MATHEMA

# API-Definition mal anders

## Schnittstellen aus Sicht des Clients definieren

Werner Eberling (@wer_eb)
werner.eberling@mathema.de

www.mathema.de


https://docs.PACT.io

# Der Sprecher


## Werner Eberling

Principal Consultant / Autor

Email: werner.eberling@mathema.de

Twitter: @Wer_Eb



# OMG was ist den das für eine Schnittstelle?

https://pixabay.com/de/vectors/junge-geschäft-cartoon-comic-1300226/

# Eine kurze Geschichte zum Thema

https://pixabay.com/de/illustrations/erzählung-geschichte-traum-erzählen-794978/

# Wie wäre es den Client zu fragen?



https://pixabay.com/de/photos/arbeitsplatz-team-geschäftstreffen-1245776/

# Das war aber anders abgesprochen!!!

https://pixabay.com/de/vectors/junge-geschäft-cartoon-comic-1300226/

# Lasst uns einen Vertrag machen




https://pixabay.com/de/photos/vereinbarung-gesch%C3%A4ft-vertrag-3476369/

# Design by Contract - Das Ende der Flexibilität?



https://pixabay.com/de/photos/igromania-spielsucht-handschellen-1894847/

# Postels Gesetz

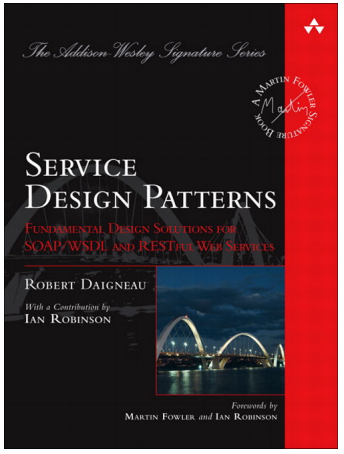
„sei streng bei dem was du tust und
offen bei dem was du von anderen akzeptierst“

– RFC 761

# Clients als Tolerant Reader

- Der Client definiert was er macht und braucht…

© Pixabay Licence




https://martinfowler.com/bliki/TolerantReader.html

- … und ignoriert alles andere.

© Pixabay Licence

# Wie war das jetzt mit dem Vertrag?




https://pixabay.com/de/photos/vereinbarung-gesch%C3%A4ft-vertrag-3476369/

# Testen – und zwar alles!!!

Logisch, oder nicht?

