TAU INF202 Software Engineering

Individuelles Projekt

Klinikverwaltungssystem
Systemtest-Dokument

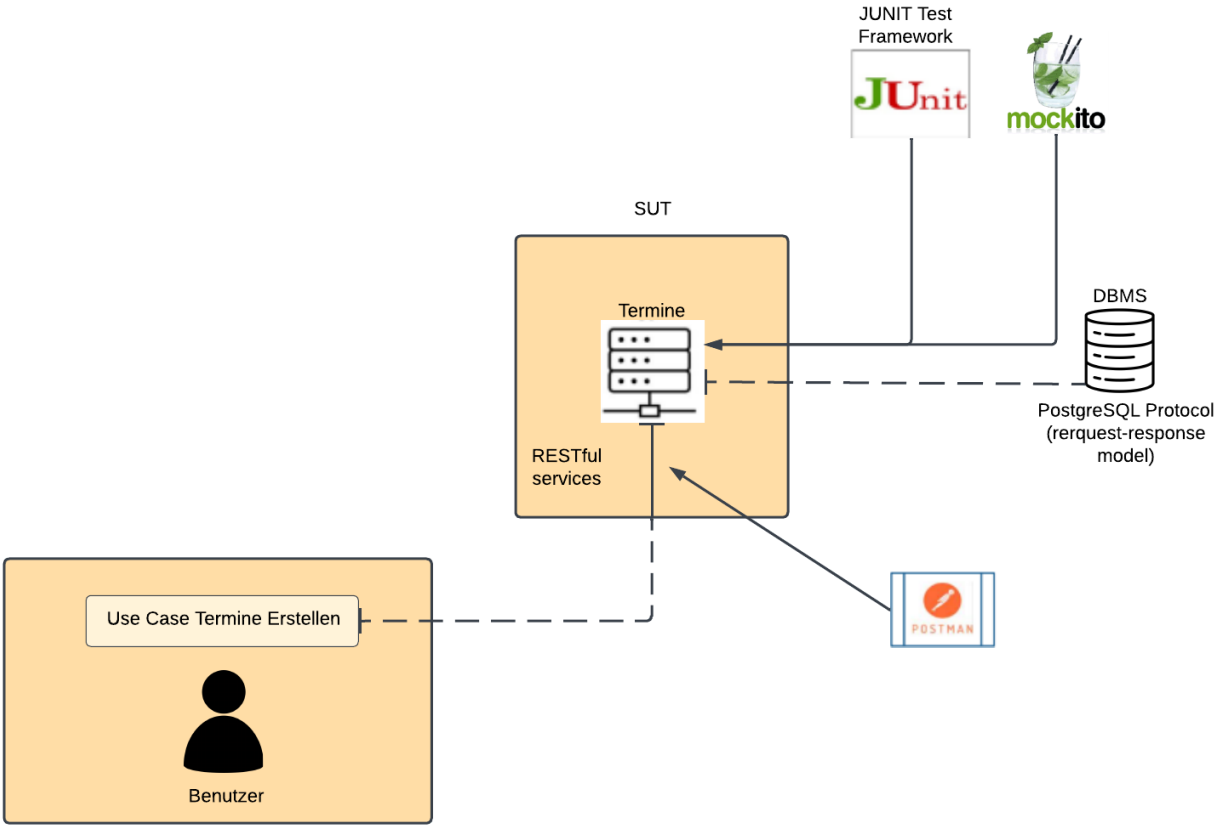
Verantwortliche/r:

İpek Yılmaz, e200503052@stud.tau.edu.tr

Haluk Harun Gündoğan, e200503031@stud.tau.edu.tr

Stakeholder: DI. Ömer Karacan, omer.karacan@tau.edu.tr

1.Systemüberblick



In unserem System wird die RESTful Standard zur Datenübertragung benutzt, indem die Daten über die spezifizierten API-Endpoint dem Backend im JSON-Format zugestellt werden. Wenn die Daten im Backend erreicht sind, muss das System zuerst kontrollieren, ob die gegebenen Daten im gewünschten Format sind. Nach diesen Kontrollen, falls die Kontrollen bestanden worden sind, werden die Daten in die Datenbank geschrieben. Um etwaige Lücken, Fehler oder fehlende Anforderungen vor der Bereitstellung der Anwendung zu identifizieren, muss man das System testen.

