Reservationssystem für einen Reinigungsdienst

INF202-Projekt

Musab Nail Çekerek

180501017

TMS-4

e180501017@stud.tau.edu.tr

Mert Bayram

170501025

TMS-4

e170501025@stud.tau.edu.tr

MS4

Architekturüberblick

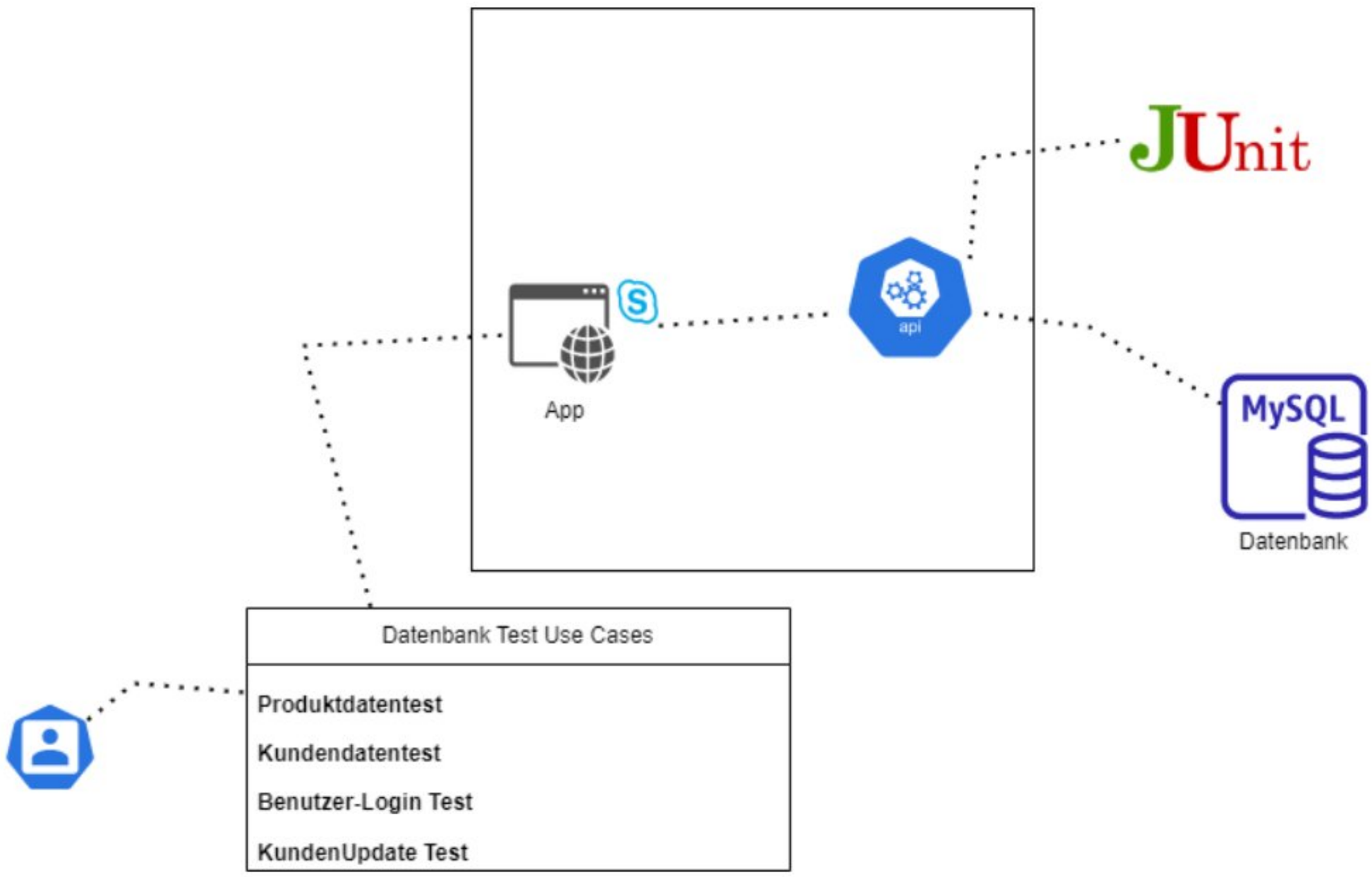

Abbildung 1: Architekturüberblick

Übersicht des Testsystems:

- Das Testsystem besteht aus einer Entwicklungs- und einer Testumgebung, in denen die Testdurchführung stattfindet.
- Die Entwicklungs-umgebung wird für die Entwicklung des Reframe und der Datenbankverbindung verwendet, während die Testumgebung für die Durchführung der Test-szenarien vorgesehen ist.

Komponenten des Testsystems:

Reframe:

- Die zu testende Reframe-Komponente befindet sich in der Entwicklungs-umgebung.
- Diese Komponente ist Teil der GUI-Ebene und wird verwendet, um Ergebnisse anzuzeigen und zu verarbeiten.
- Die Tests des Reframe haben zum Ziel, die korrekte Funktionsweise der Benutzeroberfläche zu überprüfen und sicherzustellen, dass die Ergebnisverarbeitung gemäß den Erwartungen erfolgt.

Datenbankverbindung:

- Die zu testende Komponente der Datenbankverbindung wird sowohl in der Entwicklungs- als auch in der Testumgebung eingesetzt.

- Diese Komponente wird verwendet, um Datenbankoperationen durchzuführen und dem Reframe Daten bereitzustellen.
- Die Tests der Datenbankverbindung zielen darauf ab, sicherzustellen, dass das Lesen, Schreiben, Aktualisieren und Löschen von Daten korrekt funktioniert und der Austausch von Daten zwischen dem Reframe und der Datenbank erfolgreich ist.

Testwerkzeuge:

- Testwerkzeuge werden verwendet, um die Ausführung der Test-szenarien und die Überwachung der Ergebnisse zu unterstützen.
- Zum Beispiel ermöglicht ein Testframework wie JUnit das automatisierte Ausführen der Test-szenarien und das Erstellen von Berichten über die Ergebnisse.
- Darüber hinaus können spezielle Testwerkzeuge für die Datenbankverbindungstests oder Datenbanktestwerkzeuge eingesetzt werden.

Ablauf des Testsystems:

Entwicklungs-umgebung:

- Die Reframe-Komponente wird entwickelt und die Test-szenarien in der Entwicklungs-umgebung erstellt.
- Die Konfiguration der Datenbankverbindung erfolgt ebenfalls in der Entwicklungs-umgebung und wird in die Test-szenarien einbezogen.
- Die Test-szenarien werden in der Entwicklungs-umgebung mit Hilfe von JUnit oder ähnlichen Testframeworks ausgeführt und die Ergebnisse werden überwacht.

Testumgebung:

- Die Test-szenarien werden in der Testumgebung durchgeführt.
- Die Testumgebung ist unabhängig von der Entwicklungs-umgebung und simuliert reale Anwendungsszenarien.
- Die Test-szenarien für Reframe und Datenbankverbindung werden in der Testumgebung ausgeführt und die Ergebnisse werden überwacht.

Systemtestfälle

System Test Case ID Name	**ResFrame Testen**
Preconditions	**5. testAddButtonActionPerformed-Szenario:** Setzen Sie das Feld tc_txt auf „12345“. Setzen Sie das Feld res_txt auf „1“. Setzen Sie das Feld date_txt auf „2023-06-02“. Auswahl von „Matratzen“ im ComboBox-Feld. **2. testEditButtonActionPerformed-Szenario:** Auswahl einer zu bearbeitenden Reservierung in der Tabelle. **3. testDeleteButtonActionPerformed-Szenario:** Auswahl einer zu löschenden Reservierung in der Tabelle. **4. testUpdateButtonActionPerformed-Szenario:** Es sind keine besonderen Voraussetzungen erforderlich. **5. testAbfrageButtonActionPerformed-Szenario:** Es sind keine besonderen Voraussetzungen erforderlich.
Test Steps	**1. testAddButtonActionPerformed-Szenario:** Stellen Sie sicher, dass die Voraussetzungen erfüllt sind. Klicken Sie auf die Schaltfläche „Hinzufügen“. Überprüfen Sie, ob eine neue Reservierung mit den bereitgestellten Einträgen erfolgreich hinzugefügt wurde. **2. testEditButtonActionPerformed-Szenario:** Stellen Sie sicher, dass die Voraussetzungen erfüllt sind. Wählen Sie in der Tabelle eine zu bearbeitende Reservierung aus. Klicken Sie auf die Schaltfläche „Bearbeiten“. Reservierungsinformationen bearbeiten. Stellen Sie sicher, dass die Änderungen erfolgreich übernommen wurden. **3. testDeleteButtonActionPerformed-Szenario:** Stellen Sie sicher, dass die Voraussetzungen erfüllt sind.