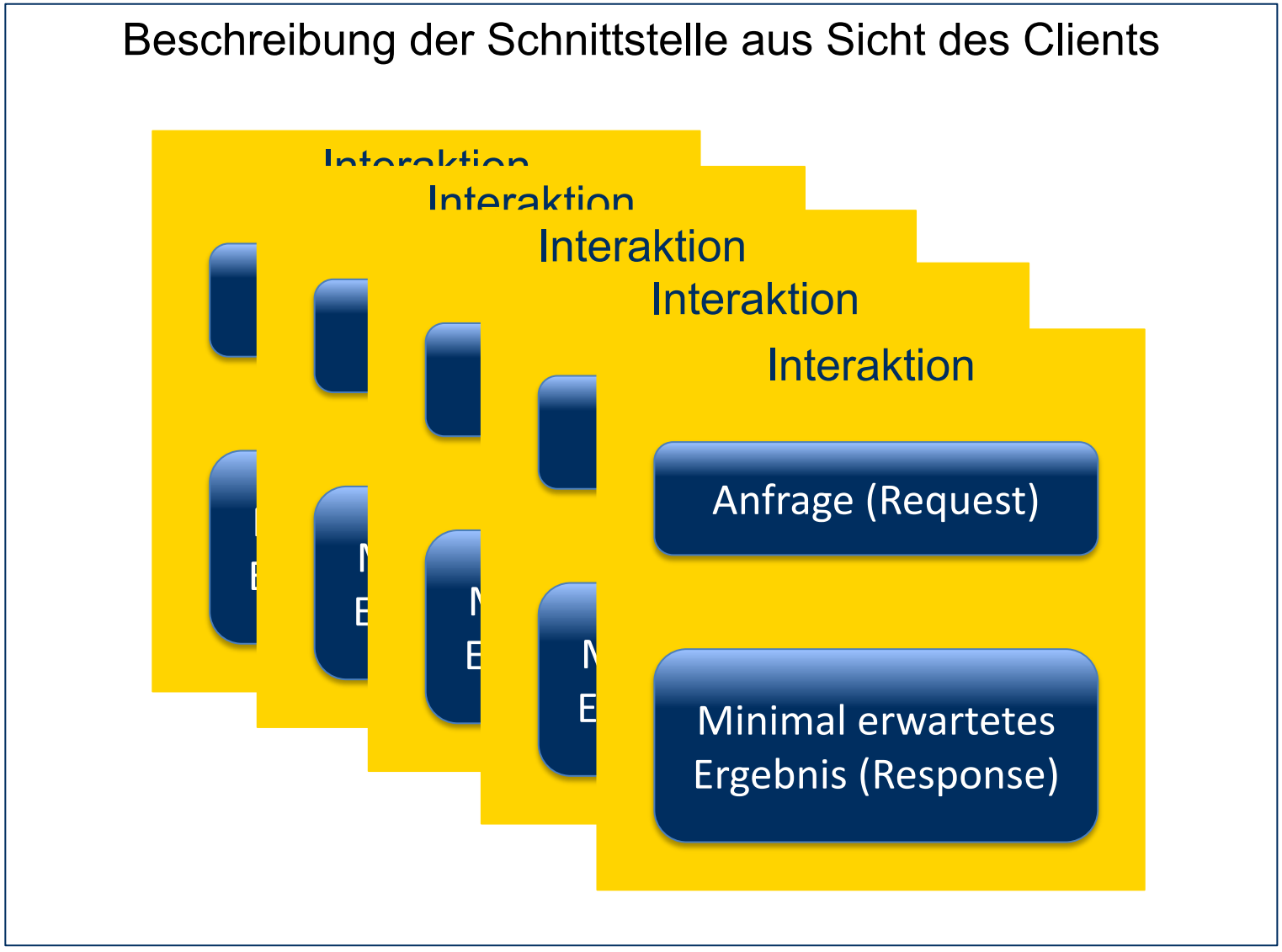
# Moment, wie ist das im echten Leben?

# Gleich das ganze Haus anzünden?

https://pixabay.com/de/photos/haus-abgebrannt-verkohlt-schwarz-262609/

# Wohl eher den Signaleingang simulieren…

# Interaktionen statt starrer Schnittstellen



# Contract Testing


https://docs.PACT.io

vs.

Spring Cloud Contract

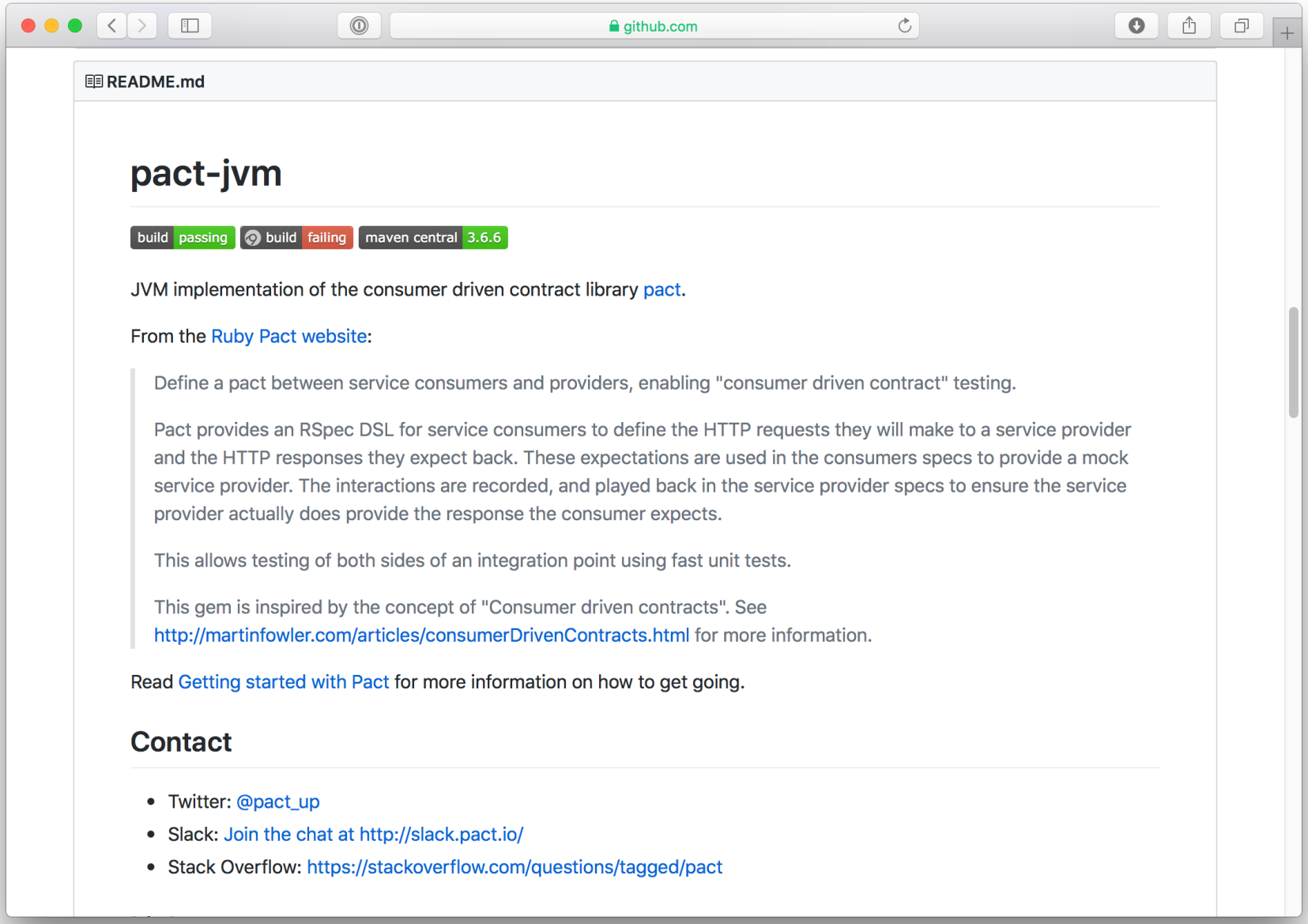
https://cloud.spring.io/spring-cloud-contract

# Contract Testing mit PACT

PACT (noun):
A formal agreement between individuals or parties.
"The country negotiated a trade PACT with the US"
Synonyms: agreement, protocol, deal, contract
~ Oxford Dictionaries
(von: https://docs.PACT.io)


