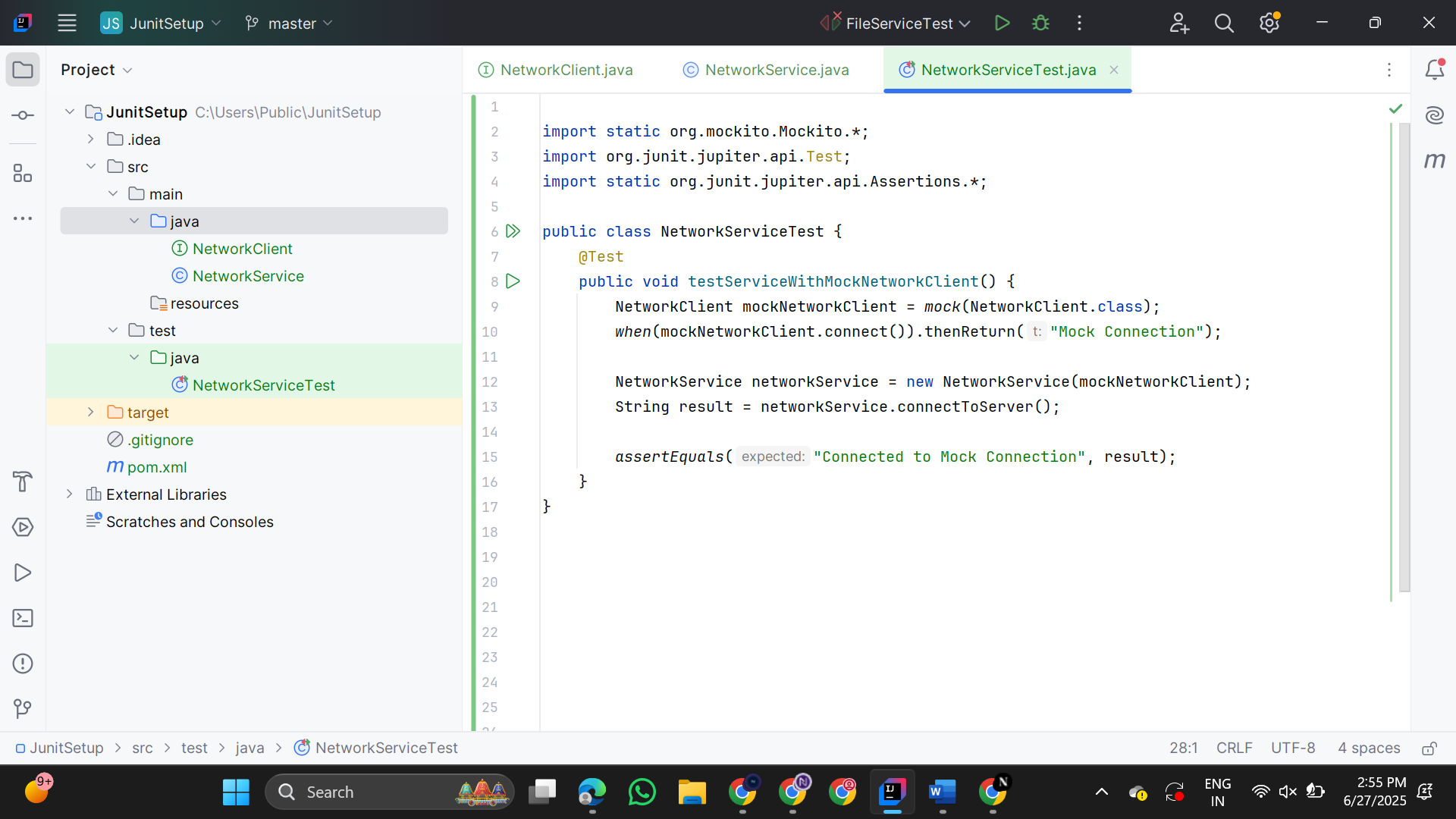
NetworkService networkService = new NetworkService(mockNetworkClient);