Beim Testen werden wir zuerst Integrationstest durchführen, indem „Termine erstellen" Use-Case getestet werden. Dann werden wir komponentenbasierte Tests durchführen, indem wir unsere Service Klassen und Repository Klassen mit Junit 5 und Mockito testen. Das Junit 5 Framework wird beim Testen der Repository Klassen benutzt. Nach dieser Testingsphase wird sichergestellt, dass es keine Probleme bei den Repository Klassen bzw. Datenbankintegration gibt. Demnach werden die Service Klassen mithilfe der Mockito Framework getestet. Da die Repository Klassen bis zu dieser Phase die Tests bestanden haben, müssen wir diese Klassen beim Testen der Service Klassen nicht wieder testen. Daher benutzt man Mockito Framework, um die Repository Klassen simulieren bzw. „mocken”.

2. Systemtestfälle

System Test Case ID Name	canErstellNeuenTermin Durch dieses Use-Case erstellt das System einen neuen Termin

Preconditions	• Web Service is given:
	  ○ Request: POST erstellNeuenTermin(Termine, termin)
	  ○ Response: 200 OK Successful HTTP Request.
	  ■ Data „Termine Erstellen"
	  ● Long termine_id // unique
	  ● LocalDateTime terminDatum
	  ● String zeit
	  ● Doktor doktor
	  ○ Long doktor_id
	  ● Patient patient
	  ○ Long patient_ausweisnummer
	  ■ Wo "terminDatum" ist einzigartig bzw. unique.
	  ■ Wo "zeit" ist einzigartig bzw. unique
	• App Port Nummer default(8080).
	• Das System ist Betriebsbereit.

Test Steps	**1. POST erstellNeuenTermin(Termine , termin) ist aufgerufen mit den oben beschriebenen Spezifikationen.** **2. Die Einzigartigkeit von ,,terminDatum" ist gecheckt.** **3. Die Einzigartigkeit von ,,zeit" ist gecheckt.**
Post-Condition	**POST wenn der ,,terminDatum" einzigartig ist.** **POST wenn der ,,zeit" einzigartig ist.**

Test Data	{
	"terminDatum": "2023-06-24T14:56:49",
	"zeit": "9.25",
	"patient": {
	"ausweisnummer": 39272278790
	},
	"doktor": {
	"doktor_id": 1
	}
	}

Expected Result	**200 OK (Standardwort für erfolgreiche HTTP-Anfragen)**
Actual Result	**200 OK (Standardwort für erfolgreiche HTTP-Anfragen)**
Verdict (Pass/Fail)	**Pass**
Verified UC & Req. IDs	**,,/REQ_3/ Termine erstellen"**

In unserem System Test Case „canErstellNeuenTermin” haben wir das ganze System mit den gegebenen Konditionen getestet. Zuerst haben wir die Voraussetzungen definiert, indem das Format des gesendeten Termins definiert wird. Nämlich müssen wir eine einzigartige Termin-ID haben, dann muss ein Datum definiert werden, das auch einzigartig ist. Demnach ist eine einzigartige Zeit definiert. Zwischen Doktor-Termine und Patient-Termine existiert eine Beziehung, deshalb werden die Doktor ID und Patientausweisnummer von der Terminetabelle als Fremdschlüssel benutzt. Bei den Spring Data JPA definierten Standardisierungen muss man die Foreign Keys wie oben darstellen. Wenn das System fürs Testen bereit ist, kann man die Teststeps durchführen.