https://docs.PACT.io

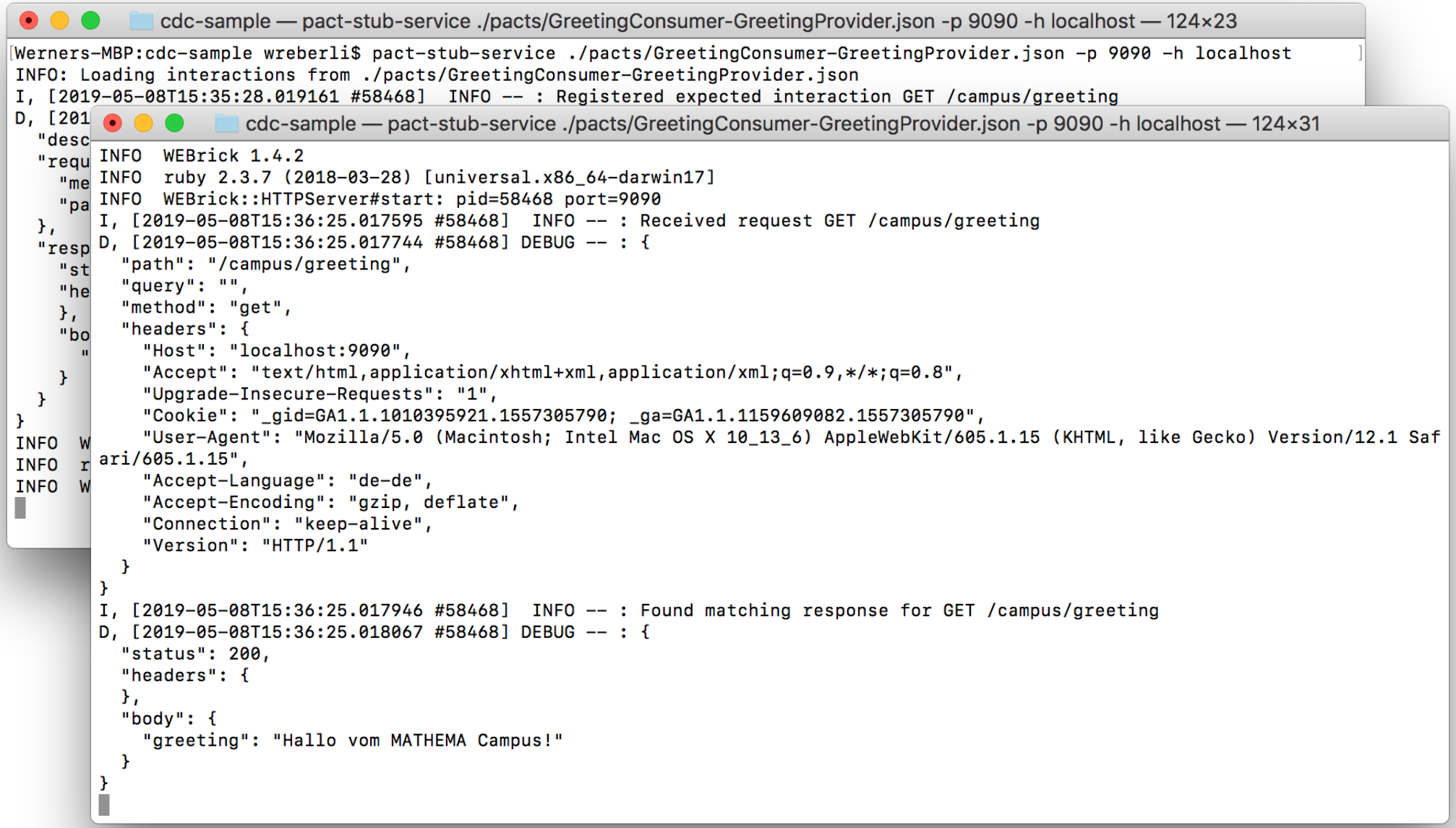
# Contract Testing mit PACT – PACT-jvm



# Wie sieht so ein Vertrag aus?


```
{
  "provider": {
    "name": "HelloWorldProvider"
  },
  "consumer": {
    "name": "HelloWorldConsumer"
  },
  "interactions": [
    {
      "description": "Abfrage von HelloWorld",
      "request": {
        "method": "GET",
        "path": "/"
      },
      "response": {
        "status": 200,
        "body": {
            "response": "Hello World"
        }
      }
    }
  ],
  "metadata": {
    "PACTSpecification": {
      "version": "3.0.0"
    },
    "PACT-jvm": {
      "version": "3.6.6"
    }
  }
}
```

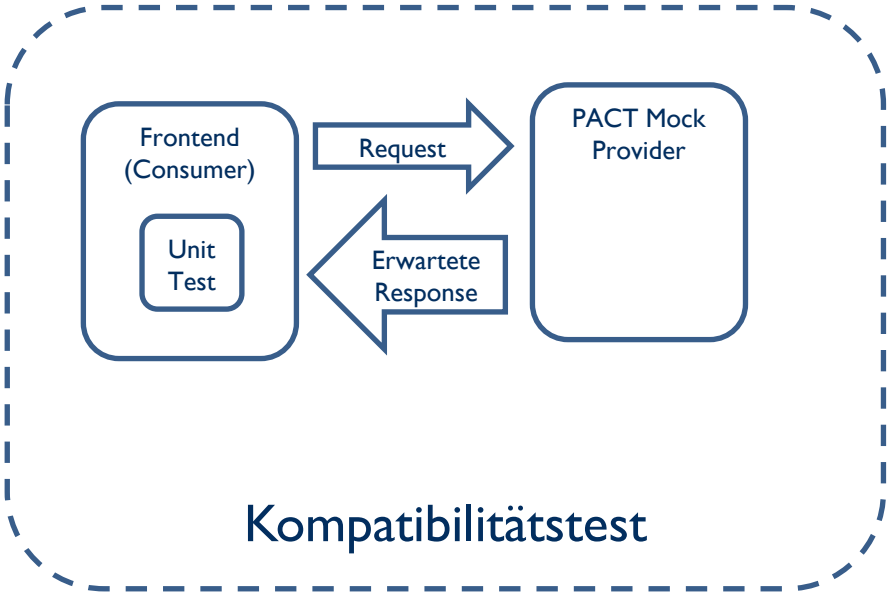
# Bonusfeature: PACTs als Mockup für Integrationstests



# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# Consumer prüft Kompatibilität gegen den PACT



# PACT Erzeugung via UnitTest

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
	return builder
		.uponReceiving("Abfrage aller Gruesse mit vorhandenen Daten")
			.path("/greetings")
			.method("GET")
		.willRespondWith()
			.status(200)
			.body(new PACTDslJsonBody()
				.array("greetings")
				.object()
					.stringValue("type", "formal")
					.stringValue("phrase", "Good morning")
					.closeObject()
				.object()
					.stringValue("type", "casual")
					.stringValue("phrase", "Hey")
					.closeObject()
			)
		.toPACT();
}
```

# Consumer prüft seine eigene Kompatibilität im selben Test

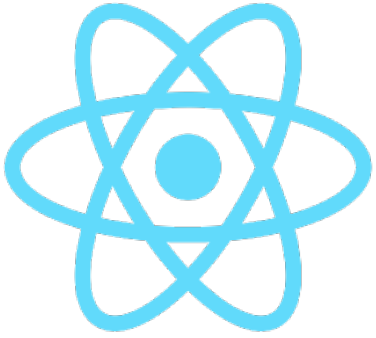
```
@ExtendWith(PACTConsumerTestExt.class)
@PACTTestFor(providerName = "GreetingProvider")
@PACTFolder("PACTs")
public class GreetingConsumerTest
{
    @PACT(provider="GreetingProvider", consumer="GreetingConsumer")
    public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
        return builder
             .uponReceiving("Abfrage aller Gruesse ohne vorhandene Daten")
                   .path("/greetings")
                   .method("GET")
             .willRespondWith()
                   .status(200)
                   .body(new PACTDslJsonBody().array("greetings"))    }

    @Test
    void testAbfrageAllerGruesse(MockServer mockServer) throws Exception {
      HttpResponse resp = Request.Get(mockServer.getUrl()+"/greetings").execute().returnResponse();
      Assertions.assertEquals(200, resp.getStatusLine().getStatusCode());
}
```

**Im “echten“ Projekt immer auch die “echten“ Clientklassen verwenden, anstatt mit dem HttpClient zu testen!**

# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# Aber: UIs entstehen heutzutage in JavaScript oder TypeScript




# PACT bauen und Verifizieren im Karma Test

```
describe('create()', () => {
    beforeAll((done) => {
      provider.addInteraction({
        uponReceiving: 'Abfrage aller Gruesse ohne vorhandene Daten‘,
        withRequest: {
          method: 'GET‘,
          path: '/greetings'        },
        willRespondWith: {
          status: 200,
          body: {
            greetings: []
          }
        }
      }).then(done, error => done.fail(error));
    });

    it('should return no Greetings', (done) => {
      const userService: GreetingService = TestBed.get(GreetingService);
      userService.loadGreetings().subscribe(response => {
        expect(response).toEqual({greetings: []});
        done();
      }, error => {
        done.fail(error);
      });
    });
});
```

# Mock Server im Karma-Test konfigurieren und einbinden

