ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektbericht REST-API

vorgelegt von
Arne Kreuz,
Joshua Nestler,
Moritz Schönenberger

betreut und begutachtet von
Prof. Dr. Reinhard Brocks

Saarbrücken, 17. Juni 2024

Inhaltsverzeichnis

1 Projekt

1.1 Zielsetzung

Write

1.2 Projektbeiträge

Write

1.3 Verwendete Technologien

Technologie	Version	Link
Java	17	https://www.java.com/
Maven	3.8.5	https://maven.apache.org/
Jakarta EE	11.0.0	https://jakarta.ee/
Wildfly	32.0.0	https://www.wildfly.org/
Spring Boot	3.2.4	https://spring.io/projects/spring-boot
.NET	8.0	https://dotnet.microsoft.com/
ASP.NET Core	8.0	https://dotnet.microsoft.com/apps/aspnet
H2	2.2.224	https://www.h2database.com/
MariaDB	11.3	https://mariadb.org/
JMeter	5.4.1	https://jmeter.apache.org/
Postman	10.23.4	https://www.postman.com/
Docker	26.1.2	https://www.docker.com/
IntelliJ IDEA	2024.1.2	https://www.jetbrains.com/idea/
Rider	2024.1.2	https://www.jetbrains.com/rider/
Git	2.45.0	https://git-scm.com/
GitHub	-	https://github.com/
GitHub Actions	-	https://github.com/features/actions
OpenAPI	3.0.3	https://www.openapis.org/
Mermaid	10.9.0	https://mermaid-js.github.io/mermaid/
LaTeX	2023	https://www.latex-project.org/

Tabelle 1.1: Verwendete Technologien

2 Spezifikation

2.1 OpenAPI

Write

3 Implementierung Webserver

3.1 Jakarta EE

Als Referenzimplementierung für dieses Projekt wurde `Jakarta EE` verwendet. Jakarta EE ist eine Sammlung von Spezifikationen für Enterprise-Java-Anwendungen. Es ist eine Weiterentwicklung von Java EE, welches von Oracle entwickelt wurde. Jakarta EE wird von der Eclipse Foundation entwickelt und ist Open Source.

Write

3.1.1 Probleme

Abhängigkeiten vom Webserver

Jakarta EE benötigt einen seperaten Webserver, um die Anwendung auszuführen. Die Konfiguration und das Setup sind stark abhängig von dem verwendeten Webserver. In diesem Fall wurde `Wildfly`[1] verwendet.

Die meisten dieser Webserver sind nicht dazu ausgelegt, eine einzelne Anwendung zu hosten. Dies macht es schwieriger, die Anwendung in einer Microservice-Architektur zu betreiben.

Um dieses Projekt mit diesem Webserver zu betreiben, ist es notwendig, ein Konfigurations-Skript zu schreiben, welches in den Bauprozess des Docker-Abbilds integriert wird.

- Konfiguration und Setup ist stark abhängig von dem verwendeten Webserver (in diesem Fall `Wildfly`)

3.2 Spring Boot

Write

3.3 ASP.NET Core

Write

[1] `https://www.wildfly.org/` | Abgerufen: 2024-05-10, 12:44 Uhr

4 Tests

4.1 Load Tests

Anders als funktionale Tests inspizieren Load Tests die Performance des Systems unter hoher Last, beispielsweise bei vielen nebenläufigen Zugriffen. Zur Umsetzung dieser Tests wird JMeter & 5.4.1 & `https://jmeter.apache.org/` eingesetzt.

4.1.1 Starten von JMeter

JMeter kann im GUI- oder CLI-Modus gestartet werden. Der GUI-Modus wird zum Erstellen und Debuggen von Tests eingesetzt, welche anschließend im CLI-Modus ausgeführt werden können.

JMeter startet standardmäßig im GUI-Modus. Um Tests im CLI-Modus auszuführen, kann folgender Befehle eingesetzt werden.

```
jmeter -n -t testplan/TestPlan.jmx -l "results/result.jtl" -j "
    logs/logs.log"
```

Die eingesetzten Optionen haben folgende Bedeutung:

- `-n` - CLI-Modus
- `-t` - Pfad zum Testplan
- `-l` - Pfad zur Testlogdatei (Testergebnisse)
- `-j` - Pfad zur Jmeterlogdatei (Informationen über Testausführung und aufgetretene Fehler)

4.1.2 Erstellung von Testplänen

Tests werden in Form eines Testplans geschrieben. Ein Testplan ist ein Baum, der Elemente aus verschiedenen Kategorien enthalten kann (siehe `https://jmeter.apache.org/usermanual/test_plan.html`).

