Messaging-Applikation
Meilenstein M#4 - Unit Test

INF202-Projekt

Eren Naci Odabaşı
180501038
TMS-4
e180501038@stud.tau.edu.tr

Nihat Akın
180501008
TMS-4
e180501008@stud.tau.edu.tr

SoSe2023
Istanbul

Systemüberblick

Tokio wird im Backend dieses Projekts verwendet. Tokio ist eine beliebte asynchrone Laufzeitumgebung für Rust. Sie bietet eine Reihe von Werkzeugen und Abstraktionen zum Schreiben von effizientem und skalierbarem asynchronem Code. Es unterstützt die async/await-Syntax von Rust, ermöglicht nicht-blockierende I/O-Operationen und enthält einen robusten Netzwerk-Stack. Tokio wird häufig für die Entwicklung von Webservern, Web-Frameworks und nebenläufigen Anwendungen verwendet. Tokio-Test ist eine Test-Utility-Box für die asynchrone Laufzeitumgebung Tokio von Rust. Sie bietet Werkzeuge zum Testen von asynchronem Code, einschließlich Mocking asynchroner Operationen, Überprüfung der virtuellen Zeit und Vereinfachung von Assertions für asynchrone Ergebnisse und Futures.

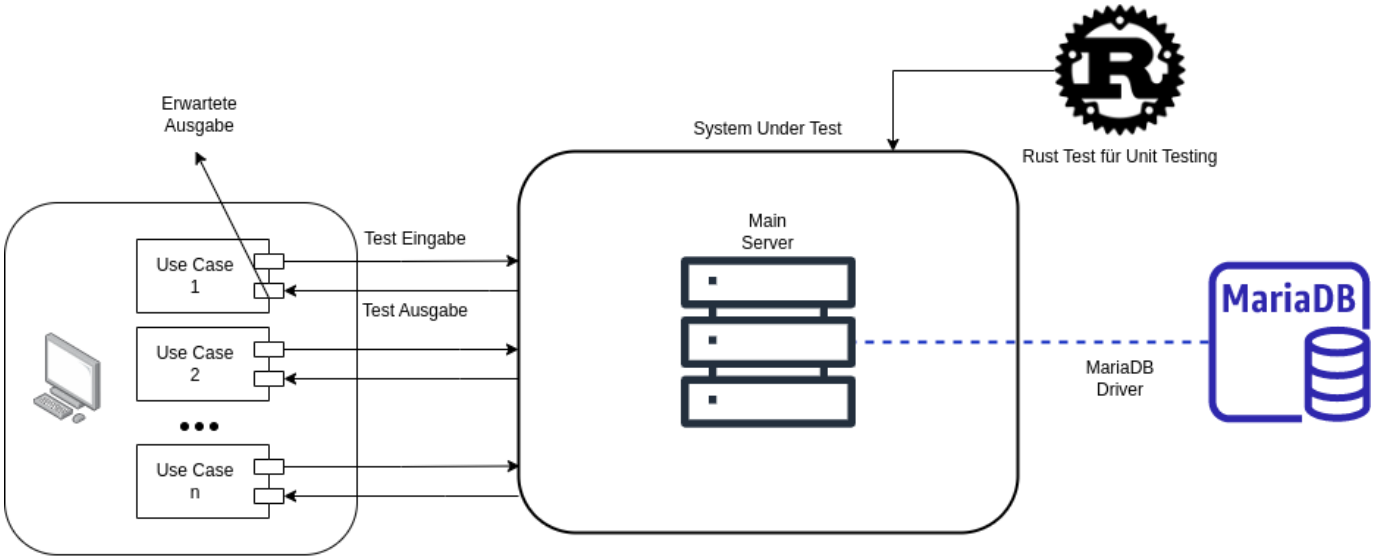

Abb.1: Testsystemüberblick

Systemtestfälle

TCP/IP-Server Unit Tests

In diesem Dokument wird beschrieben, wie Sie die Tests des entwickelten TCP/IP-Servers durchführen. Die Tests sind in zwei Kategorien unterteilt: Unit-Tests und Regressionstests. Während die Unit-Tests die einzelnen Komponenten des Servers separat testen, testen die Regressionstests alle Komponenten des Servers in integrierter Weise.