```
beforeAll(done => {
    provider = new PACTWeb({
      consumer: 'GreetingConsumer‘,
      provider: 'GreetingProvider‘,
      port: 1234,
      host: '127.0.0.1‘,
      log: path.resolve(process.cwd(), 'logs', 'PACT.log‘),
    });
    // required for slower CI environments
    setTimeout(done, 2000);
    // Required if run with `singleRun: false`
    provider.removeInteractions();
  });
```

*greeting.service.PACT.spec.ts*

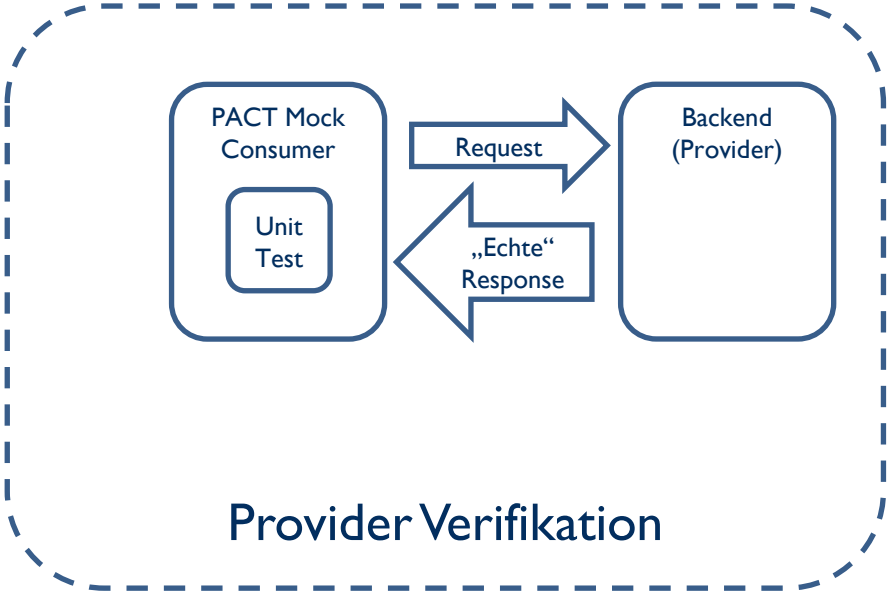
*karma.conf.js*

```
module.exports = function (config) {
  config.set({
    ... ,
    pact: [{
      cors: true,
      port: 8080,
      consumer: "GreetingConsumer",
      provider: "GreetingProvider",
      dir: "pacts",
      spec: 3
    }]
  });
};
```

# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# Server validiert Verhalten gegen den PACT (I)



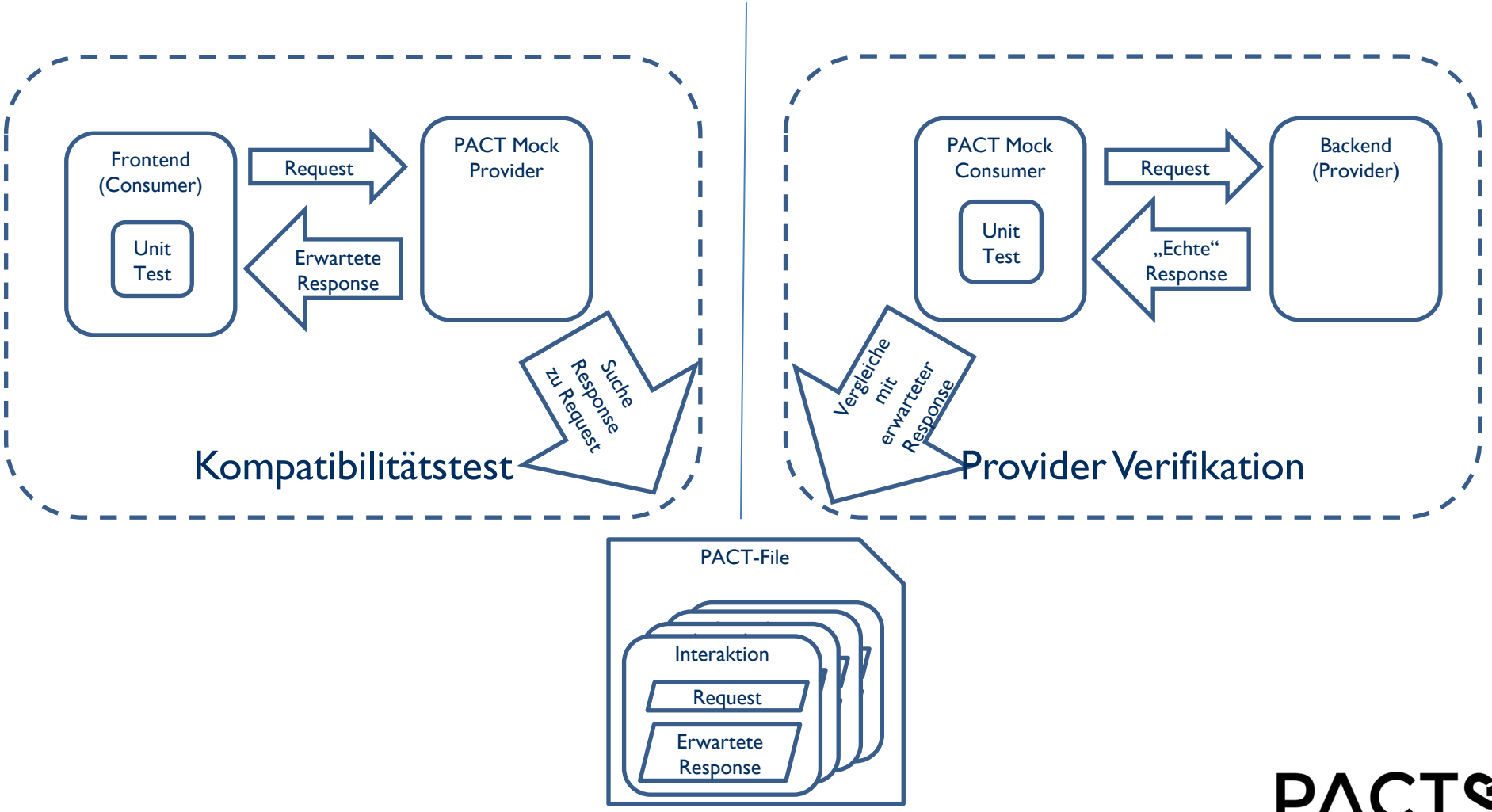
# Provider verifiziert sein Verhalten gegen den PACT