	Wählen Sie in der Tabelle eine Reservierung aus, die gelöscht werden soll. Klicken Sie auf den Button „Löschen“. Stellen Sie sicher, dass die Reservierung erfolgreich gelöscht wurde. **4. testUpdateButtonActionPerformed-Szenario:** Stellen Sie sicher, dass die Voraussetzungen erfüllt sind. Klicken Sie auf die Schaltfläche „Aktualisieren“. Stellen Sie sicher, dass die Tabelle aktualisiert ist. **5. testAbfrageButtonActionPerformed-Szenario:** Stellen Sie sicher, dass die Voraussetzungen erfüllt sind. Klicken Sie auf die Schaltfläche „Abfrage des Reservierungen“. Überprüfen Sie, ob Reservierungen richtig gefiltert werden.
Post-Condition	**1. testAddButtonActionPerformed-Szenario-Nachbedingungen:** Die neue Reservierung wurde erfolgreich hinzugefügt. In der Reservierungstabelle wurde eine neue Zeile erstellt. Pflichtfelder der Reservierung (Reservations_Nr, Kunden_TC, Datum, Produkt) sind korrekt ausgefüllt. **2. TestEditButtonActionPerformed-Szenario-Nachbedingungen:** Die Reservierungsinformationen wurden erfolgreich aktualisiert. Bestätigt, dass die aktualisierte Reservierung korrekt in der Tabelle wiedergegeben wird. **3. TestDeleteButtonActionPerformed-Szenario-Nachbedingungen:** Die ausgewählte Reservierung wurde erfolgreich gelöscht. Es wurde überprüft, dass die entsprechende Zeile in der Reservierungstabelle gelöscht wurde.

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

public class ResFrameTest {

    private ResFrame resFrame;

    @Before
    public void setUp() {
        resFrame = new ResFrame();
        resFrame.setVisible(false); // Set the frame to invisible during the test
    }

    @Test
    public void testAddButtonAction() {
        // Simulate clicking on the "Hinzufügen" (Add) button
        resFrame.tc_txt.setText("123456789"); // Fill in the TC field
        resFrame.res_txt.setText("1001"); // Fill in the Reservations Nr. field
        resFrame.date_txt.setText("2023-06-02"); // Fill in the Date field
        resFrame.comboBox.setSelectedItem("Matratzen"); // Select an item from the ComboBox

        resFrame.addButton.doClick(); // Click the "Hinzufügen" (Add) button

        // Check if a row is added to the table after the add operation
        assertEquals(1, resFrame.table.getRowCount());
        assertEquals("1001", resFrame.table.getValueAt(0, 0));
        assertEquals("123456789", resFrame.table.getValueAt(0, 1));
        assertEquals("2023-06-02", resFrame.table.getValueAt(0, 2));
        assertEquals("Matratzen", resFrame.table.getValueAt(0, 3));
    }

    @Test
    public void testDeleteButtonAction() {
        // Add a row to the table beforehand
        resFrame.b.addRow(new Object[]{"1002", "987654321", "2023-06-03", "Teppich"});

        // Set the selected row in the table
        resFrame.table.setRowSelectionInterval(0, 0);

        // Simulate clicking on the "Löschen" (Delete) button
        resFrame.deleteButton.doClick();

        // Check if the row is deleted from the table after the delete operation
        assertEquals(0, resFrame.table.getRowCount());
    }

    @Test
    public void testEditButtonAction() {
        // Add a row to the table beforehand
        resFrame.b.addRow(new Object[]{"1003", "555555555", "2023-06-04", "Sofa"});

        // Set the selected row in the table
        resFrame.table.setRowSelectionInterval(0, 0);

        // Simulate clicking on the "Bearbeiten" (Edit) button
        resFrame.editButton.doClick();

        // Check if the changes made in the table after the edit operation
        assertEquals("555555555", resFrame.table.getValueAt(0, 1));
        assertEquals("2023-06-04", resFrame.table.getValueAt(0, 2));
        assertEquals("Sofa", resFrame.table.getValueAt(0, 3));
    }

    @Test
    public void testUpdateButtonAction() {
        // Add a few rows to the table beforehand
        resFrame.b.addRow(new Object[]{"1004", "111111111", "2023-06-05", "Gardinen"});
        resFrame.b.addRow(new Object[]{"1005", "222222222", "2023-06-06", "Hausreinigung"});

        // Simulate clicking on the "Aktualisieren" (Update) button
        resFrame.addButton.doClick();

```

Abbildung 2: Test Code

Abbildung 3: Test Code

```
        // Simulate clicking on the "Aktualisieren" (Update) button
        resFrame.addButton.doClick();

        // Check if the table is updated after the update operation
        assertEquals(2, resFrame.table.getRowCount());
        assertEquals("1004", resFrame.table.getValueAt(0, 0));
        assertEquals("111111111", resFrame.table.getValueAt(0, 1));
        assertEquals("2023-06-05", resFrame.table.getValueAt(0, 2));
        assertEquals("Gardinen", resFrame.table.getValueAt(0, 3));
        assertEquals("1005", resFrame.table.getValueAt(1, 0));
        assertEquals("222222222", resFrame.table.getValueAt(1, 1));
        assertEquals("2023-06-06", resFrame.table.getValueAt(1, 2));
        assertEquals("Hausreinigung", resFrame.table.getValueAt(1, 3));
    }
```

Abbildung 4: Test Code

KonnektorTest : Database

```
1  import org.junit.jupiter.api.BeforeEach;
2  import org.junit.jupiter.api.Test;
3
4  import java.sql.Connection;
5  import java.sql.DriverManager;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8
9  import static org.junit.jupiter.api.Assertions.*;
10
11 public class KonnektorTest {
12
13     private Konnektor konnektor;
14
15     @BeforeEach
16     public void setUp() {
17         konnektor = new Konnektor();
18         konnektor.conn();
19     }
20
21     @Test
22     public void testConnection() {
23         Connection connection = Konnektor.myConn;
24         assertNotNull(connection);
25         assertTrue(connection instanceof Connection);
26     }
27
28     @Test
29     public void testStatProdukt() throws SQLException {
30         assertNotNull(Konnektor.stat_produkt());
31     }
32
33     @Test
34     public void testStatKunden() throws SQLException {
35         assertNotNull(Konnektor.stat_kunden());
36     }
```

Abbildung 5: Test Code

```java
@Test
public void testStatLogin() throws SQLException {
    assertNotNull(Konnektor.stat_login());
}

@Test
public void testStatLoginUser() throws SQLException {
    // You can add different scenarios to fully test the stat_loginUser() method.
    // For now, let's just check that it's not null:
    assertNotNull(Konnektor.stat_loginUser());
}

// Similar tests can be written for other methods.

@Test
public void testStatRes() throws SQLException {
    assertNotNull(Konnektor.stat_res());
```

Abbildung 6: Test Code

```java
@Test
public void testStatKundenDelete() throws SQLException {
    // Let's add some data for the test scenario
    String user_TC = "123456789";
    String vorname = "John";
    String nachname = "Doe";
    String email = "john.doe@example.com";
    String tel = "1234567890";

    Statement stmt = Konnektor.myConn.createStatement();
    String insert = "INSERT INTO kunden (`Kunden_TC`, `Vorname`, `Nachname`, `email`, `telefonnummer`) VALUES "
            + "('" + user_TC + "', '" + vorname + "','" + nachname + "','" + email + "','" + tel + "')";

    int x = stmt.executeUpdate(insert);
    assertEquals(1, x);

    assertNotNull(Konnektor.stat_kunden_delete());
}

// Similar tests can be written for other methods.

@Test
public void testStatKundenEdit() throws SQLException {
    assertNotNull(Konnektor.stat_kunden_edit());
}
```

Abbildung 7: Test Code

Abbildung 8: Ergebnisse

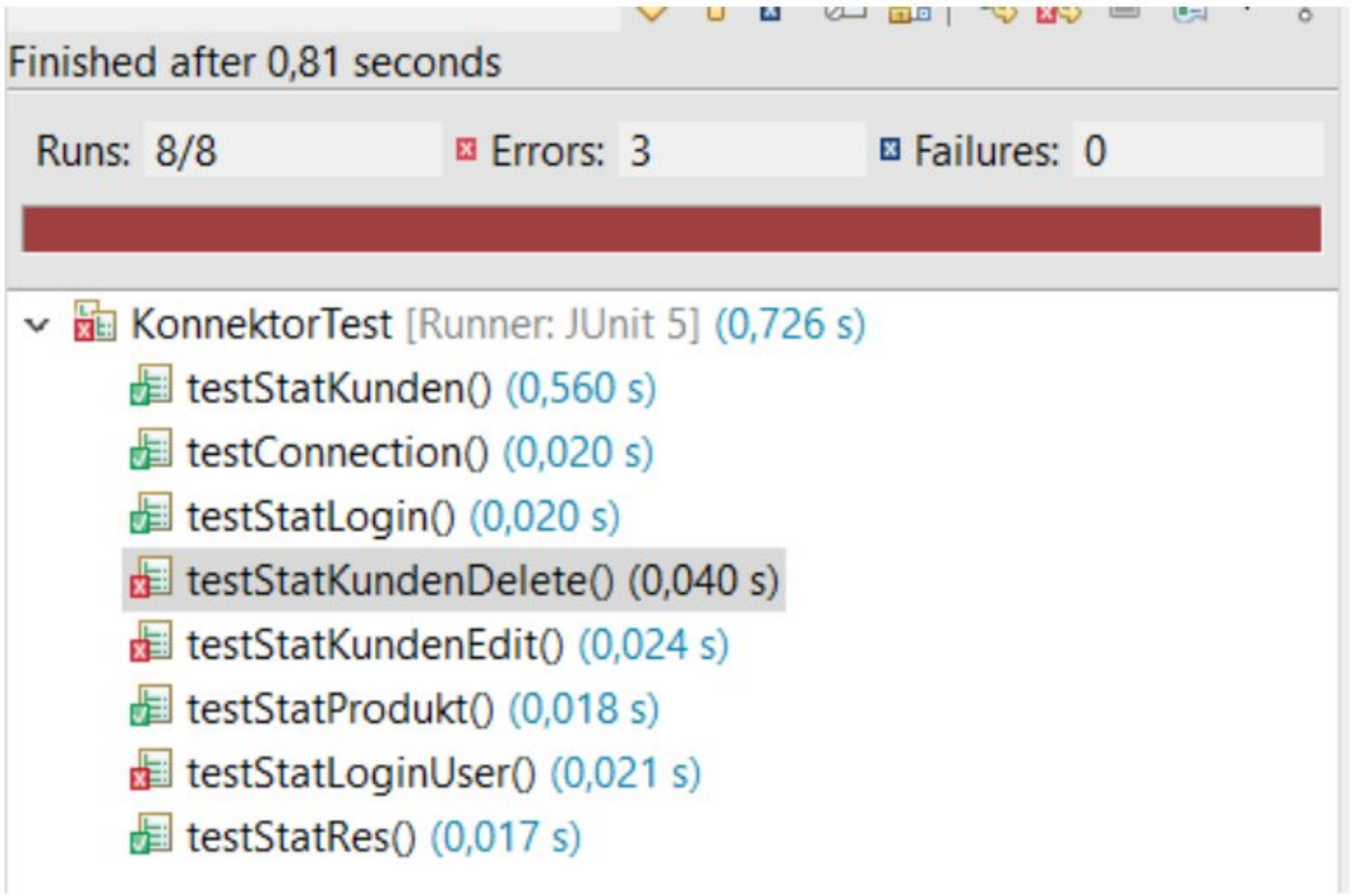

Abbildung 9: Test Classen

Test Case ID Name	KonnektorTest
Preconditions	1. Der MySQL-Datenbankserver muss betriebsbereit sein. 2. Für die Datenbankverbindung müssen die richtige URL, der richtige Benutzername und das richtige Passwort angegeben werden. Die Verbindungs-URL wird als „jdbc:mysql://localhost:3306/reinigung“ angegeben. Anstelle von „localhost“ sollte der entsprechende Servername oder die IP-Adresse verwendet werden. „3306“ ist der MySQL-Standardport. „reinigung“ ist der Datenbankname und sollte durch einen vorhandenen Datenbanknamen ersetzt werden. 3. Der entsprechende JDBC-Treiber (MySQL JDBC-Treiber) muss im Projekt enthalten sein. 4. Die Datenbank sollte vier Tabellen mit den Namen „produkt“, „kunden“, „user“ und „reservation“ enthalten. Wir gehen davon aus, dass diese Tabellen vorhanden sind und über die richtigen Felder verfügen. 5. Wir gehen davon aus, dass GUI-Komponenten wie „LoginFrame“, „ProduktFrame“, „KundenFrame“, „AdminFrame“, „KDienstMenuFrame“ und „ReinigungFrame“ ordnungsgemäß definiert sind. Die Designs dieser Komponenten müssen korrekt erstellt und mit dem Code verknüpft werden.

	6. Wir gehen davon aus, dass Komponenten wie „LoginFrame.userField“ und „LoginFrame.passwordField_1“ vorhanden sind und in diesen Feldern Benutzernamen- und Passworteingaben vorgenommen werden. Diese Komponenten müssen im Design identifiziert und mit den relevanten Stellen verknüpft werden. 7. Wir gehen davon aus, dass je nach Benutzername unterschiedliche Fenster geöffnet und zugehörige Meldungen angezeigt werden. Diese Benutzernamen werden als „admin“, „kundendienst“, „buchhalter“ und „reinigung“ bezeichnet. Diese Benutzernamen können je nach Wunsch geändert oder weitere Benutzernamen hinzugefügt werden. 8. An einigen Stellen werden Nachrichtendialoge („JOptionPane.showMessageDialog“) verwendet. Diese Dialogfelder müssen entsprechend dem Inhalt der Nachrichten entsprechend angezeigt und organisiert werden. 9. Wir gehen davon aus, dass für andere Funktionen benötigte GUI-Komponenten (z. B. „KundenFrame.tc_txt“, „KundenFrame.vorn_txt“, etc.) vorhanden sind und mit dem Design verknüpft sind. Diese Komponenten müssen ordnungsgemäß identifiziert und korrekt verwendet werden.
Test Steps	**1. Verbindungstest:** - Es wird überprüft, ob der Code erfolgreich mit der Datenbank verbunden wurde, indem die Methode „conn()“ aufgerufen wird. **2. Produktdatentest:** - Die Methode „stat_produkt()“ wird aufgerufen, um zu prüfen, ob die Produktdaten erfolgreich abgerufen wurden. **3. Kundendatentest:** - Die Methode „stat_kunden()“ wird aufgerufen, um zu prüfen, ob die Kundendaten erfolgreich abgerufen wurden. **4. Benutzer-Login-Test:** - Durch den Aufruf der Methode „stat_loginUser()“ wird überprüft, ob der Benutzer mit Benutzername und Passwort angemeldet ist. **5. Buchungsdatentest:**

	- Die Methode „stat_res()“ wird aufgerufen, um zu prüfen, ob die Reservierungsdaten erfolgreich abgerufen wurden. **6. Kundenlöschtest:** - Die Methode „stat_kunden_delete()“ wird aufgerufen, um zu prüfen, ob ein bestimmter Kunde erfolgreich gelöscht wurde. **7. Client-Bearbeitungstest:** - Die Methode „stat_kunden_edit()“ wird aufgerufen, um zu prüfen, ob ein bestimmter Mandant erfolgreich bearbeitet wurde. **8. Kunden-Update-Test:** - Die Methode „stat_kunden_update()“ wird aufgerufen, um zu prüfen, ob die Kundendaten erfolgreich aktualisiert wurden. **9. Kundenadditionstest:** - Die Methode „stat_kunden_add()“ wird aufgerufen, um zu prüfen, ob ein neuer Kunde erfolgreich hinzugefügt wurde. **10. Reinigungsdatenaktualisierungstest:** - Die Methode „stat_reinigung_update()“ wird aufgerufen, um zu prüfen, ob die Reinigungsdaten erfolgreich aktualisiert wurden.
Verdict (Pass/Fail)	Pass
Actual Result	BEA DeviceReport.deviceName = “Reinigungdienst”

Rückverfolgbarkeit

Die Verbindung zwischen Testfällen und Anwendungsfall- und Webservice-Endpunkten ist im folgenden Diagramm dargestellt.

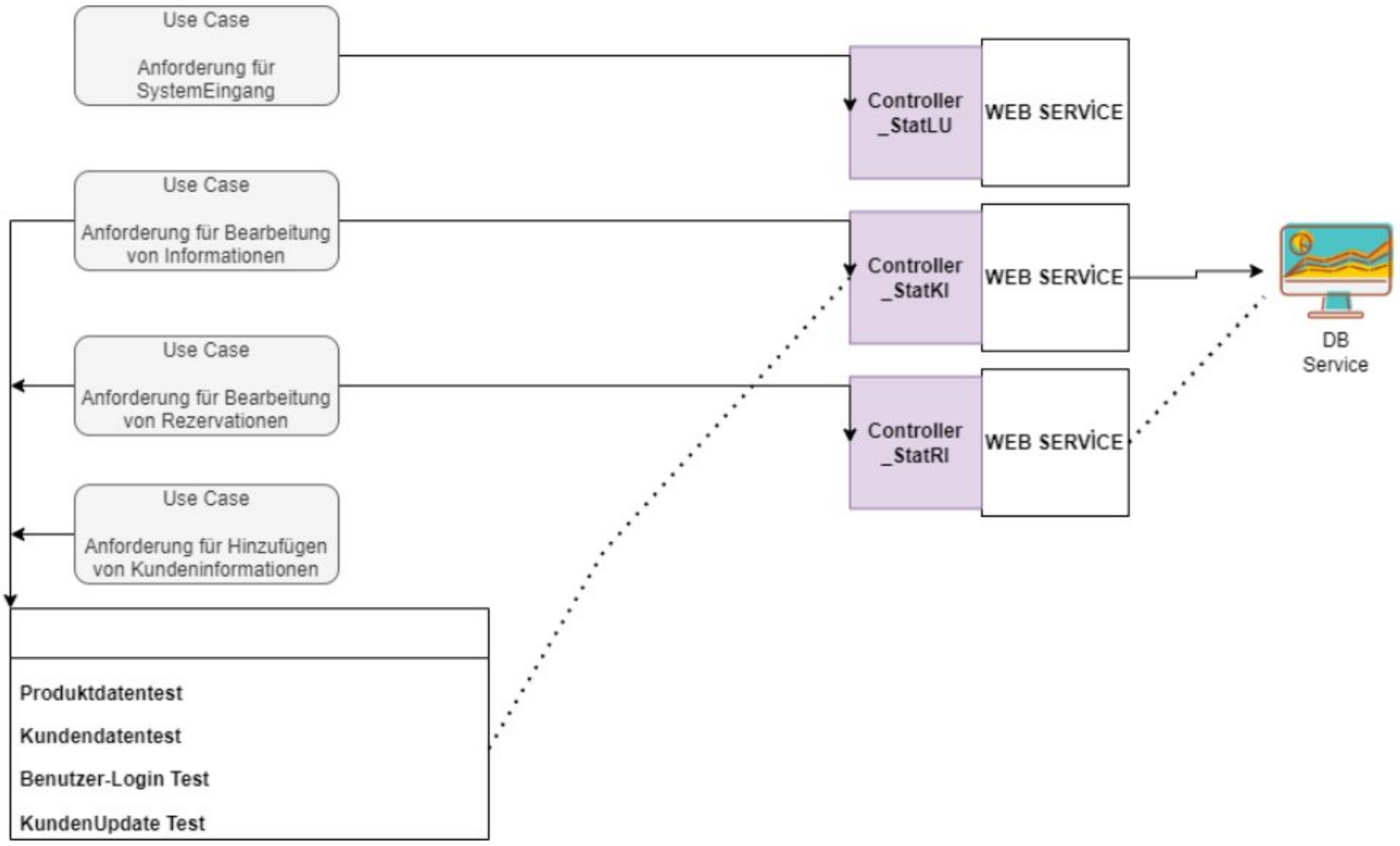

Abbildung 10: Rückverfolgbarkeit