Unit-Tests dienen dazu, einzelne Komponenten des Servers isoliert zu testen. Diese Tests überprüfen, ob sich jede Komponente wie erwartet verhält, um sicherzustellen, dass der Server korrekt funktioniert.

Beispiele für Einheitstests

test_clear_user_tablet: Unittest für die Funktion zum Löschen der Datenbank.

System Test Case Name	test_clear_user_table
Preconditions	None
Test Steps	1. Lösche alle Daten in Datenbank 2. Versuch einer Benutzer zu greifen.

Abb.2: Test der Löschen der Datenbank

Test_get_port_number_from_id: Unit-Test, um die Adressinformationen aus der Datenbank anhand der E-Mail-Addresse des Benutzers als ID abzurufen.

System Test Case Name	test_get_port_number_from_id
Preconditions	None
Test Steps	1. Erstelle eine neue Benutzer 2. Versuch dieser Benutzer zu greifen.

Abb.3: Antrag auf Benutzergreifenstest

test_insert_new_user: Unit-Test zum Hinzufügen eines neuen Benutzers zur Datenbank.

System Test Case Name	test_insert_new_user_to_db
Preconditions	None
Test Steps	1. Erstelle eine neue Benutzer 2. Versuch dieser Benutzer zu greifen.

Abb.4: Test für Hinzüfugung des Benutzers

test_init_main_server: Unit-Test zur Untersuchung der Veränderung der aktiven Benutzeranzahl auf dem Server nach der Verbindungsherstellung.

System Test Case Name	test_init_main_server
Preconditions	None
Test Steps	1. Erstelle eine Listener 2. Zähle aktive Benutzer 3. Starte Server 4. Zähle aktive Benutzer 5. Vergleich erste und ende Benutzeranzahl

Abb.4: Test für Main-Server

Resultät der Testen:

Die Ausgabe des Systems als Ergebnis einer Anfrage, die mit einer falschen json-Struktur erstellt wurde:

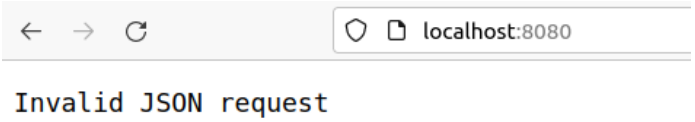

Abb.5: Ungültige json-Anfrage

Terminalausgabe des Systems, wenn versucht wird, einen neuen Server mit der gleichen Serveradresse zu öffnen:

```
    Running `target/debug/messaging_app`
Error: Os { code: 98, kind: AddrInUse, message: "Address already in use" }
```

Abb.5: Fehler: Adresse ist bereits in Gebrauch

Regressionstests

Regressionstests werden verwendet, um alle Komponenten des Servers auf integrierte Weise zu testen. Diese Tests simulieren reale Szenarien, um sicherzustellen, dass alle Funktionen des Servers wie erwartet funktionieren.

Regressionstests können das folgende Szenario abdecken:

Szenario: Ein Client möchte eine Nachricht an einen bestimmten Empfänger weiterleiten, indem er eine HTTP-Anfrage sendet.

Schritt 1: Der Server wird gestartet.

Schritt 2: Der Client sendet die HTTP-Anfrage an den Server.

Schritt 3: Der Server empfängt die Anfrage, findet die IP-Adresse und den Port des Empfängers und leitet die Anfrage an den Empfänger weiter.

Schritt 4: Der Empfänger sendet eine HTTP-Antwort an den Server.

Schritt 5: Der Server prüft, ob die Übertragung erfolgreich war, indem er die Antwort an den Client weiterleitet.

Schritt 6: Der Server fährt herunter.

