SEW

IT-Medientechnik

NVS

Informatik

RESTful Web

Representational State Transfer

TWS

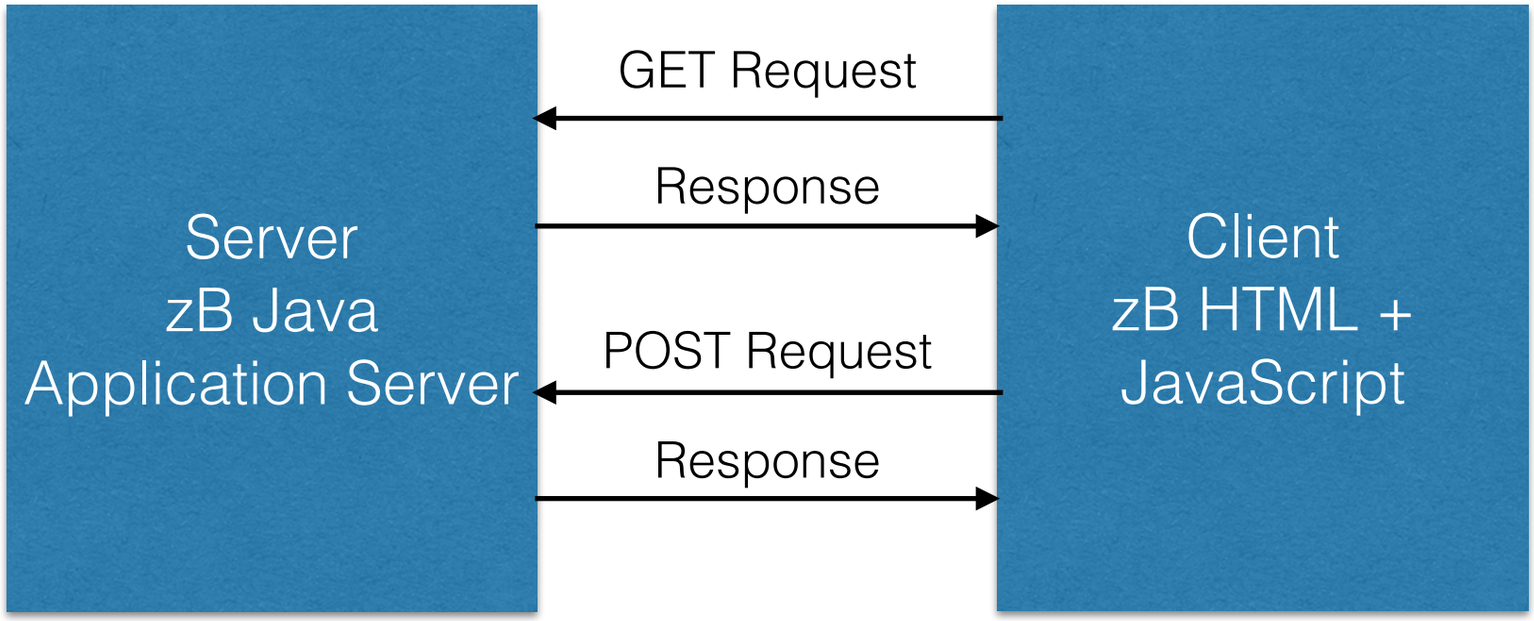
Warum REST?

- REST ist die Lingua Franca des Webs
- Heterogene (verschiedenartige) Systeme können mit REST kommunizieren, unabhängig von der Technologie der beteiligten Systeme
- REST Resourcen werden im Web angeboten: zB Google Maps, Twitter, Youtube, Facebook

http://www.programmableweb.com/apis/directory

Funktionsweise

- Sämtliche Informationen im Web sind Ressourcen, auf die mittels HTTP zugegriffen werden kann.



Anforderungen

- **Adressierbarkeit**: Jede Ressource muss über einen eindeutigen Unique Resource Identifier (kurz URI) identifiziert werden können. Ein Kunde mit der Kundennummer 123456 könnte also zum Beispiel über die URI http://localhost/customers/123456 adressiert werden.

- **Zustandslosigkeit**: Die Kommunikation der Teilnehmer untereinander ist zustandslos. Dies bedeutet, dass keine Benutzersitzungen (etwa in Form von Sessions und Cookies) existieren, sondern dass bei jeder Anfrage alle notwendigen Informationen wieder neu mitgeschickt werden müssen.

- **Einheitliche Schnittstelle**: Auf jede Ressource muss über einen einheitlichen Satz von Standardmethoden zugegriffen werden können. Beispiele für solche Methoden sind die Standard-HTTP-Methoden wie GET, POST, PUT, und mehr.

- **Entkopplung von Ressourcen und Repräsentation**: Das bedeutet, dass verschiedene Repräsentationen einer Ressource existieren können. Ein Client kann somit etwa eine Ressource explizit beispielsweise im XML- oder JSON-Format anfordern.

https://www.mittwald.de/blog/webentwicklung-webdesign/webentwicklung/restful-webservices-1-was-ist-das-uberhaupt

HTTP Methoden

GET	READ: Lesen einer bestimmten Ressource
POST	CREATE (und UPDATE): Erstellen oder Ändern ohne einer ID
PUT	(CREATE und) UPDATE: Erstellen oder Ändern mit einer bekannten ID
DELETE	DELETE: Löschen einer Ressource

Beispiele

Request	Response
GET /products HTTP/1.0 Accept: application/json	HTTP/1.0 200 OK Content-Type: application/json Content-Length: 1234 [{ uri: "http://ws.mydomain.tld/products/1000", name: "Gartenstuhl", price: 24.99 }, { uri: "http://ws.mydomain.tld/products/1001", name: "Sonnenschirm", price: 49.99 }]