- Thread Group
  Kontrolliert wie oft Sampler ausgeführt werden, insbesondere wie viele Anfragen nebenläufig stattfinden.
- Logic Controllers
  Beeinflusst die Ausführungsreihenfolge der Sampler.
- Samplers
  Die auszuführenden Tests, z.B. HTTP-Anfragen.
- Listener
  Speichert die Ergebnisse eines Samplers und kann diese im GUI-Modus grafisch aufbereiten.

- Configuration Elements
  Stellen Daten für Tests zur Verfügung, unter anderem durch Definition von Konstanten, Erzeugung zufälliger Werte oder Einlesen externen Dateien.

Grundsätzlich beinhaltet ein Testplan `Thread Groups`. Jeder Thread steht für einen User. `Thread Groups` enthalten `Sampler`, welche von jeden Thread von oben nach unten ausgeführt werden. `Samplers` enthalten (einen) Listener, welcher die Ergebnisse aufzeichnet und im GUI-Modus visualisieren kann. `Configuration Elements` werden nach Bedarf als Kinder von `Thread Groups` oder `Samplers` definiert.

4.1.3 Verwendung von Variablen

JMeter ermöglicht die Definition von Variablen, um Tests besser wart- und anpassbar zu machen. Hierzu kann etwas das Configuration Element `User Defined Variables` eingesetzt werden. In diesem Element definierte Konstanten können grundsätzlich mit der Syntax `${Variablenname}` referenziert werden.

Leider wird diese Art Variablen zu referenzieren nicht überall unterstützt. Beispielsweise funktioniert dies im Körper einer HTTP Request und zur Angabe der Widerholungen eines Tests, aber nicht bei der Angabe von Dateipfaden.

In solchen Fällen kann der Wert einer Variable oftmals mit Groovy[1] ausgelesen werden. Dies ist mit folgender Syntax möglich: `${__groovy(vars.get("Variablenname"))}`. Leider funktioniert auch dieser Ansatz nicht in allen Fällen.

Der Einsatz von Groovy ermöglicht die Berechnung von Konstanten in Abhängigkeit von anderen Konstanten. Die abhängige Konstante muss in einem eigenen `User Defined Variables`-Element definiert werden. Dieses muss sich im Baum unter dem ursprünglichen Konfigurationselement befinden. Wird die Konstante im gleichen Element oder einem im Baum höherstehenden Element definiert, scheitert die Definition der abhängigen Variable und sie wird als leerer String behandelt.

Um Fehler beim Einsatz von Variablen zu untersuchen, kann der Sampler `Debug Sampler` eingesetzt werden. Dieser wertet bei Ausführung alle verfügbaren Variablen aus, sodass deren Werte überprüft werden können. Die Werte sind im GUI-Modus als `Response Data` des zum Sampler hinzugefügten Listener einsehbar.

4.1.4 Relative Pfade

JMeter bietet keine vollständige Unterstützung für relative Pfade.

Die Angabe relativer Pfade ist im GUI-Modus nicht möglich. Ersetzt man im durch den GUI-Modus erzeugten Testplan, einer XML-Datei, absolute durch relative Pfade, so werden diese im CLI-Modus korrekt aufgelöst. Öffnet man den manipulierten Testplan erneut im GUI-Modus, kann JMeter die relative Pfade jedoch nicht auflösen.

Um trotzdem absolute Pfade zu vermeiden, kann ein absoluter Basispfad als Konstante definiert werden. Pfade werden dann als Konkatenation dieser Variable und des gewünschten relativen Pfades definiert. Somit muss nur der Basispfad angepasst werden, um die Ausführung auf einer anderen Maschine zu ermöglichen.

Listing 4.1: Workaround zur Angabe relativer Pfade

```
${__groovy(vars.get("PathBase"))}/listeners
```

[1] `https://www.groovy-lang.org/`

5 Workflow

Write

6 Fazit

Write

6.1 Zusammenfassung

Write

6.2 Lessons Learned

Write

6.3 Ausblick

Write

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

REST Representational State Transfer

Anhang

A REST-Endpunkte

A.1 User

Methode	Endpunkt	Parameter	Beschreibung
GET	/users	-	Gibt alle Benutzer zurück
GET	/users/{id}	id	Gibt den Benutzer mit der ID zurück
POST	/users	-	Erstellt einen neuen Benutzer
PUT	/users/{id}	id	Aktualisiert den Benutzer mit der ID
DELETE	/users/{id}	id	Löscht den Benutzer mit der ID

Tabelle A.1: User-Endpunkte

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.23, März 2022). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt