Mocking-Alternativen: EasyMock, WireMock & more

Michael Inden

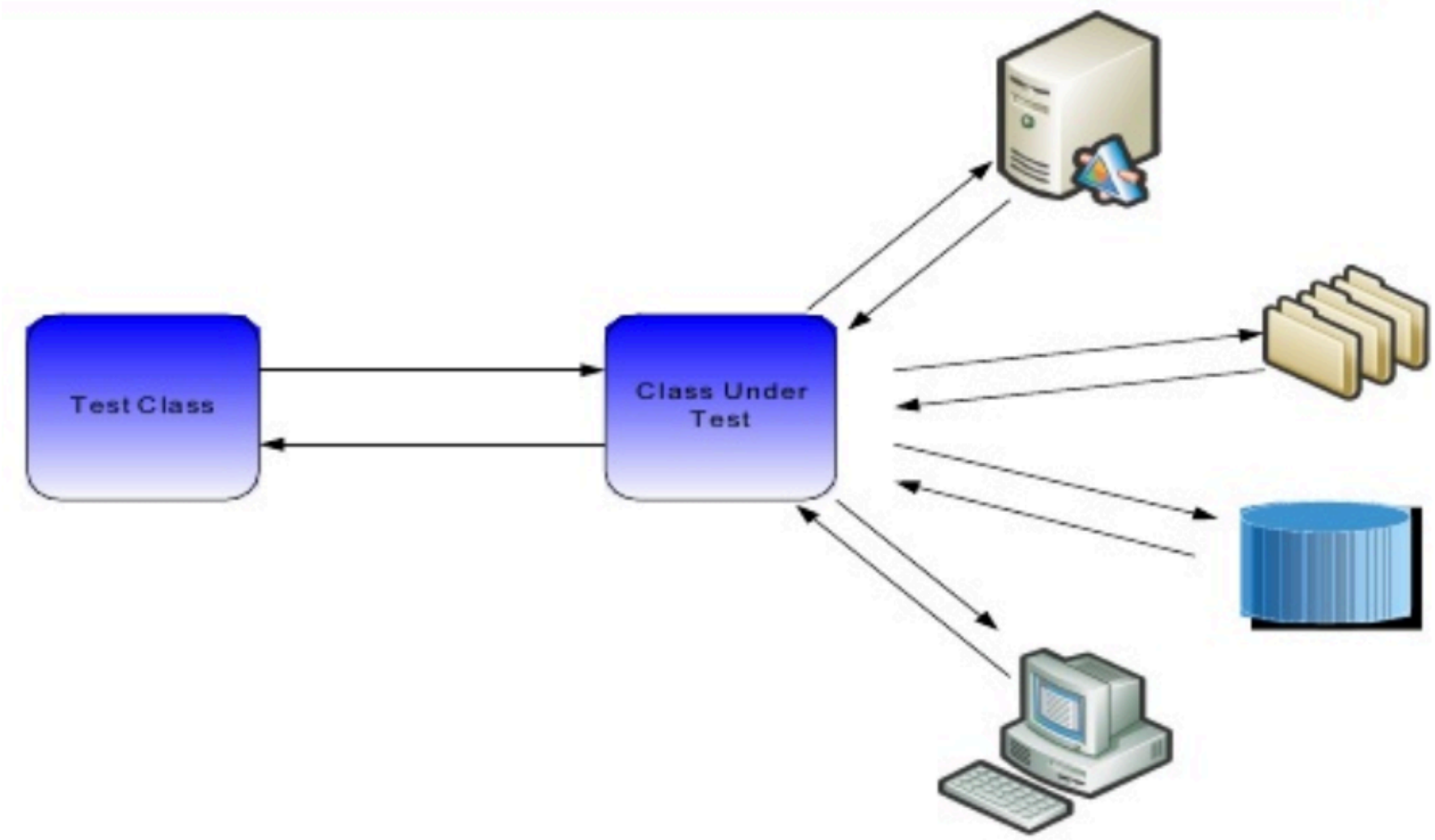
Freiberuflicher Consultant, Buchautor und Trainer

Coding Guidelines – Agenda

- Rekapitulation: Motivation für Mocking
- Alternativen zu Mockito am Beispiel von EasyMock
- WireMock im Überblick
- RESTAssured für lesbare Tests
- WireMock in Kombination mit JUnit
- MockServer als Alternative zu WireMock

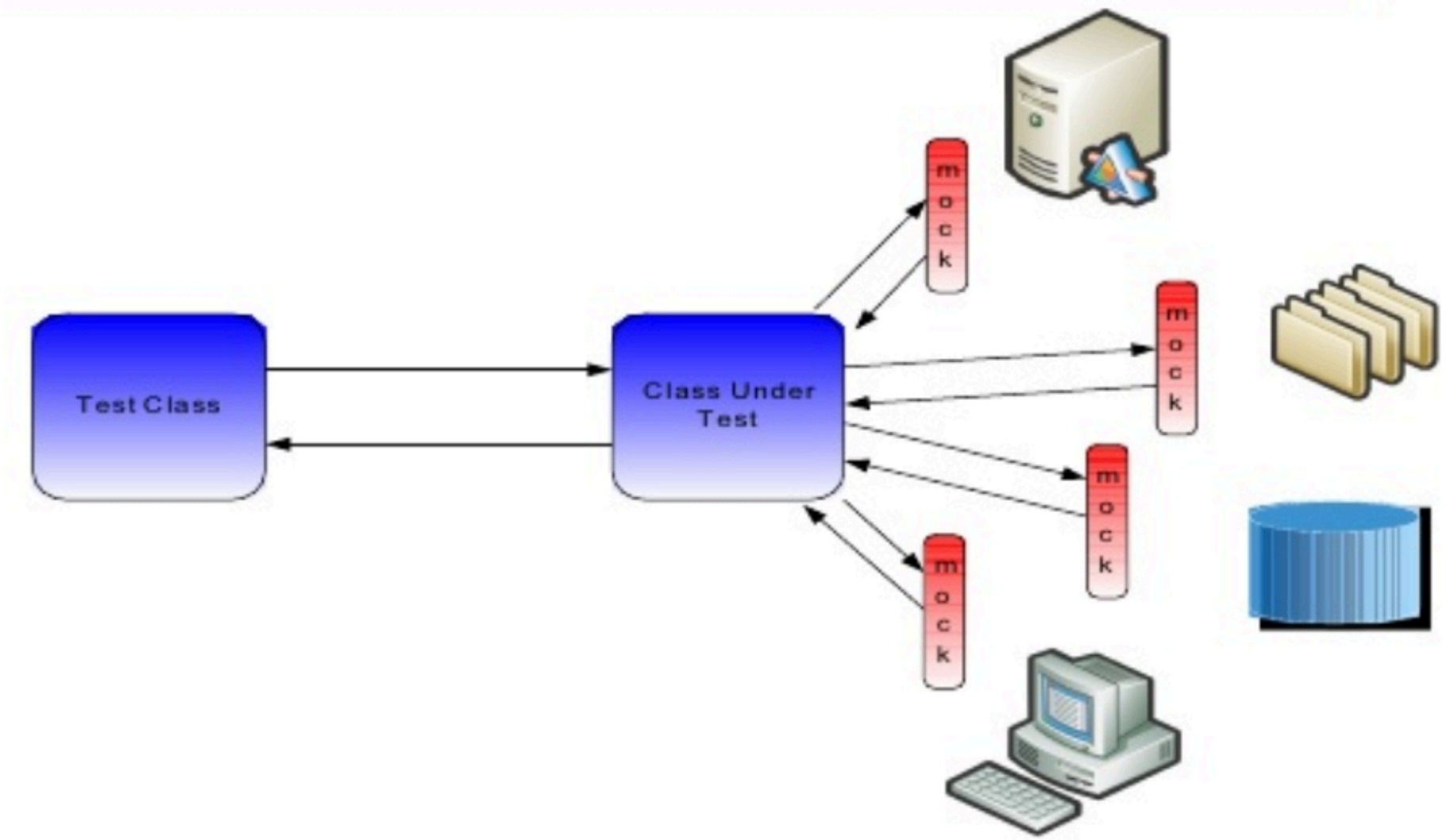
Motivation für Mocking

Zu testende Klasse mit vielen Abhängigkeiten



https://www.slideshare.net/Gemsbond/easymock-for-java

Abhängigkeiten mit Mocks im Griff



https://www.slideshare.net/Gemsbond/easymock-for-java

Alternativen zu Mockito

Alternativen zu Mockito

Google Trends | Vergleichen

Mockito	EasyMock	JMock	JMockIt	
Suchbegriff	Suchbegriff	Suchbegriff	Suchbegriff	+

Weltweit | Letzte 5 Jahre | Alle Kategorien | Websuche

Interesse im zeitlichen Verlauf

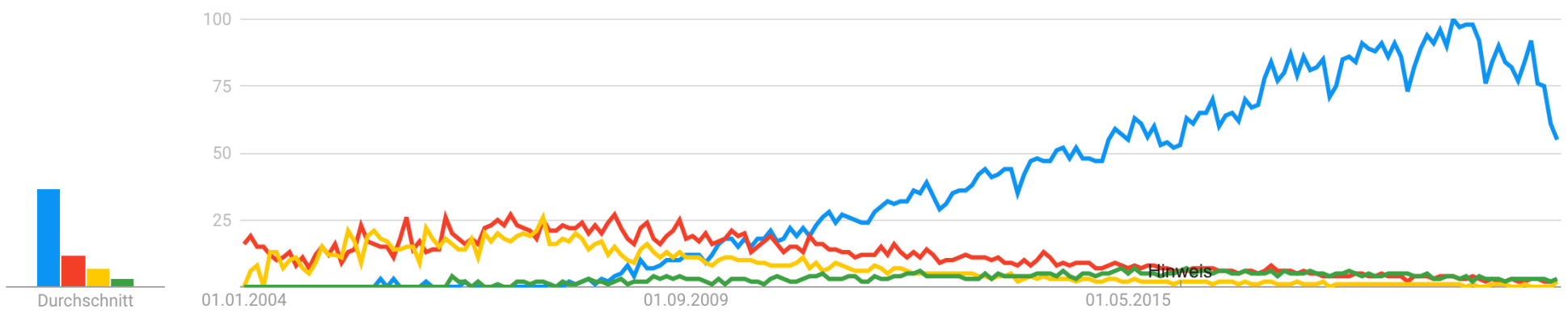
Alternativen zu Mockito

Google Trends | Vergleichen

Mockito	EasyMock	JMock	JMockIt	
Suchbegriff	Suchbegriff	Suchbegriff	Suchbegriff	+

Weltweit | 2004 - heute | Alle Kategorien | Websuche

Interesse im zeitlichen Verlauf



Easy Mock im Kurzüberblick

EasyMock als Alternative zu Mockito

```
<!-- https://mvnrepository.com/artifact/org.easymock/easymock -->
<dependency>
    <groupId>org.easymock</groupId>
    <artifactId>easymock</artifactId>
    <version>4.2</version>
    <scope>test</scope>
</dependency>
```

EasyMock als Alternative zu Mockito

```
public class IncomeCalculator
{
    private final SalaryService salaryService;

    public IncomeCalculator(final SalaryService salaryService)
    {
        this.salaryService = salaryService;
    }

    public int getIncomePerMonth(final Position position)
    {
        return salaryService.salaryForPosition(position) / 12;
    }
}
```

Wie schreiben wir einen Test ohne auf den externen Salary Service angewiesen zu sein?

EasyMock als Alternative zu Mockito

```
public class IncomeCalculator
{
    private final SalaryService salaryService;

    public IncomeCalculator(final SalaryService salaryService)
    {
        this.salaryService = salaryService;
    }

    public int getIncomePerMonth(final Position position)
    {
        return salaryService.salaryForPosition(position) / 12;
    }
}
```

EasyMock als Alternative zu Mockito

```
@Test
public void testWithNiceMock()
{
    // ARRANGE
    SalaryService salaryService = createNiceMock(SalaryService.class);
    IncomeCalculator incomeCalculator = new IncomeCalculator(salaryService);

    expect(salaryService.salaryForPosition(Position.ARCHITECT)).andReturn(100000).times(2);
    expect(salaryService.salaryForPosition(Position.PROGRAMMER)).andReturn(60000);
    replay(salaryService);

    // ACT
    int result1 = incomeCalculator.getIncomePerMonth(Position.ARCHITECT);
    int result2 = incomeCalculator.getIncomePerMonth(Position.ARCHITECT);
    int result3 = incomeCalculator.getIncomePerMonth(Position.PROGRAMMER);
    int result4 = incomeCalculator.getIncomePerMonth(Position.LEAD_DEVELOPER); // => 0
    String desc = salaryService.getJobDescription(); // => null

    // ASSERT AND VERIFY
    assertEquals(8333, result1);
    assertEquals(8333, result2);
    assertEquals(5000, result3);
    // CHECK IF ALL EXPECTED CALLS REALLY OCCURED
    verify(salaryService);
}
```

EasyMock als Alternative zu Mockito

```
@Test
public void testWithPlainMock()
{
    // ARRANGE
    SalaryService salaryService = createMock(SalaryService.class);
    IncomeCalculator incomeCalculator = new IncomeCalculator(salaryService);

    expect(salaryService.salaryForPosition(Position.ARCHITECT)).andReturn(100000).times(2);
    expect(salaryService.salaryForPosition(Position.PROGRAMMER)).andReturn(60000);
    replay(salaryService);

    // ACT
    int result1 = incomeCalculator.getIncomePerMonth(Position.ARCHITECT);
    int result2 = incomeCalculator.getIncomePerMonth(Position.ARCHITECT);
    int result3 = incomeCalculator.getIncomePerMonth(Position.PROGRAMMER);
    // Unexpected method call SalaryService.salaryForPosition(LEAD_DEVELOPER):
    // int result4 = incomeCalculator.getIncomePerMonth(Position.LEAD_DEVELOPER);

    // ASSERT AND VERIFY
    assertEquals(8333, result1);
    assertEquals(8333, result2);
    assertEquals(5000, result3);

    // CHECK IF ALL EXPECTED CALLS REALLY OCCURED
    verify(salaryService);
}
```

D E M O

Bislang haben wir Komponenten gemocked, die über den Sourcecode angebunden waren. Was machen wir mit REST-APIs?

WireMock im Überblick

Ein REST / Web Service Test Double für alle Gelegenheiten

Herausforderungen

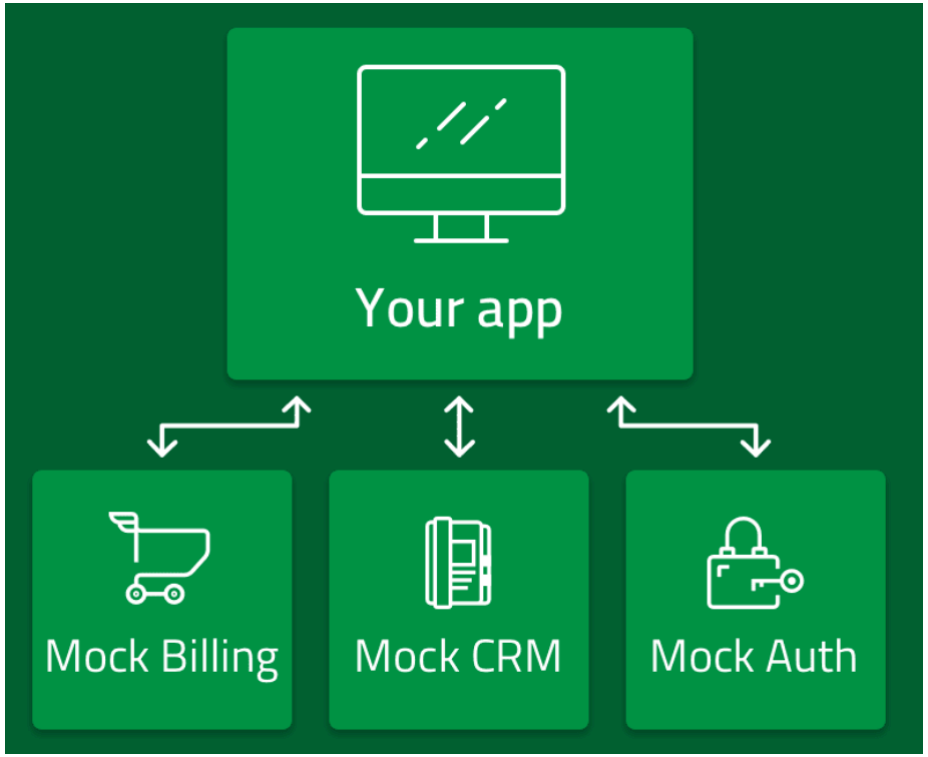
Beim Testen im Zusammenspiel erwarten uns folgende Herausforderungen:

- Tests schlagen aufgrund von Verbindungsproblemen fehl
- Tests schlagen aufgrund von Konfigurationseinstellungen fehl
- Tests können durch Kommunikation mit externem System ziemlich lange dauern
- Gewisse Seiten haben ein API rate limit (z.B. Twitter)
- Das benötigte API existiert noch nicht oder zumindest nicht vollständig
- Das benötigte API bietet keinen Testserver

Aufgrund dieser Punkte sollten unsere Tests ohne Zugriff auf externe Systeme lauffähig sein. Hier kommt WireMock ins Spiel.

WireMock Intro

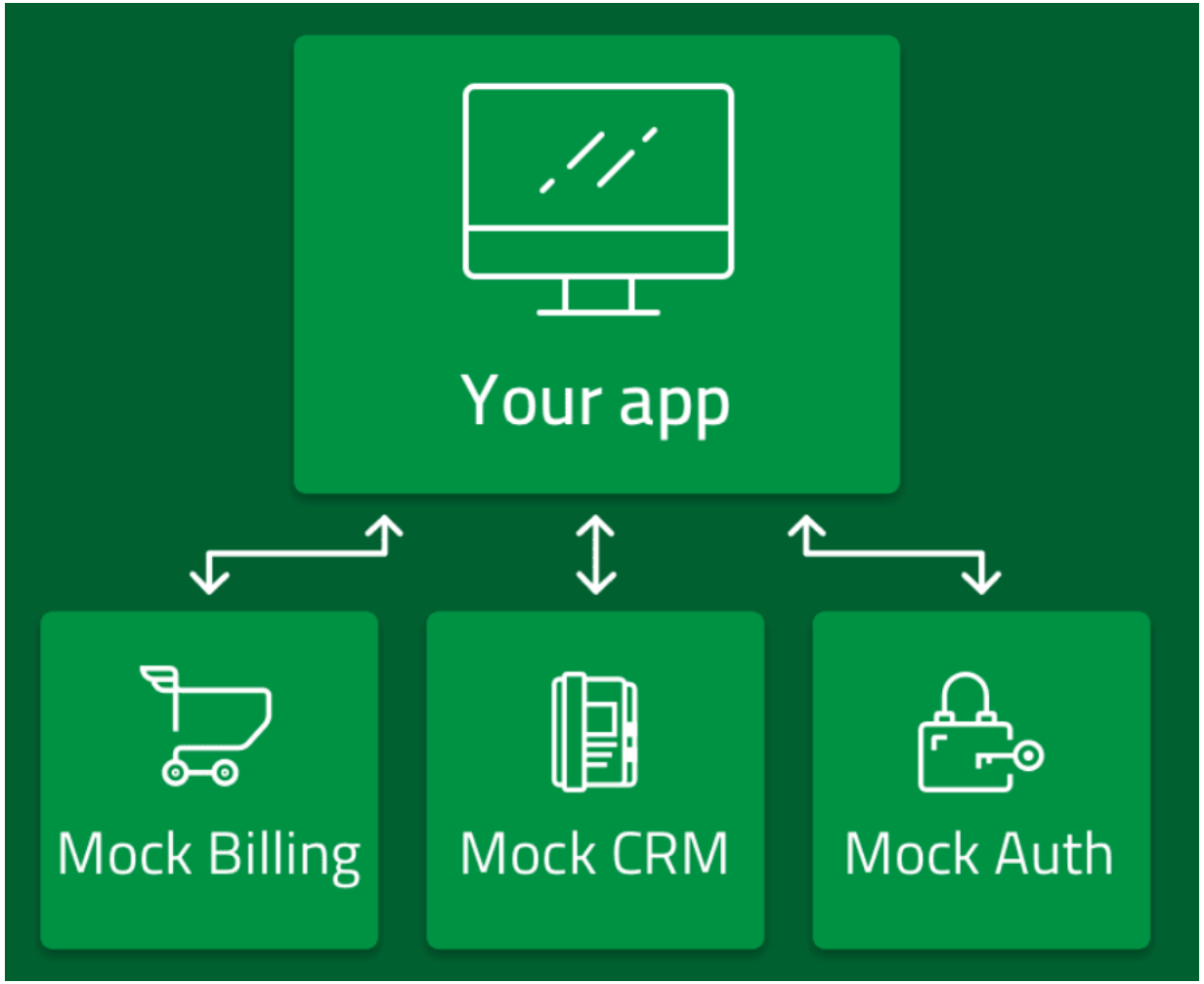
- WireMock als Tool, um REST APIs zu simulieren (a.k.a. Mock Server)
- Lässt sich in Unit Tests oder Standalone verwenden
- WireMock erlaubt es, die Entwicklung voranzutreiben, obwohl ein eingesetztes REST-API derzeit noch nicht (vollständig) zur Verfügung steht



WireMock Intro

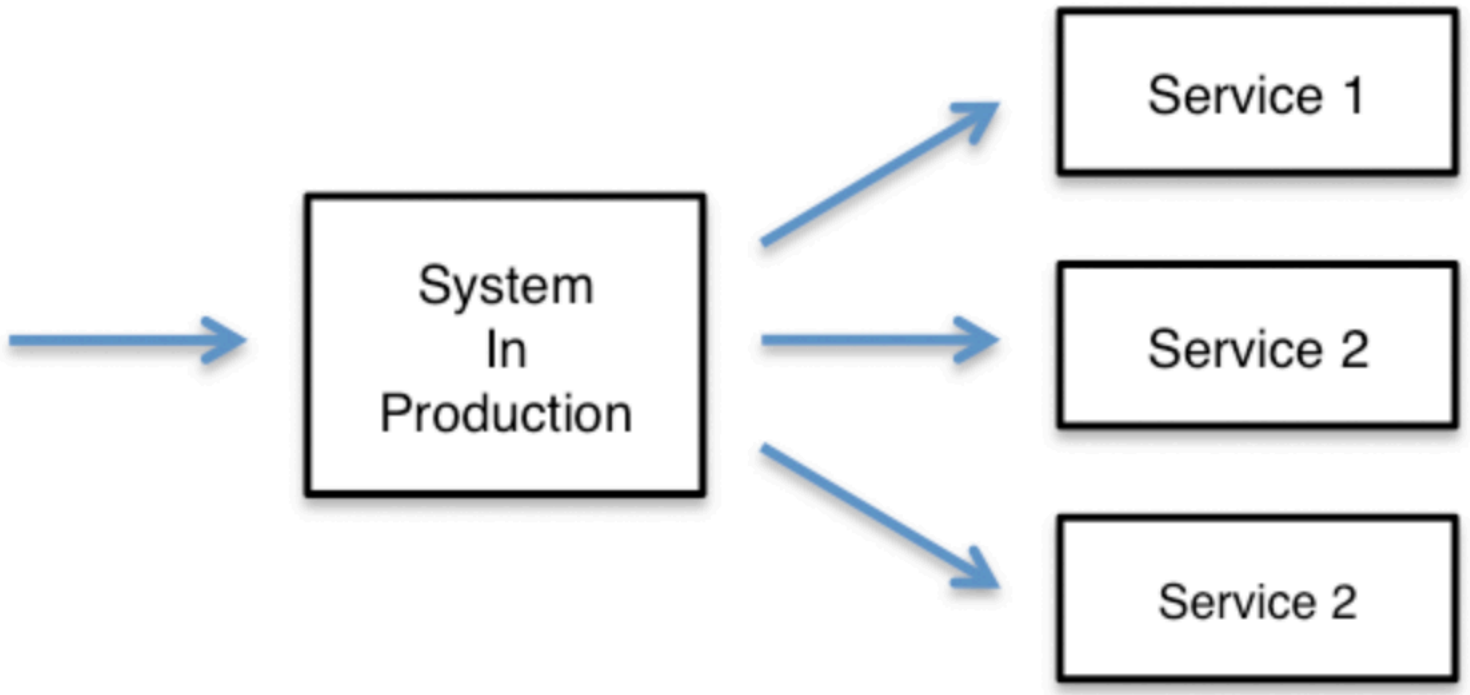
- WireMock ist ein WebServer, der vordefinierte Antworten auf eingehende Requests liefert und die Request zur späteren Verifikation aufzeichnet
- Gute Konfigurierbarkeit mit Fluent API, JSON usw.
- Es lassen sich Fehlersituationen provozieren oder Verzögerungen simulieren, was im Normalbetrieb kaum möglich ist
- Proxying möglich
- Simulation von zustandsabhängigen Verhalten möglich

WireMock Motivation und Vorteile vom HTTP Mocking

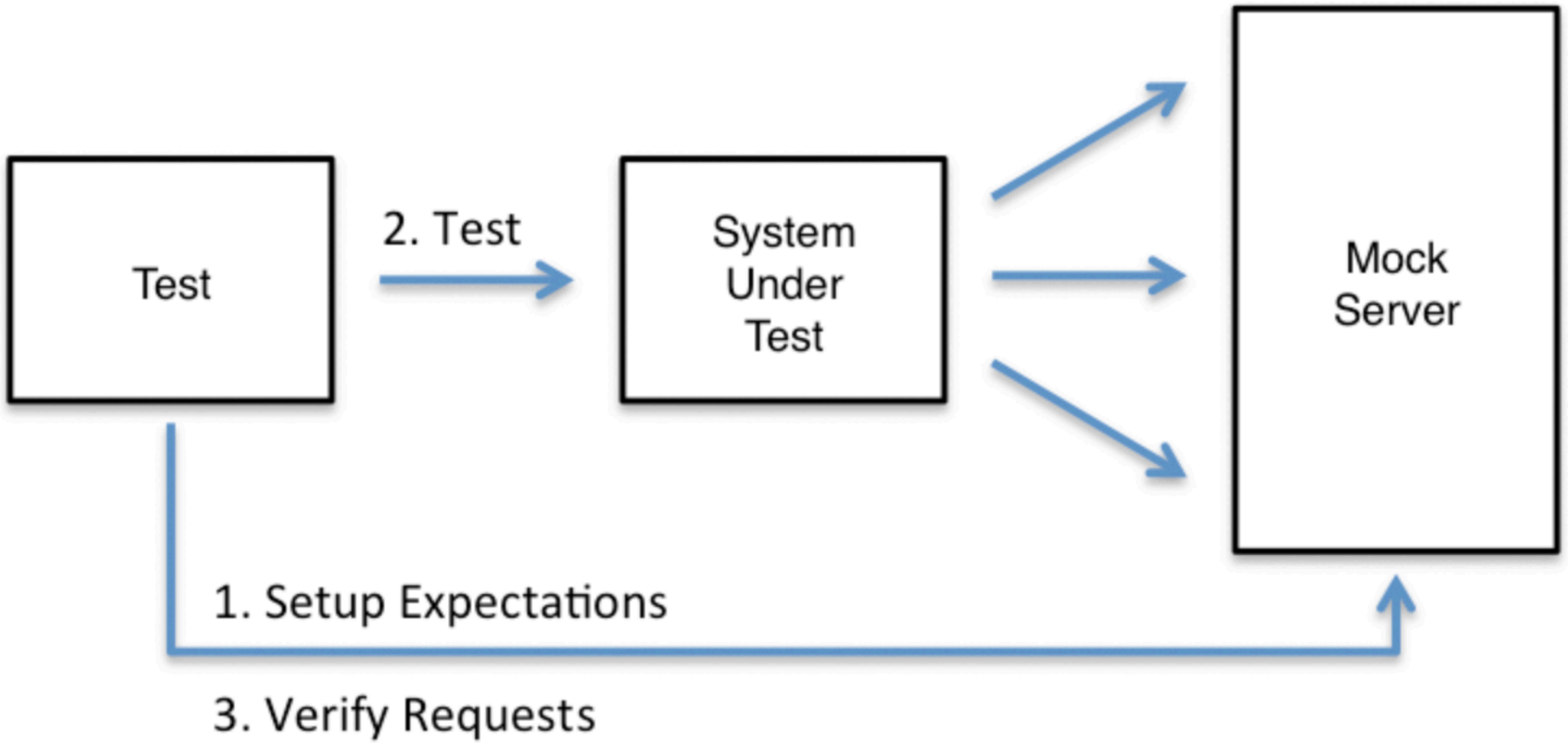


- Unabhängigkeit von anderen Komponenten
- Geschwindigkeit
- Keine Gefahr, versehentlich Modifikationen auszuführen
- Tests von Randfällen, Fehlersituationen usw.
- Reproduzierbarkeit

WireMock Erste Schritte

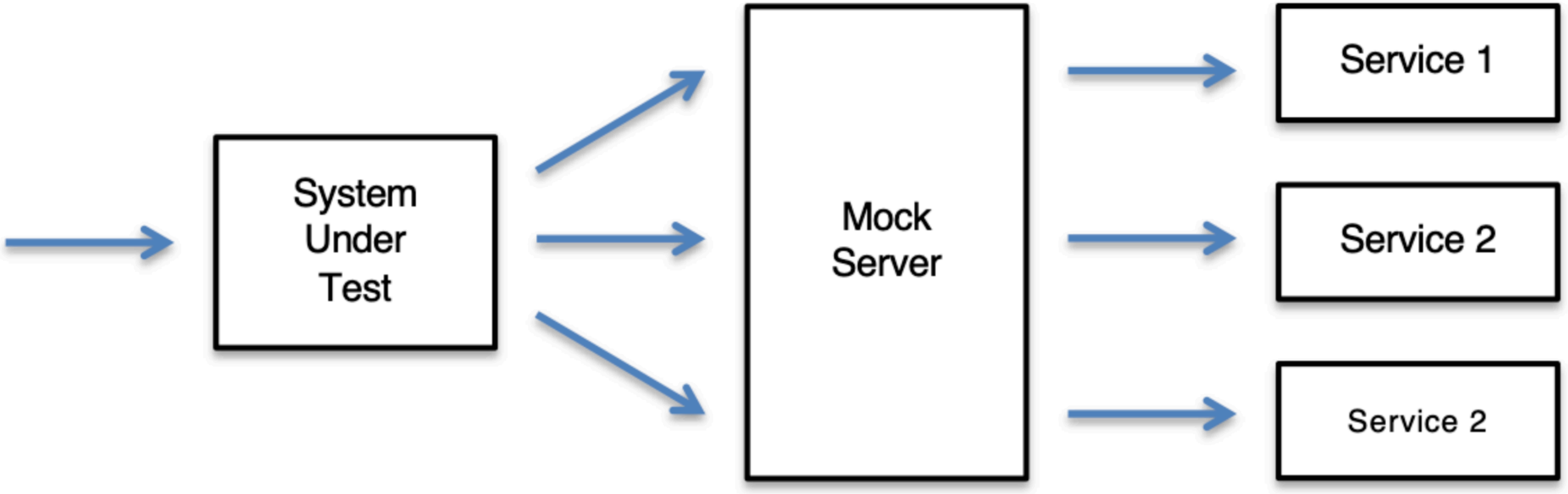


WireMock Erste Schritte: Ersatz für externe Services



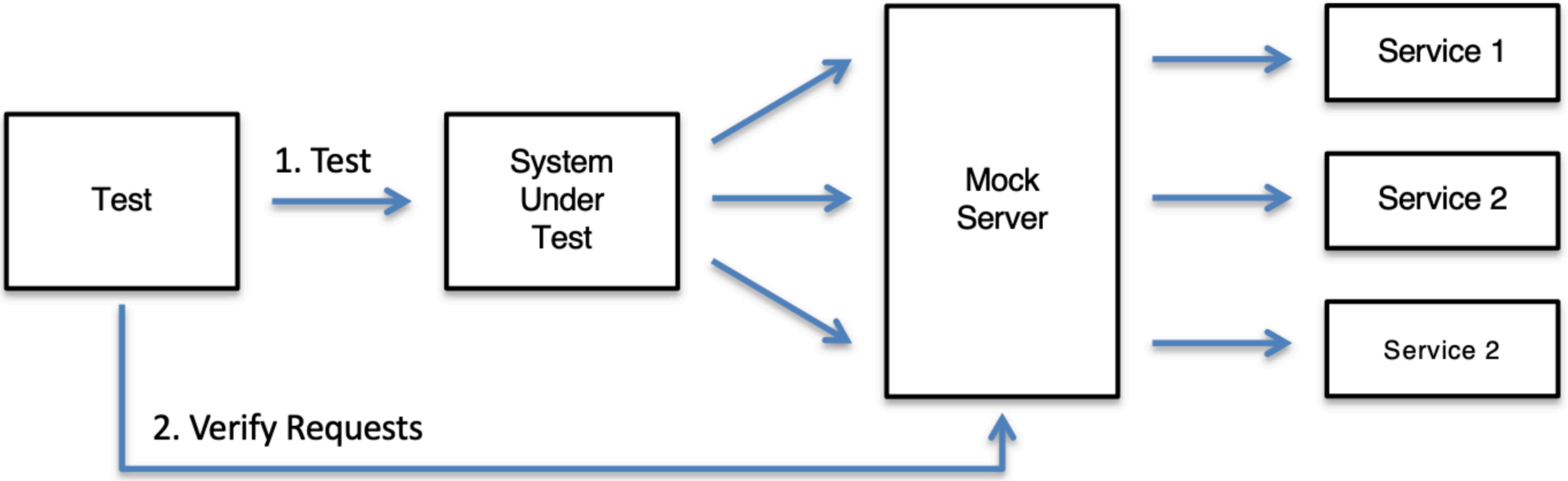
https://www.mock-server.com/

WireMock Erste Schritte: Als Proxy für externe Services



https://www.mock-server.com/

WireMock Erste Schritte: Als Proxy für externe Services



Einschub REST Assured und Demo-Webserver «Req Res» & «Ergast» & «http://api.zippopotam.us»