Beispiel POST

Request	Response
POST /products HTTP/1.0 Content-Type: application/json Content-Length: 38 { name: "Sandkasten", price: 89.99 }	HTTP/1.0 201 Created Location: http://ws.mydomain.tld/products/1002

Begriffe

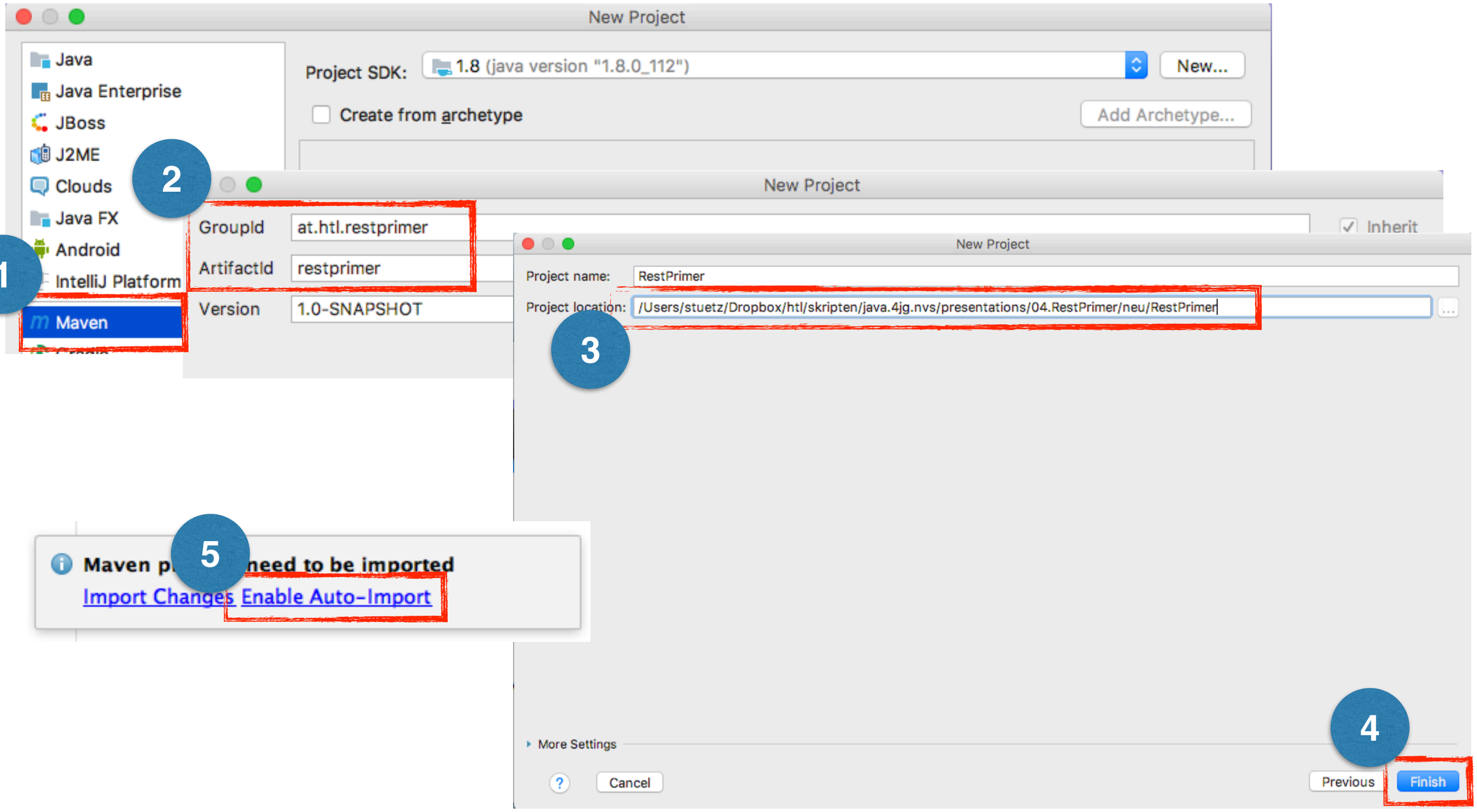
- **Safety** (Sicherheit): Daten werden nicht verändert
- **Idempotence** (Idempotenz): Die Ressource behält auch bei mehrmaligen Aufruf den gleichen Zustand.

HTTP Method	Safe	Idempotent
GET	✅	✅
POST	❌	❌
PUT	❌	✅
DELETE	❌	✅
OPTIONS	✅	✅
HEAD	✅	✅