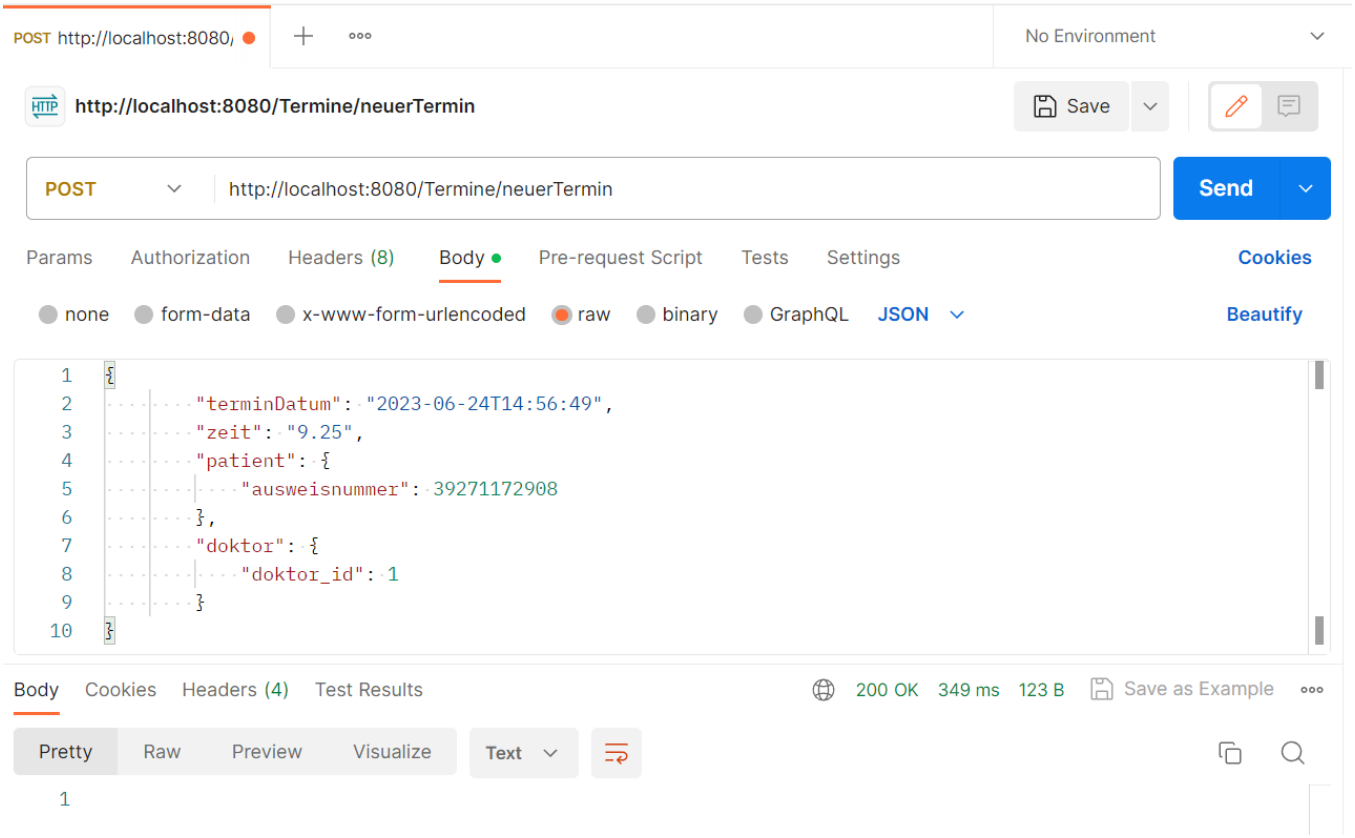


Abbildung 1: Postman Ausgabe

Component Integration Test Case ID Name	**checksWhetherDatumExists beschafft Termine database interface**

Preconditions	• Databasetabelle “Patient” enthält eine Zeile für:
	  ○ ausweisnummer = 39272278790
	  ○ name = “Ali”
	  ○ nachname =”Can”
	  ○ phonenummer = “05424237478”
	• Databasetabelle “Doktor” enthält eine Zeile für:
	  ○ doktor_id = 1
	  ○ doktor_name = “Mehmet”
	  ○ doktor_nachname = “Ozer”
	  ○ abteilung = “Kordiologie”
	• Databasetabelle “Termine” enthält eine Zeile für:
	  ○ termin_id = 1
	  ○ terminDatum = "2023-06-24T14:56:49"
	  ○ zeit = “9.25”
	  ○ doktor_id = 1
	  ○ patient_ausweisnummer = 39272278790