REST Assured

- REST Assured als DSL für lesbare REST-API-Tests
- Folgt dem AAA / GWT-Stil (BDD-Stil) mit Given / When / Then
- Gute lesbar und verständlich
- Fluent API, dadurch in einem Rutsch die gesamte Verarbeitung spezifizierbar

```
@Test
public void makeSureThatGoogleIsUp()
{
    given().
    when().get("http://www.google.com").
    then().statusCode(200);
}
```

REST Assured Maven Dependency

```xml
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.2</version>
    <scope>test</scope>
</dependency>


<!-- Ab Java 9 -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured-all</artifactId>
    <version>4.3.2</version>
    <scope>test</scope>
</dependency>
```

https://reqres.in/

REQ | RES

Test your front-end against a real API

Fake data

No more tedious sample data creation, we've got it covered.

Real responses

Develop with real response codes. GET, POST, PUT & DELETE supported.

Always-on

24/7 *free* access in your development phases. Go nuts.

A hosted REST-API ready to respond to your AJAX requests.

Beispiel für POST und DELETE

```
@Test
public void test_post()
{
    JSONObject request = new JSONObject();
    request.put("name", "Michael");
    request.put("job", "Trainer");

    given().body(request.toJSONString()).
    when().post("https://reqres.in/api/users").
    then().statusCode(201);
}

@Test
public void test_delete()
{
    given().
    when().delete("https://reqres.in/api/users/5").
    then().statusCode(204);
}
```

http://ergast.com/mrd/

API Documentation

The Ergast Developer API is an experimental web service which provides a historical record of motor racing data for non-commercial purposes. Please read the terms and conditions of use. The API provides data for the Formula One series, from the beginning of the world championships in 1950.

Non-programmers can query the database using the manual interface or download the database tables in CSV format for import into spreadsheets or analysis software.

If you have any comments or suggestions please post them on the Feedback page. If you find any bugs or errors in the data please report them on the Bug Reports page. Any enhancements to the API will be reported on the News page. Example applications are shown in the Application Gallery.

Overview

All API queries require a GET request using a URL of the form:

```
http[s]://ergast.com/api/<series>/<season>/<round>/...
```

where:

`<series>`	should be set to "f1"
`<season>`	is a 4 digit integer
`<round>`	is a 1 or 2 digit integer

Index

- API Documentation
  - Season List
  - Race Schedule
  - Race Results
  - Qualifying Results
  - Standings
  - Driver Information
  - Constructor Information
  - Circuit Information
  - Finishing Status
  - Lap Times
  - Pit Stops
- Query Database
- Database Images
- Terms & Conditions
- Application Gallery
- Feedback
- FAQ
- Latest News
- Bug Reports

Links

- Contact Us
- Programmable Web

Meta

- Log in

Lesbare Abfragen ohne Parametrierung

```
@Test
public void test_NumberOfCircuitsFor2019Season_ShouldBe21() {

    given().
    when().
        get("http://ergast.com/api/f1/2019/circuits.json").
    then().
        assertThat().
        body("MRData.CircuitTable.Circuits.circuitId", hasSize(21));
}
```

Lesbare Abfragen mit Parametrierung und als ParameterizedTest

```
@ParameterizedTest
@CsvSource({"2017,20", "2019, 21"})
public void test_NumberOfCircuits_Parameterized(String season, int numberOfRaces)
{
    given().
        pathParam("raceSeason", season).
    when().
        get("http://ergast.com/api/f1/{raceSeason}/circuits.json").
    then().
        assertThat().
        body("MRData.CircuitTable.Circuits.circuitId",hasSize(numberOfRaces));
}
```

http://api.zippopotam.us

Zippopotam.us Samples Countries Who Code Uptime

New Management The Zippopotam.us project is under new management. View the Github Repository for details. ×

Zip Code Galore!

Zip·po·pot·amus /ˈzipōpätəməs/

Postal Codes and Zip Codes made easy

- Free API with JSON Response Format
- Over 60 Countries Supported
- Perfect for Form Autocompletion
- Open for Crowdsourcing and Contribution

Try It Out »

Structure: api.zippopotam.us/country/postal-code
Example: api.zippopotam.us/us/90210
NEW! City->Zip: api.zippopotam.us/country/state/city
Example: api.zippopotam.us/us/ma/belmont

Autocomplete Examples: USA Germany Spain France

Lesbare Abfragen mit Verknüpfung und Zugriff auf Body

```
import static io.restassured.RestAssured.*;
import static io.restassured.matcher.RestAssuredMatchers.*;
import static org.hamcrest.Matchers.*;


@Test
public void checkContentTypeAndLog()
{
    given().
        pathParam("country","ch").
        pathParam("zipcode","8047").
    when().
        get("http://api.zippopotam.us/{country}/{zipcode}").
    then().
        assertThat().statusCode(200).
        body("places.'place name'[0]", equalTo("Zürich")).
     and().
        contentType(ContentType.JSON).
        log().all();
}
```

Ergebnis der ZIP-Abfrage


```
{
    "post code": "8047",
    "country": "Switzerland",
    "country abbreviation": "CH",
    "places": [
        {
            "place name": "Zürich",
            "longitude": "8.487",
            "state": "Kanton Zürich",
            "state abbreviation": "ZH",
            "latitude": "47.3739"
        }
    ]
}
```

D E M O

WireMock in Kombination mit JUnit 4/5

WireMock Maven Dependency

```
<dependency>
    <groupId>com.github.tomakehurst</groupId>
    <artifactId>wiremock-jre8</artifactId>
    <version>2.27.2</version>
    <scope>test</scope>
</dependency>
```

WireMock Konfiguration eines Endpunkts

```
public void setupStub()
{
    stubFor(get(urlEqualTo("/an/endpoint"))
            .willReturn(aResponse()
                .withHeader("Content-Type", "text/plain")
                .withStatus(200)
                .withBody("You've reached a valid WireMock endpoint")));
}
```