RESTful API

GET PUT POST DELETE

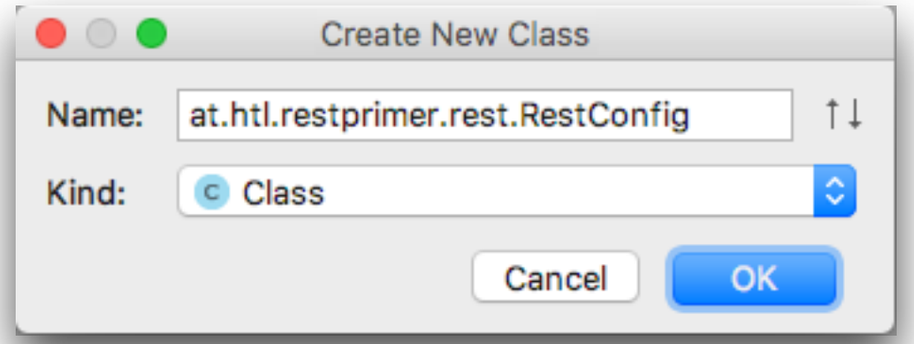
Ein erstes Beispiel

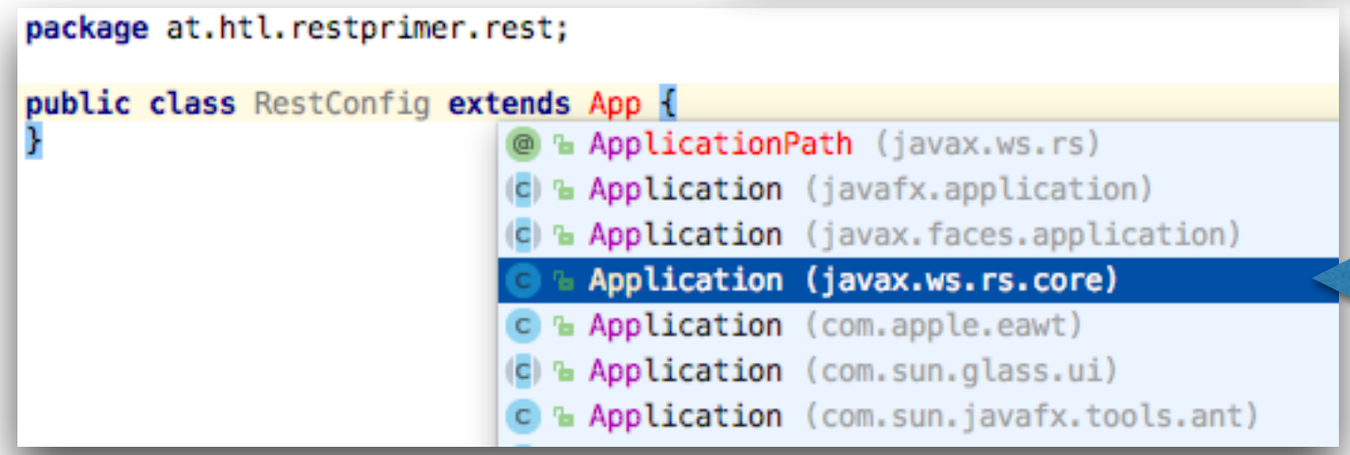


pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <packaging>war</packaging>

    <groupId>at.htl</groupId>
    <artifactId>restprimer</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>8.0.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <build>
        <finalName>restprimer</finalName>
    </build>

</project>
```

REST-Konfiguration



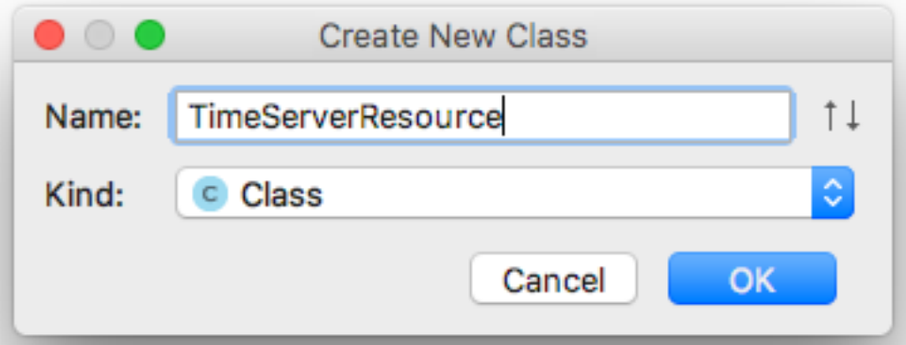


BEACHTE: Wir verwenden javax.ws.rs !!!

```java
package at.htl.restprimer.rest;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath(„api")
public class RestConfig extends Application {
}
```

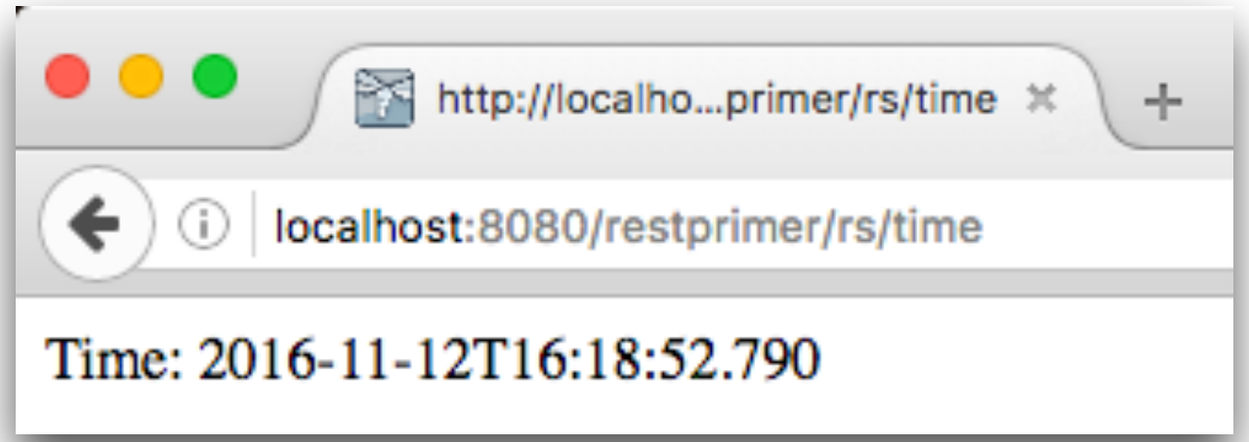
Das ist die komplette Konfiguration, um REST-Ressourcen verwenden zu können. Ev. könnte man noch CORS konfigurieren. Warum wohl?

REST Ressource