Test Steps	**1. Es soll sichergestellt werden, dass ein Patient mit gegebener ,,ausweisnummer” in der Datenbank in der Patienttabelle existiert.** **2. Es soll sichergestellt werden, dass ein Doktor mit gegebener ,,doktor_id” in der Datenbank in der Doktortabelle existiert.** **3. Es soll sichergestellt werden, dass ein Termin mit gegebener ,,terminDatum” in der Datenbank in der Terminetabelle existiert.**
Post-Condition	**none**
Test Data	**terminDatum**
Expected Result	**termineRepository.existsTerminByDatum(terminDatum) = True**
Actual Result	**termineRepository.existsTerminByDatum(terminDatum) = True**
Verdict (Pass/Fail)	**Pass**
Verified UC & Req. IDs	**,,/REQ_3/ Termine erstellen"**

Mit dem Komponentintegrationstest „checksWhetherDatumExists” wollten wir testen, ob die TermineRepository Klasse die Kommunikation zwischen der Applikation und der Datenbank bieten kann. Das Hauptziel dieses Testcases ist zu sehen, ob die TermineRepository Klasse die richtige Queries fürs Checken des einzigartigen Datums der Termine bereitstellt. Der gleiche Test wird auch bei der Einzigartigkeit der „zeit” benutzt.