```java
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Provider("GreetingProvider")
@PACTUrl(urls = "PACTs/GreetingConsumer-GreetingProvider.json")
public class GreetingProviderVerifcationTest {

    @LocalServerPort
    int randomServerPort;

    @Autowired
    GreetingService greetingService;

    @TestTemplate
    @ExtendWith(PACTVerificationInvocationContextProvider.class)
    void PACTVerificationTestTemplate(PACTVerificationContext context) {
       context.verifyInteraction();
    }

    @BeforeEach
    void before(PACTVerificationContext context) throws Exception {
        context.setTarget(HttpTestTarget.fromUrl(new URL("http", "localhost", randomServerPort, "")));
        greetingService.reset();
    }
```

# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# Verträge automatisiert & unabhängig verifizieren




PACT

https://docs.PACT.io

# Entwicklung mit Consumer Driven Contracts

- Entkopplung von Frontend- und Service-Entwicklung durch den Einsatz von Consumer Driven Contracts
  - Definition der Interaktionen zwischen Frontend und Service in Form von PACT Files
  - Test und Entwicklung des Frontends gegen aus diesem PACT File erzeugte MockUps bzw. Kompatibilitätstests
  - Entwicklung und Verifikation der Services gegen aus PACT File erzeugte Verifikationstests
- Automatisierte Integrationstests „ohne zu integrieren“