Dieser Regressionstest verifiziert, dass der Server die HTTP-Anfrage erhalten, den richtigen Empfänger gefunden und die Antwort erfolgreich weitergeleitet hat.

```
running 5 tests
test tests::test_lookup_receiver_ip ... ok
test tests::test_lookup_receiver_port ... ok
test tests::test_process_request ... FAILED
test tests::test_send_forward_request ... ok
test tests::tests::test_regression ... FAILED
```

Abb.6: Aktuelle Ergebnisse der Einheitstests

Wie aus der Ausgabe des Testers, der mit der Rust-tokio-test-Kiste erstellt wurde, ersichtlich ist, sind einige Funktionen des Servers nicht optimal und funktionieren nicht richtig.

Rückverfolgbarkeit

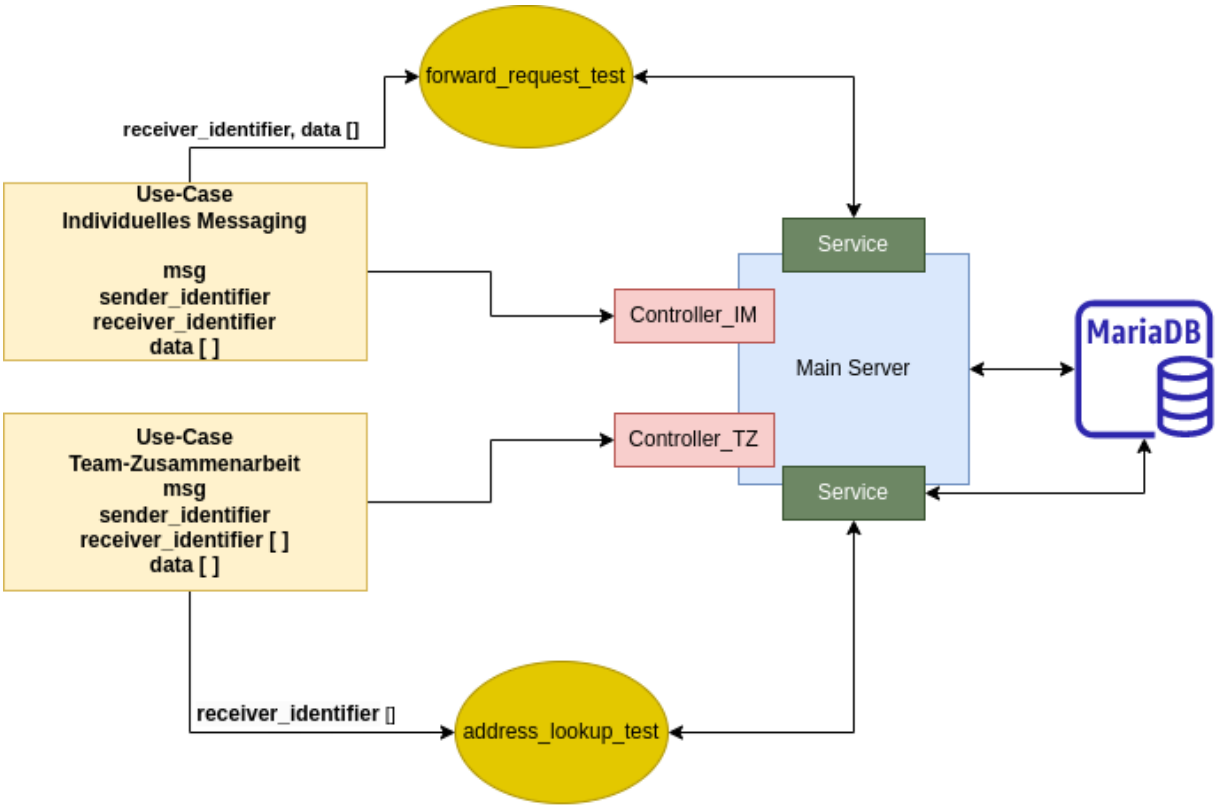

Abb. 7: Beziehung zwischen Test- und Usecases