WireMock Prüfen mit JUnit

```
@Rule
public WireMockRule wireMockRule = new WireMockRule(8089);
// No-args constructor defaults to port 8080 (we use this for DOCKER)

@Test
public void testAnEndpoint_ShouldReturn_StatusCode_200()
{
    setupStub();

    when().get("http://localhost:8089/an/endpoint").
    then().assertThat().statusCode(200);
}
```

WireMock Prüfen mit JUnit

```
@Test
public void createUserWithPost() throws Exception
{
    stubFor(post(urlEqualTo("/users")).willReturn(aResponse().withStatus(201)
            .withHeader("content-type", "text/plain").withBody("User created!")));

    given().
        body("<user>USER-MICHAEL</user>").
    when().
        post("http://localhost:8089/users").
    then().
        assertThat().
        statusCode(201).
    and().
        assertThat().body(equalTo("User created!"));

    verify(postRequestedFor(urlMatching("/users")).
           withRequestBody(matching(".*USER.*")).
           withHeader("Content-Type", notMatching("application/json")));
}
```

Weitere Möglichkeiten der Definition von Endpunkten (Stubs)

```
stubFor(delete("/fine").willReturn(ok()));

stubFor(get("/fine-with-body").willReturn(ok("body content")));

stubFor(get("/json").willReturn(okJson("{ \"message\": \"Hello\" }")));

stubFor(post("/redirect").willReturn(temporaryRedirect("/new/location")));

stubFor(post("/sorry-no").willReturn(unauthorized()));

stubFor(put("/status-only").willReturn(status(418)));
```

Prof of Concept
HTTP/2-API statt REST Assured

WireMock Prüfen mit JUnit und HTTP/2-API

```
@Test
public void testAnEndpoint_ShouldReturn_StatusCode_200_Http2_API() throws Exception
{
    setupStub();

    HttpResponse<String> result = performGetCall("/an/endpoint", "");

    assertEquals(200, result.statusCode());
}
```

HTTP/2-API aus Java 11 zum Request senden (statt REST Assured)

```
static HttpResponse<String> performGetCall(String path, String contentType)
                            throws IOException, InterruptedException
{
    if (contentType == null || contentType.isEmpty())
        contentType = "text/plain";

    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder()
                          .uri(URI.create("http://localhost:8089" + path))
                          .header("Content-Type", contentType)
                          .build();

    return client.send(request, BodyHandlers.ofString());
}
```

WireMock Prüfen mit JUnit und HTTP/2-API

```
@Test
public void createUserWithPost_Http2_API() throws Exception
{
    stubFor(post(urlEqualTo("/users")).willReturn(aResponse().withStatus(201)
            .withHeader("content-type", "text/plain").withBody("User created!")));

    HttpResponse<String> result = performPostWithXmlBody("/users", "USER-MICHAEL");

    assertEquals(201, result.statusCode());
    assertEquals("User created!", result.body());

    verify(postRequestedFor(urlMatching("/users")).
           withRequestBody(matching(".*USER.*")).
           withHeader("Content-Type", notMatching("application/json")));
}
```

HTTP/2-API aus Java 11 für POST-Request (statt REST Assured)

```
static HttpResponse<String> performPostWithXmlBody(String path, String content)
                            throws IOException, InterruptedException
{
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder()
                          .uri(URI.create("http://localhost:8089" + path))
                          .POST(HttpRequest.BodyPublishers.ofString(content))
                          .header("Content-Type", "text/xml")
                          .header("Accept", "text/xml")
                          .build();

    return client.send(request, BodyHandlers.ofString());
}
```

Zustandsbehaftete Kommunikation mit Szenarios

Stub mit Szenario erstellen ... Teil I

```
public void setupStatefulStub()
{
    stubFor(get(urlEqualTo("/shoppingcart"))
        .inScenario("shopping")
        .whenScenarioStateIs(Scenario.STARTED)
        .willReturn(aResponse().withHeader("Content-Type", "application/xml")
                               .withBody("<list>Empty</list>")
                               .withStatus(200)));

    stubFor(post(urlEqualTo("/shoppingcart"))
        .inScenario("shopping")
        .whenScenarioStateIs(Scenario.STARTED)
        .willSetStateTo("itemAdded-1")
        .willReturn(aResponse().withHeader("Content-Type", "application/xml")
                               .withStatus(201)));

    stubFor(get(urlEqualTo("/shoppingcart"))
        .inScenario("shopping")
        .whenScenarioStateIs("itemAdded-1")
        .willReturn(aResponse().withHeader("Content-Type", "application/xml")
                               .withBody("<list><item>Item 1</item></list>")
                               .withStatus(200)));
```

SZENARIO

ZUSTAND: START

ZUSTAND: START

FOLGEZUSTAND

Stub mit Szenario erstellen ... Teil II

```
    stubFor(post(urlEqualTo("/shoppingcart"))
        .inScenario("shopping")
        .whenScenarioStateIs("itemAdded-1")
        .willSetStateTo("itemAdded-2")
        .willReturn(aResponse()
                    .withHeader("Content-Type", "application/xml")
                    .withStatus(201)));

    stubFor(get(urlEqualTo("/shoppingcart"))
        .inScenario("shopping")
        .whenScenarioStateIs("itemAdded-2")
        .willReturn(aResponse()
                    .withStatus(200)
                    .withHeader("Content-Type", "application/xml")
                    .withBody("<list><item>Item 1</item><item>Item 2</item></list>")));
}
```

Test-Schritt 1 & 2 & 3

```
@Test
public void testStatefulMock()
{
    setupStatefulStub();

    given().
    when().
       get("http://localhost:8089/shoppingcart").
    then().
        assertThat().statusCode(200).
        and().
        assertThat().body("list", org.hamcrest.Matchers.equalTo("Empty"));

    given().
    when().
        post("http://localhost:8089/shoppingcart").
    then().
        assertThat().statusCode(201);

    given().
    when().
        get("http://localhost:8089/shoppingcart").
    then().
        assertThat().statusCode(200).
        and().
        assertThat().body("list.item", org.hamcrest.Matchers.equalTo("Item 1"));
```

1. GET /shoppingcart
   => „leere Liste“

2. POST /shoppingcart
   => „Element hinzufügen“

3. GET /shoppingcart
   => „Liste mit 1 Element“

Test-Schritt 4 & 5

```
    given().
    when().
        post("http://localhost:8089/shoppingcart").
    then().
        assertThat().
        statusCode(201);


    given().
    when().
        get("http://localhost:8089/shoppingcart").
    then().
        assertThat().
        statusCode(200).
        and().
        assertThat().
        body(org.hamcrest.Matchers.hasXPath("count(//item)",
             org.hamcrest.Matchers.equalTo("2"))).
        assertThat().
        body("list.item[0]", org.hamcrest.Matchers.equalTo("Item 1")).
        assertThat().
        body("list.item[1]", org.hamcrest.Matchers.equalTo("Item 2"));
}
```

4. `POST /shoppingcart`
   => „Element hinzufügen“

5. `GET /shoppingcart`
   => „Liste mit 2 Elementen“

Variante mit XPATH

Explizite Prüfungen

D E M O

MockServer als Alternative zu WireMock im Kurzüberblick

MockServer Maven Dependency

```
<!-- mockserver -->
<dependency>
      <groupId>org.mock-server</groupId>
      <artifactId>mockserver-netty</artifactId>
      <version>5.11.1</version>
</dependency>


<dependency>
    <groupId>org.mock-server</groupId>
    <artifactId>mockserver-junit-jupiter</artifactId>
    <version>5.11.1</version>
</dependency>
```

MockServer Beispiel

```
@ExtendWith(MockServerExtension.class)
public class MockServerIntroTest
{
    MockServerClient client = new MockServerClient("127.0.0.1", 1080);

    private void setupStub()
    {
        client.when(request()
                    .withMethod("GET")
                    .withPath("/an/endpoint"))
              .respond(response()
                      .withStatusCode(200)
                      .withBody("You've reached a valid MockServer endpoint"));
    }

    @Test
    public void testStatusCodePositive()
    {
        setupStub();

        when().get("http://localhost:8090/an/endpoint").then().assertThat().statusCode(200);
    }
}
```

Interesse an den beiden Tools

Google Trends | Vergleichen

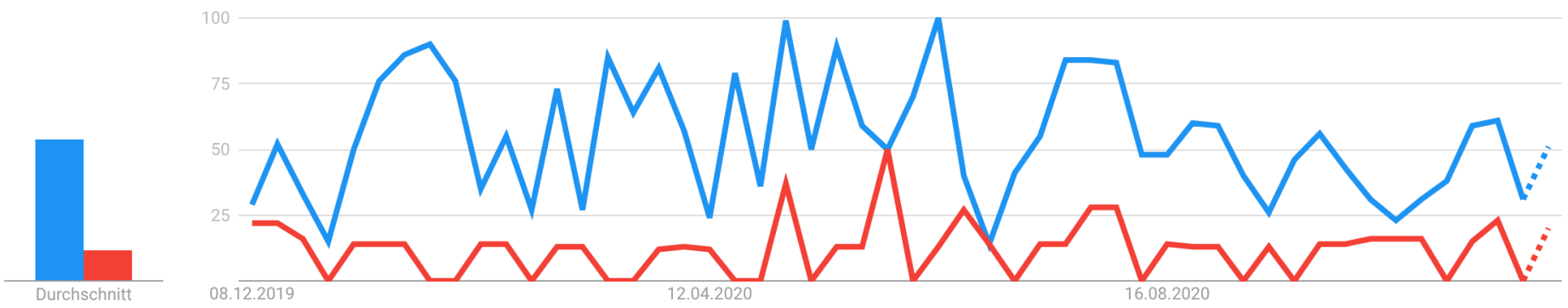
WireMock
Suchbegriff

MockServer
Suchbegriff

+ Vergleich hinzufügen

Vereinigte Staaten | letzte 12 Monate | Alle Kategorien | Websuche

Interesse im zeitlichen Verlauf



Tools in Docker

WireMock / MockServer und Docker

WireMock

- `docker pull rodolpheche/wiremock`
- `docker run -it --rm -p 8080:8080 rodolpheche/wiremock`

MockServer

- `docker pull mockserver/mockserver`
- `docker run -d -p 1080:1080 -p 1090:1090 jamesdbloom/mockserver`

```
Michaels-MBP-2:AAA_WireMock_EasyMock michaeli$ docker ps
CONTAINER ID        IMAGE                     COMMAND                  ... PORTS                              NAMES
8993c9564b9b        mockserver/mockserver     "java -Dfile.encodin…"   ... 0.0.0.0:32768->1080/tcp            xenodochial_cerf
1e98e4b202c5        rodolpheche/wiremock      "/docker-entrypoint.…"   ... 0.0.0.0:8080->8080/tcp, 8443/tcp   wizardly_haslett
```

WireMock / MockServer und Docker Konfiguration

wiremock_upload_curl.sh

```
curl -X POST --data '@unauthorized_invoices.json' "http://localhost:8080/__admin/mappings/new"
curl -X POST --data '@authorize.json' "http://localhost:8080/__admin/mappings/new"
curl -X POST --data '@authorized_invoices.json' http://localhost:8080/__admin/mappings/new

...
```

unauthorized_invoices.json

```
{
  "scenarioName": "Test scenario",
  "requiredScenarioState": "Started",
  "request": {
    "method": "GET",
    "url": "/invoices"
  },
  "response": {
    "status": 403
  }
}
```

localhost:8080/__admin/

Meistbesucht | Erste Schritte | Apple | iCloud | Yahoo | Bing | Google | Wikipe

JSON | Rohdaten | Kopfzeilen

Speichern | Kopieren | Alle einklappen | Alle ausklappen | JSON durchsuchen


```
mappings:
  0:
    id:                     "8cda4e47-9cf2-4f20-9042-13d1e3262fd8"
    request:
      url:                  "/invoices"
      method:               "GET"
    response:
      status:               200
      body:                 "[ first: Hello world! / second: Very expensive! ]"
    uuid:                   "8cda4e47-9cf2-4f20-9042-13d1e3262fd8"
    scenarioName:           "Test scenario"
    requiredScenarioState:  "Authorized"
  1:
    id:                     "5ddd9076-8907-4961-8e45-7248fff6b28e"
    request:
      url:                  "/auth"
      method:               "GET"
    response:
      status:               200
    uuid:                   "5ddd9076-8907-4961-8e45-7248fff6b28e"
    scenarioName:           "Test scenario"
    newScenarioState:       "Authorized"
```

WireMock vs Mockito

Benefits..

WireMock

- Web server just like real api
- Real http
- External to app code
- Can simulate network faults
- Language agnostic
- Prod like testing

Mocking(Mockito)

- No web server
- No http
- It's in application code
- Can't
- Language specific mocking libs
- No

https://www.youtube.com/watch?v=xJs-KuEL-co

Übungen

Infos / Links

- http://wiremock.org/docs/
- http://wiremock.org/docs/getting-started/
- http://www.ontestautomation.com/getting-started-with-wiremock/
- https://blog.softwaremill.com/stateful-tests-in-wiremock-fca68f855264
- https://www.youtube.com/watch?v=c_egn9-PZQ4
- https://www.youtube.com/watch?v=x3MvZ8DFrpE

Infos / Links

- https://rest-assured.io/
- https://techbeacon.com/app-dev-testing/how-perform-api-testing-rest-assured
- https://techbeacon.com/app-dev-testing/efficient-api-testing-how-get-started-rest-assured
- https://www.mock-server.com/mock_server/getting_started.html

Thank You