# Server GUI Development: APIs & Libraries Documentation

This document provides an overview of the APIs and libraries used in the Server GUI development project. It covers Java Swing for GUI implementation, JUnit for testing, and Mockito for mocking server responses during integration testing.

## 1. Java Swing

Java Swing is a GUI toolkit for creating desktop applications in Java. It provides a set of lightweight components for building user interfaces, such as windows, panels, buttons, and text fields. In this project, Java Swing is used to implement various panels for the Server GUI.

### Key Swing Components

The following Swing components are used in this project:

* JFrame: The main window container for the GUI.
* JPanel: Used to organize and group components in the interface.
* JTextArea: Displays logs and allows text input/output.
* JTextField: Provides input fields for server configurations.
* JLabel: Displays text information, such as server status or client list.
* JScrollPane: Adds scroll functionality to JTextArea for viewing long logs.
* BorderLayout, GridLayout, BoxLayout: Used to manage the layout of components in the panels.

### Code Integration Example

Here is an example of how Java Swing components are used to create a simple GUI panel:

import javax.swing.\*;  
  
public class SimplePanelExample {  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Simple Swing Example");  
 JPanel panel = new JPanel();  
   
 panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));  
 panel.add(new JLabel("Server Status: Running"));  
 panel.add(new JTextArea(5, 20));  
   
 frame.add(panel);  
 frame.setSize(300, 200);  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.setVisible(true);  
 }  
}

## 2. JUnit

JUnit is a unit testing framework for Java, used to write and run repeatable tests. It is widely used for test-driven development and provides annotations for structuring tests and assertions for validating results.

### Setting Up JUnit

To use JUnit in the project, add the following dependency to your Maven POM file:

<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
</dependency>

### Writing Test Cases with JUnit

JUnit provides annotations such as @Test, @Before, and @After to structure tests. The following example shows a basic test case for verifying server status updates:

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class ServerStatusTest {  
 @Test  
 public void testServerStatus() {  
 String status = "Server is running...";  
 assertEquals("Server is running...", status);  
 }  
}

## 3. Mockito

Mockito is a popular mocking framework for Java, used to simulate dependencies and create mock objects during testing. It is integrated with JUnit to perform integration testing by mocking server responses in this project.

### Setting Up Mockito

To use Mockito, add the following dependency to your Maven POM file:

<dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>4.11.0</version>  
 <scope>test</scope>  
</dependency>

### Using Mockito for Mocking

Mockito is used to mock server responses and simulate interactions. The following example demonstrates how to mock a server status call:

import org.mockito.Mockito;  
import static org.mockito.Mockito.\*;  
  
public class MockServerTest {  
 public static void main(String[] args) {  
 ServerMock server = mock(ServerMock.class);  
 when(server.getServerStatus()).thenReturn("Server is running...");  
  
 // Simulated test  
 String status = server.getServerStatus();  
 System.out.println(status); // Output: Server is running...  
 }  
}

## Summary

This document covered the APIs and libraries used in the Server GUI development project. Java Swing is utilized for GUI implementation, JUnit for writing unit and integration tests, and Mockito for mocking dependencies.