```java
package at.htl.restprimer.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import java.time.LocalDateTime;

@Path("time")
public class TimeServerResource {

    @GET
    public String time() {
        return "Time: " + LocalDateTime.now();
    }
}
```

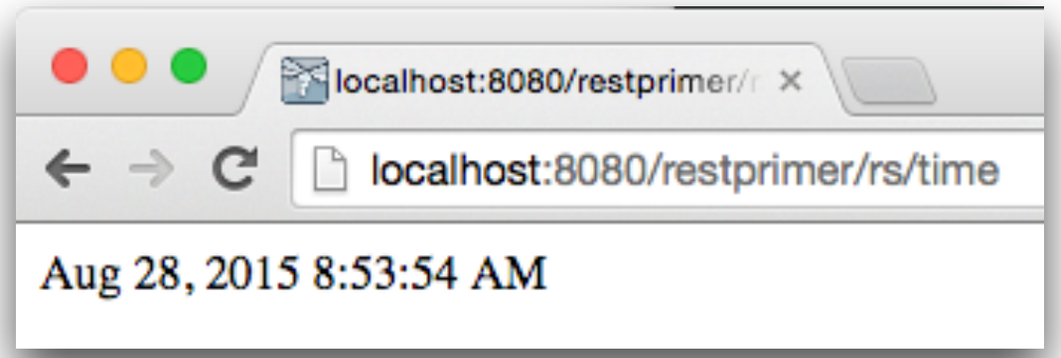
Testlauf

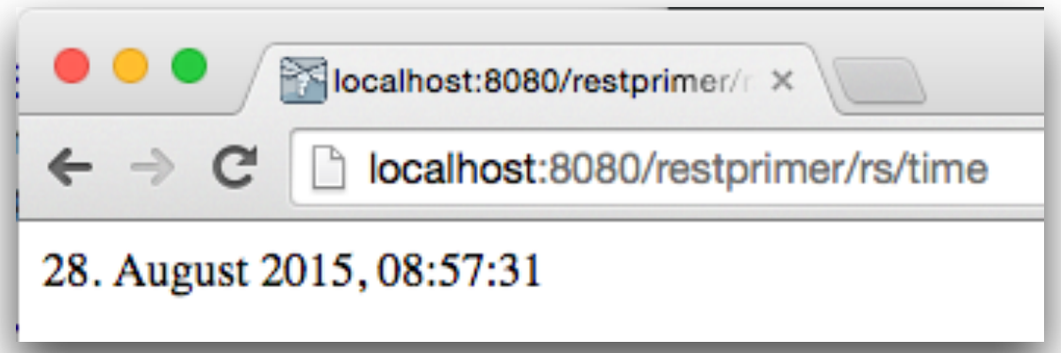


Nicht vergessen: Vor dem ersten Start den Application Server („Run/Debug“) konfigurieren

Verbesserung der Ausgabe

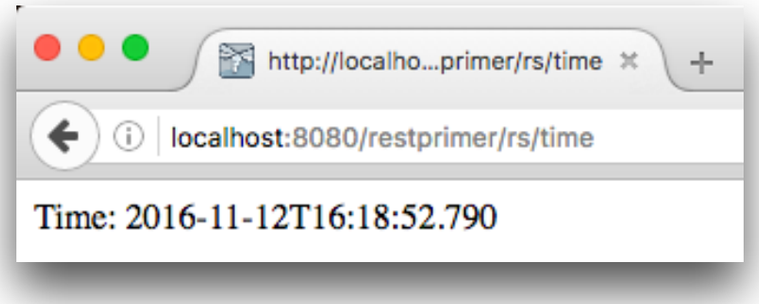
Wie muss hier die Rückgabe aussehen?



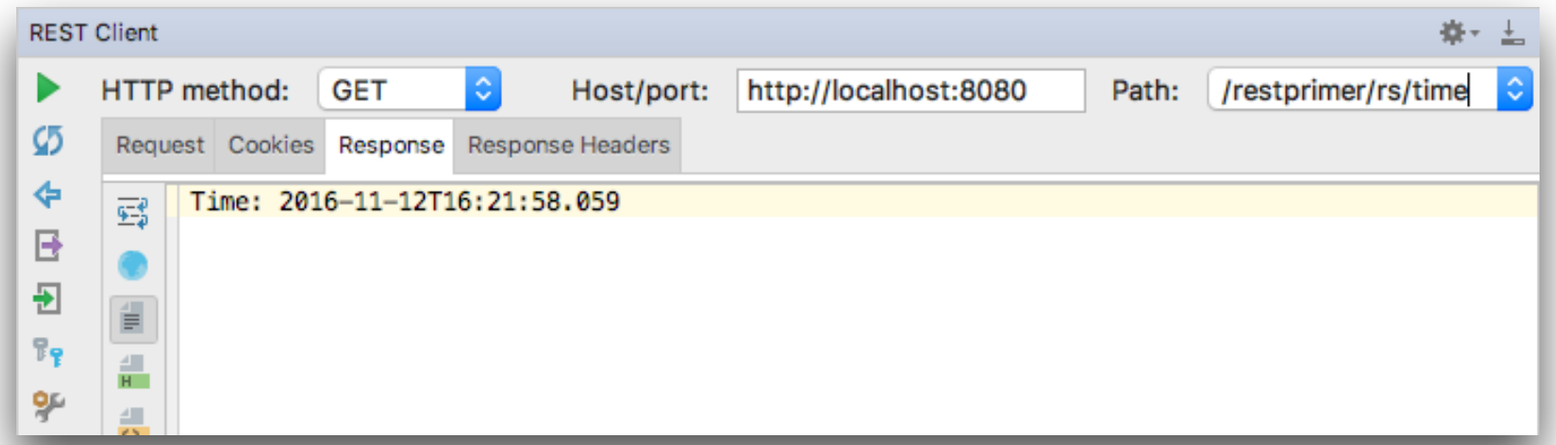


Mögliche Clients

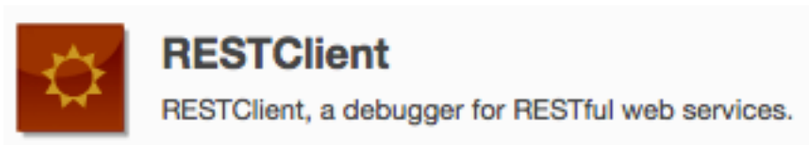
- Web-Browser

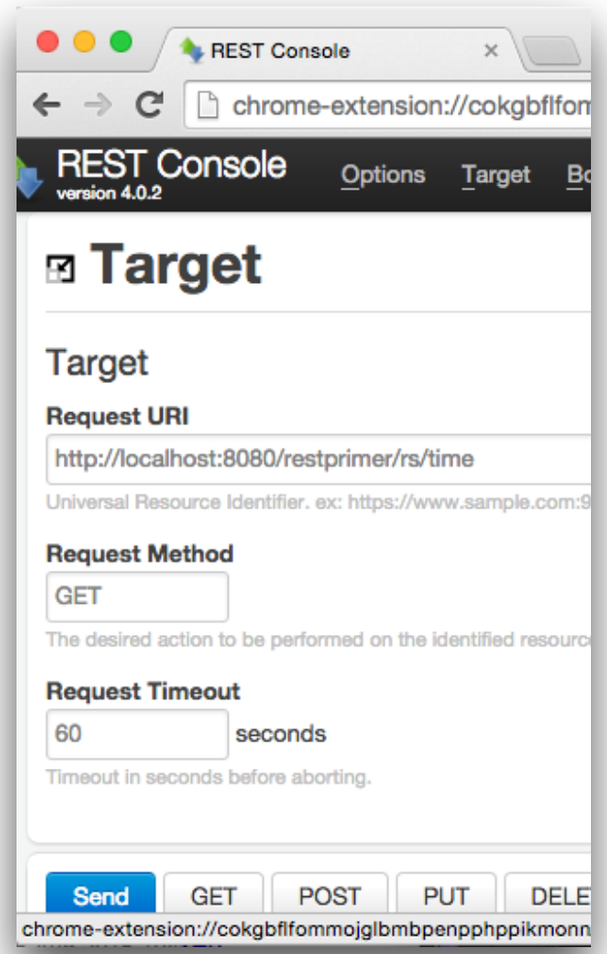


- REST Client in IDE



- Plugins in WebBrowser
  oder Standalone REST-Clients





RESTClient

RESTClient, a debugger for RESTful web services.

File Authentication Headers View Favorite Requests Setting **RESTClient**

[-] Request

Method GET URL http://localhost:8080/restprimer/rs/time ☆ SEND

Body

Request Body

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)

```
28. August 2015, 09:10:38
```

Hier sieht man sehr gut die Header-Informationen wie Status Code und Content-Type

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Res

```
1.  Status Code      : 200 OK
2.  Connection       : keep-alive
3.  Content-Length   : 25
4.  Content-Type     : text/html
5.  Date             : Fri, 28 Aug 2015 07:10:38 GMT
6.  Server           : WildFly/9
7.  X-Powered-By     : Undertow/1
```

Eigener Java-Client

- Der Nachteil der vorher genannten Clients besteht darin, dass die REST-Abfragen i.N. nicht wiederholbar sind. D.h. bei jeder Änderung im Code, sind die Tests manuell durchzuführen
- Abhilfe: Die Erstellung eines eigenen Java-Clients (mit der Jersey API, REST assured, …)

Exkurs: Java Community Process (~~JCP~~) EFSP

- Die Sprache Java wird durch den Java Community Process weiterentwickelt.

- Mitglieder des JCP (Firmen und Einzelpersonen) können Vorschläge zur Änderung der Sprache Java einbringen → Java Specification Request (JSR)

- Wird ein JSR angenommen, erstellt eine Expertengruppe eine Referenzimplementierung

- Für JAX-RS (Java API for RESTful Web Services) ist dies **Jersey**.

- Weitere Implementierungen sind zB RESTEasy, CXF, Restlet,…

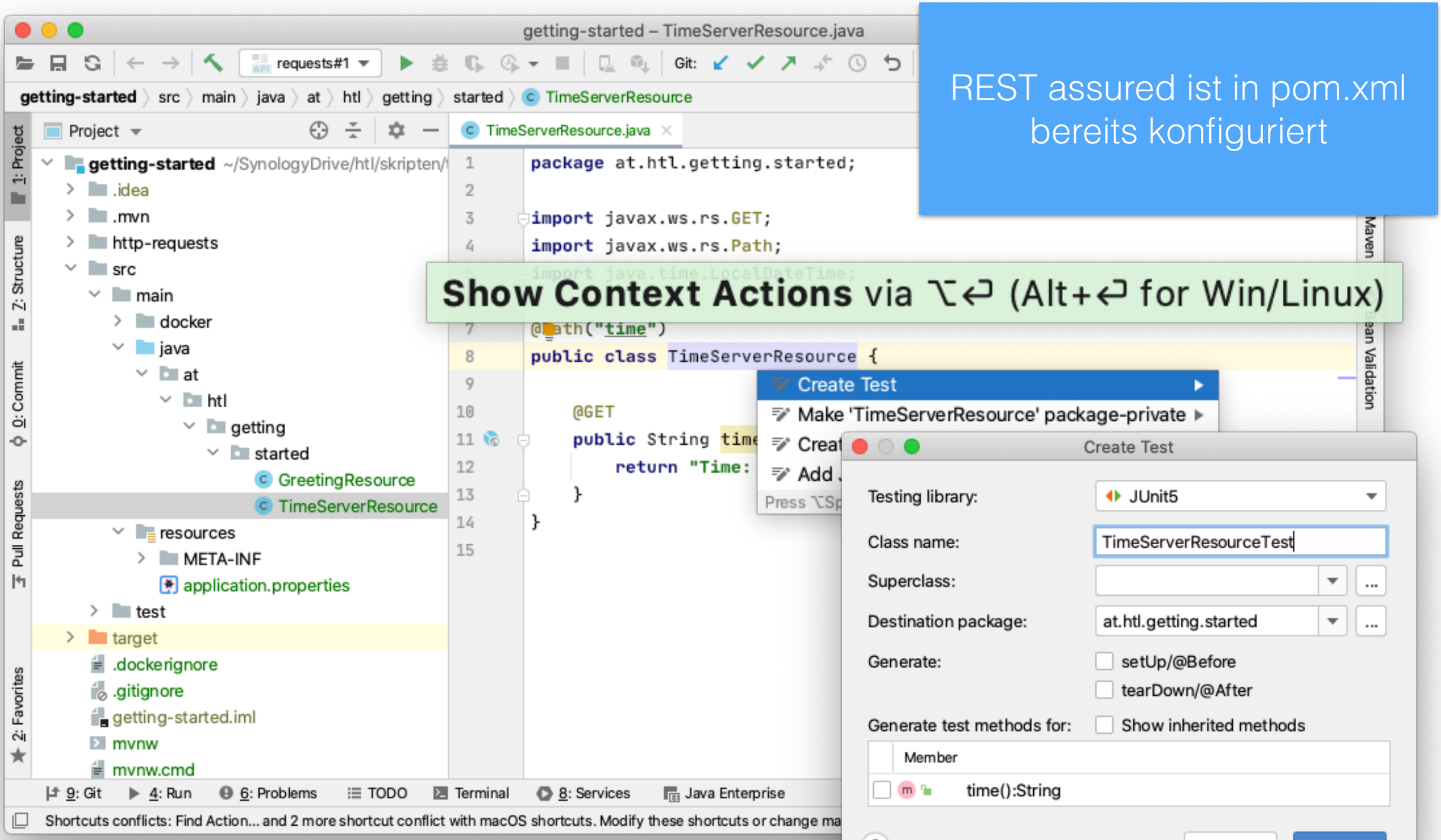
https://de.wikipedia.org/wiki/Java_Community_Process

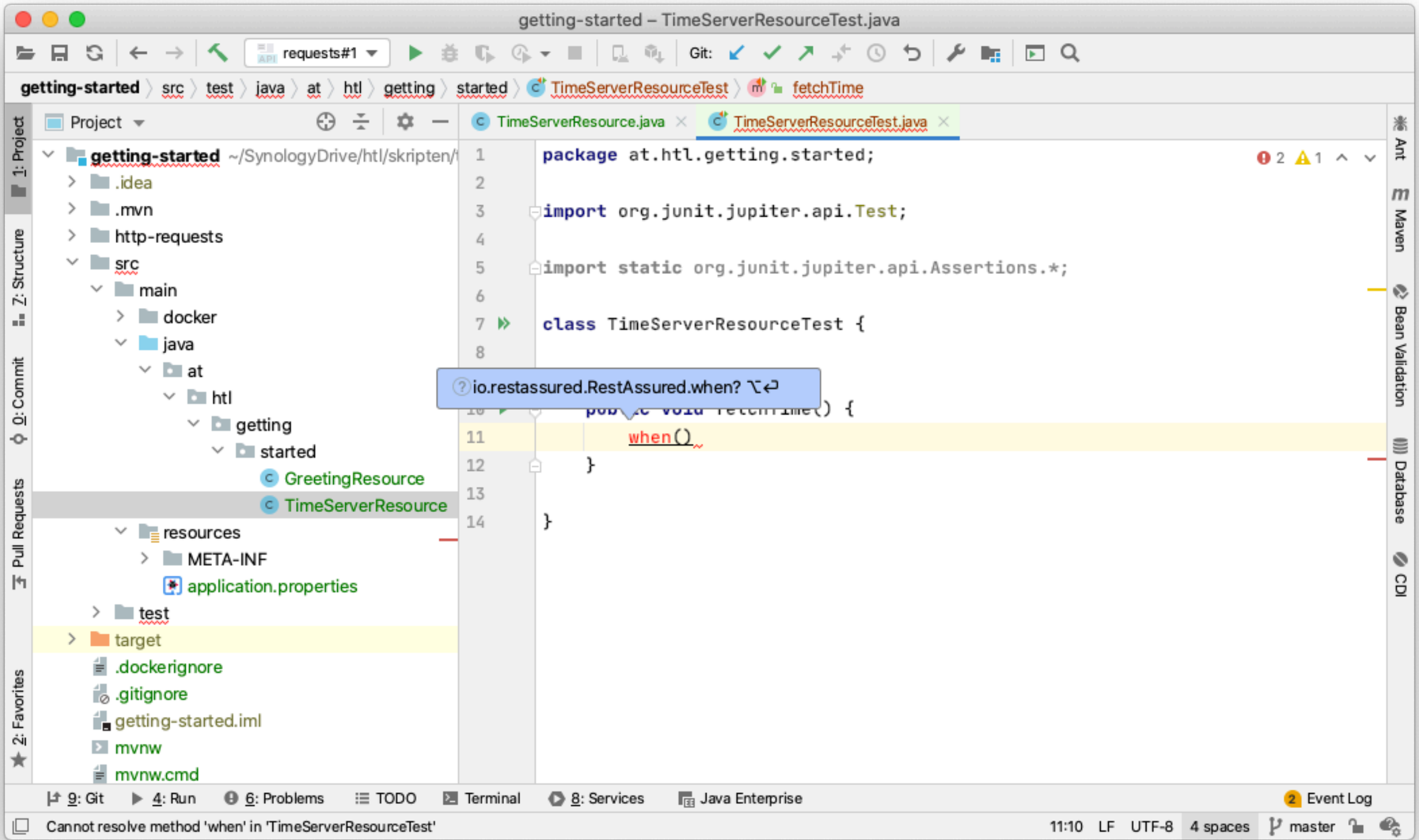
Eclipse Foundation Specification Process

- EFSP
- https://www.eclipse.org/community/eclipse_newsletter/2019/january/EFSP_vs_JCP.php
- https://jaxenter.de/ee-insights-jakarta-ee-8-spezifikationsprozess-82216
- https://jaxenter.de/5-unterschiede-jcp-efsp-78514

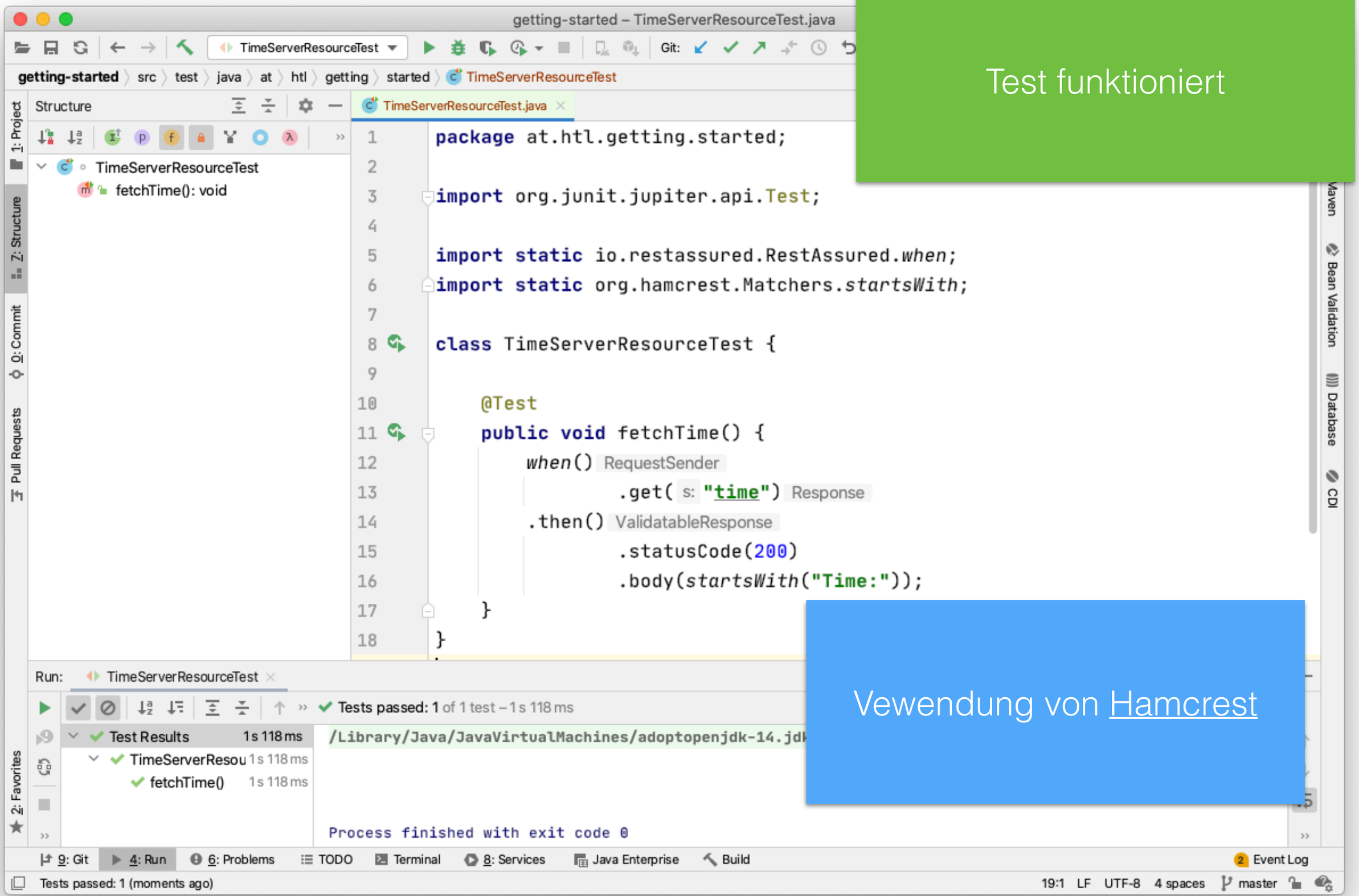
REST assured - Testclient

REST assured ist in pom.xml bereits konfiguriert



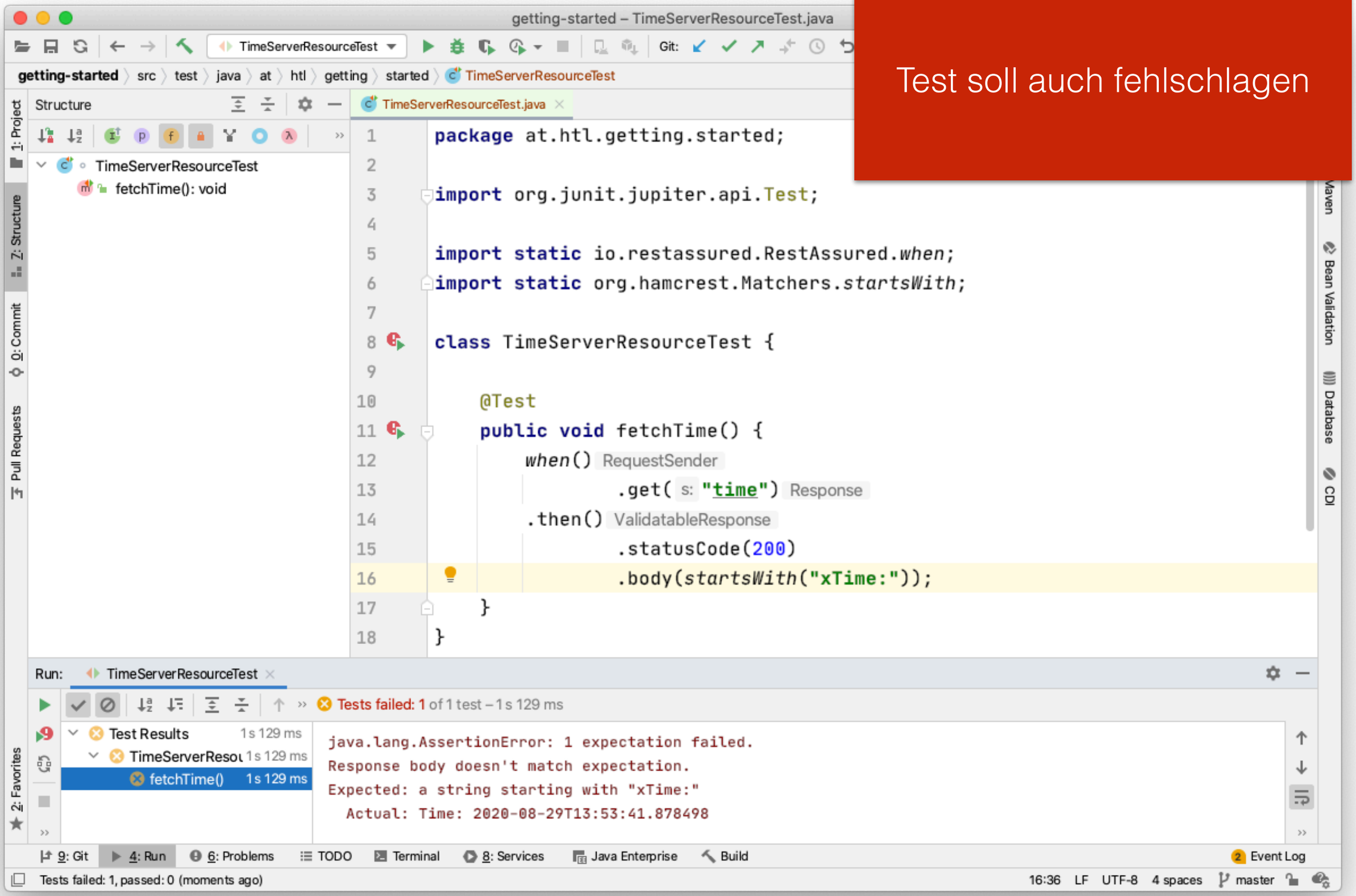

getting-started – TimeServerResourceTest.java
requests#1
Git:
getting-started src test java at htl getting started TimeServerResourceTest fetchTime
Project
getting-started ~/SynologyDrive/htl/skripten/t
.idea
.mvn
http-requests
src
main
docker
java
at
htl
getting
started
GreetingResource
TimeServerResource
resources
META-INF
application.properties
test
target
.dockerignore
.gitignore
getting-started.iml
mvnw
mvnw.cmd
TimeServerResource.java
TimeServerResourceTest.java
package at.htl.getting.started;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class TimeServerResourceTest {
io.restassured.RestAssured.when? ⌥⏎
when()
}
2 1
1: Project
Z: Structure
0: Commit
Pull Requests
2: Favorites
Ant
Maven
Bean Validation
Database
CDI
9: Git
4: Run
6: Problems
TODO
Terminal
8: Services
Java Enterprise
Event Log
Cannot resolve method 'when' in 'TimeServerResourceTest'
11:10
LF
UTF-8
4 spaces
master

Test funktioniert



Vewendung von Hamcrest

Test soll auch fehlschlagen



Lösung

```
package at.htl.restprimer.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@Path("time")
public class TimeServerResource {

    @GET
    public String time() {
        return "Time: " + LocalDateTime
                .now()
                .format(DateTimeFormatter.ofPattern("dd. MMMM yyyy, hh:mm:ss"));
    }
}
```