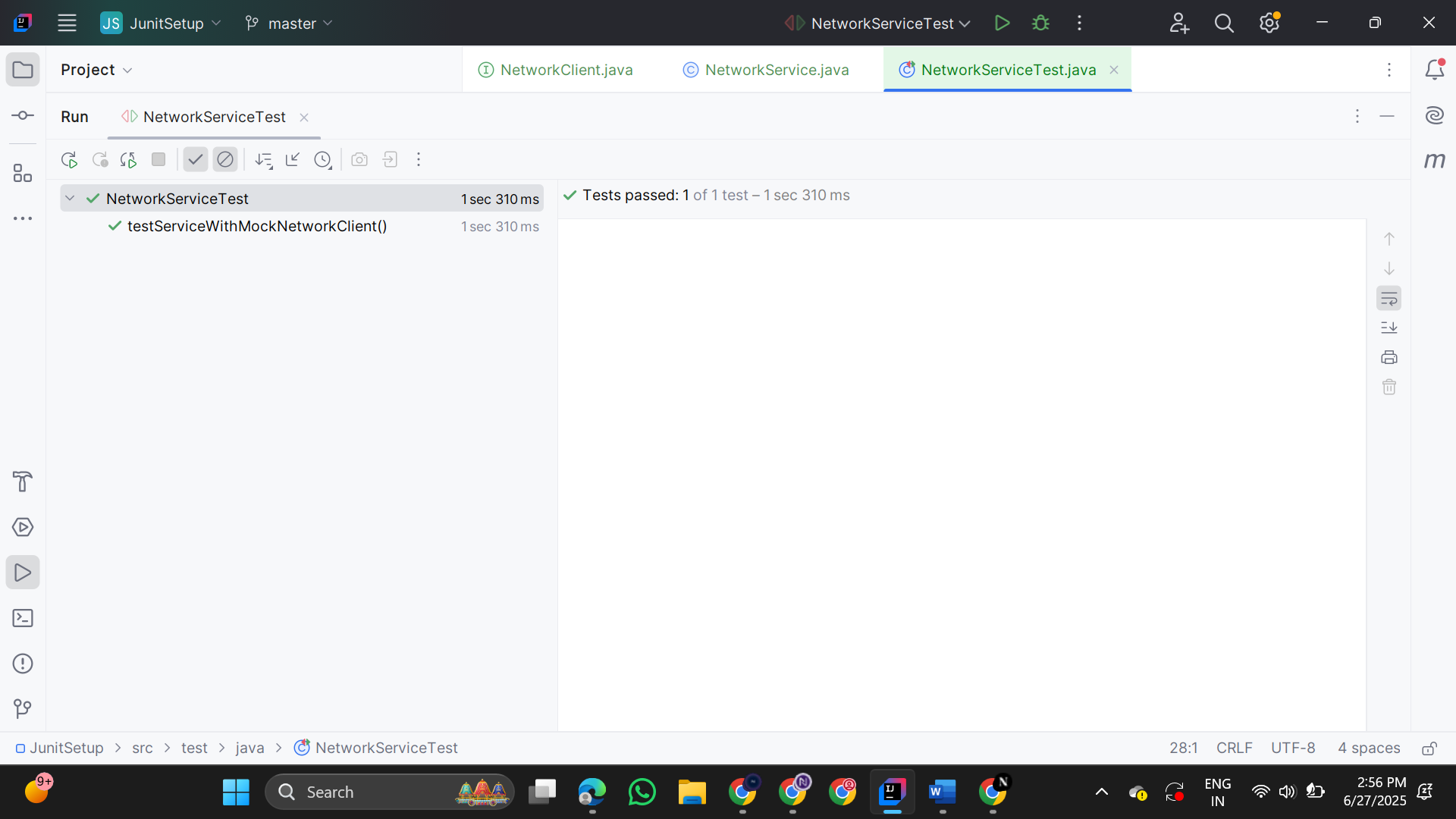
String result = networkService.connectToServer();

assertEquals("Connected to Mock Connection", result);

}

}





**5.Mocking Multiple Return Values**

**Repository.java**

public interface Repository {

String getData();

}

**Service.java**

public class Service {

private final Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**MultiReturnServiceTest.java**

public class MultiReturnServiceTest {

@Test

public void testServiceWithMultipleReturnValues() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

Service service = new Service(mockRepository);

String firstResult = service.processData();

String secondResult = service.processData();

assertEquals("Processed First Mock Data", firstResult);

assertEquals("Processed Second Mock Data", secondResult);

}

}