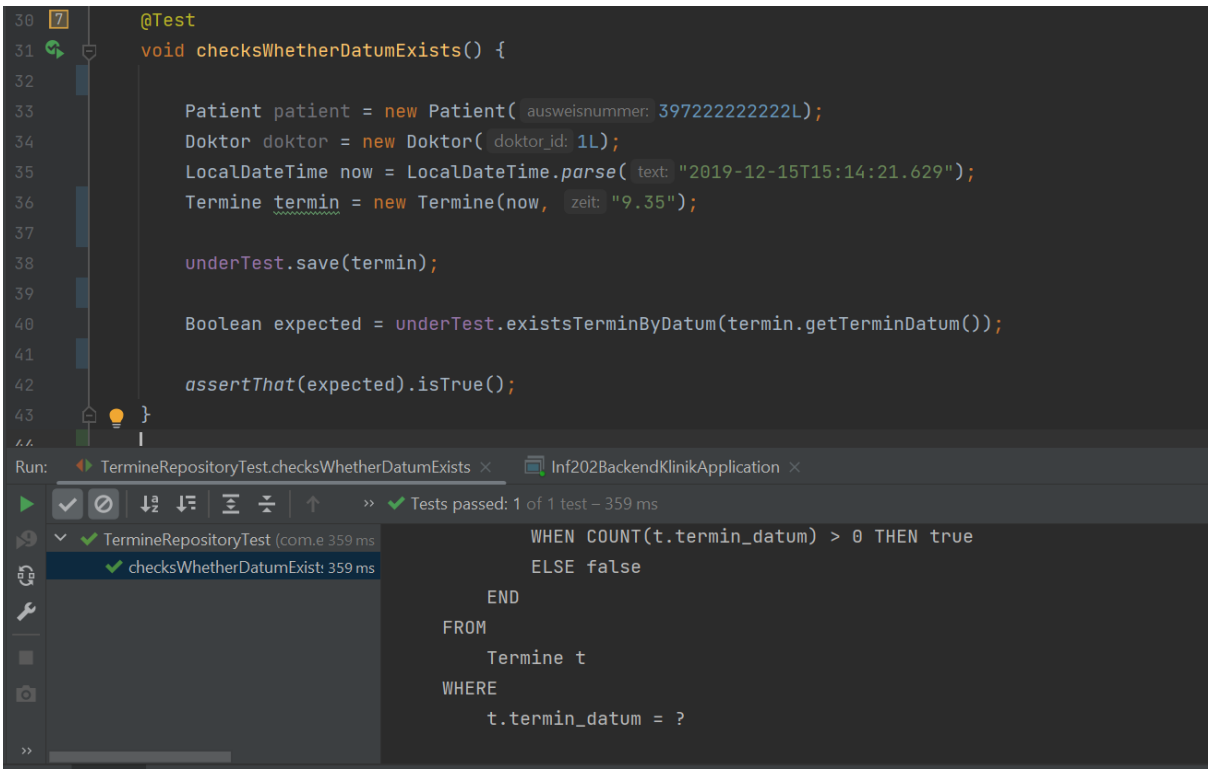
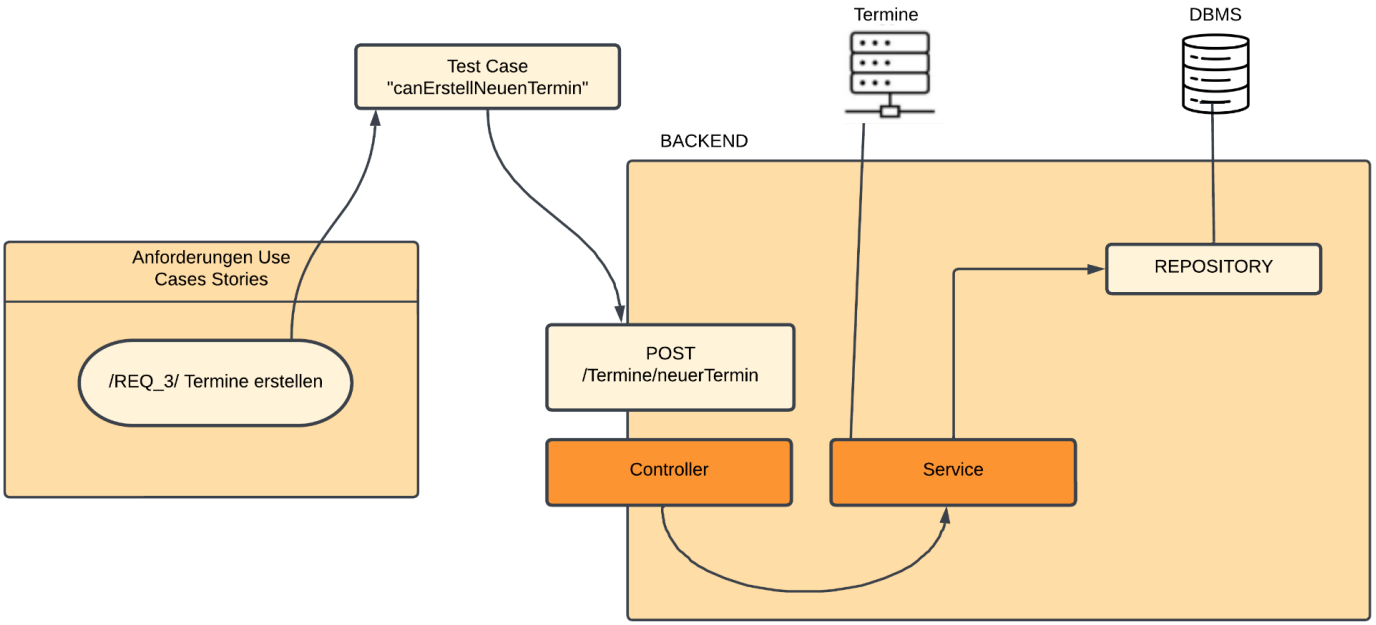


Abbildung 2: Junit 5 Ausgabe

3.Rückverfolgbarkeit



Bei der Rückverfolgbarkeit, ist es gewünscht zu visualisieren wie der Test „canErstellNeuen Termin” parallel zum MS#3 „Architekturspezifikation” funktioniert. In der Architekturspezifikation haben wir „/REQ_3/ Termine Erstellen” Anfrage definiert, die für den Benutzer einen neuen Termin erstellt. Um zu kontrollieren, ob diese Anfrage mit dem System wie gewünscht integriert ist, wird der Test Case „canErstellNeuenTermin” erstellt. Da diese zu einem POST Request entspricht, werden die im Testcase definierten Daten über die gegebenen API-Endpoint gesendet bzw. gepostet. Nachdem diese Daten zum Service gelanden sind, wird die Business Logik getestet. Nämlich, „Sind die neu erstellenden Termine einzigartig?”. Schließlich wird überwacht, ob die Daten in die Datenbank erfolgreich geschrieben werden.