PACT FOUNDATION

# PACT basierte Integrationstests

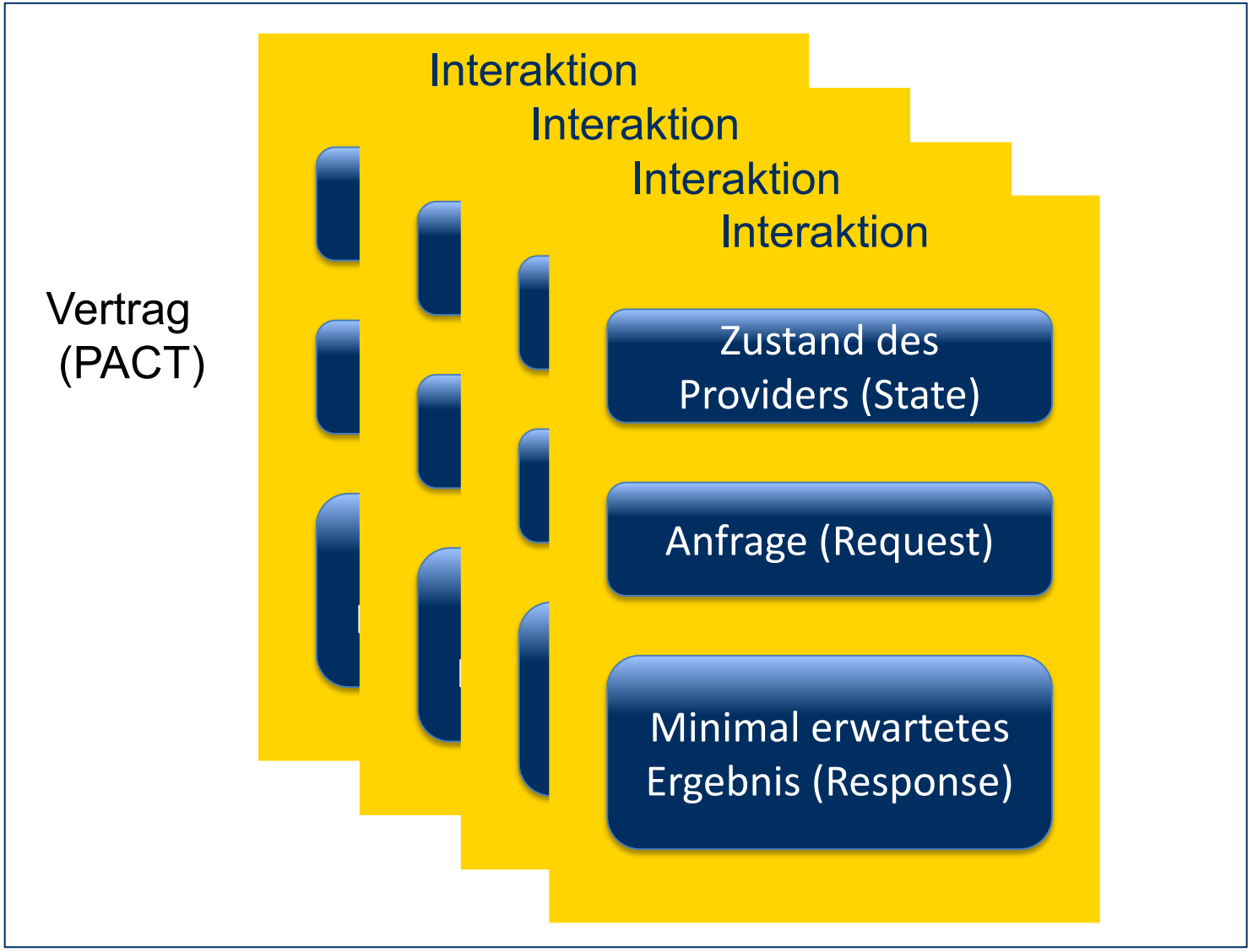
- Gut für unabhängige und schnelle Entwicklung
  - Einfache Mockups „for free“

- ABER:
  - Zustandsänderungen NICHT abbildbar!
  - Nicht geeignet für komplexe UI Flows
  - Abwesenheit von Elementen nicht testbar

https://pixabay.com/de/photos/schraubendreher-hintergrund-schraube-1008974/

# Zustände statt komplexer Abläufe (I)

# Zustände statt komplexer Abläufe (II)



# PACT Erzeugung mit State

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
	return builder
		.given("Es gibt zwei Gruesse")
		.uponReceiving("Abfrage aller Gruesse mit vorhandenen Daten")
			.path("/greetings")
			.method("GET")
		.willRespondWith()
			.status(200)
			.body(new PACTDslJsonBody()
				.array("greetings")
				.object()
					.stringValue("type", "formal")
					.stringValue("phrase", "Good morning")
					.closeObject()
				.object()
					.stringValue("type", "casual")
					.stringValue("phrase", "Hey")
					.closeObject()
			)
		.toPACT();
}
```

# Aber: Die ganzen Testdaten!!!!



https://pixabay.com/de/illustrations/panik-gro%C3%9Fe-augen-krummen-arm-1393619

# Struktur vs. Inhalt

- Akzeptanztest prüfen **Logik** und **Inhalte**
- Kompatibilitätstests prüfen **Struktrur**
- CDC stellt Kompatibiltät sicher, nicht Akzeptanz!

# Mit Matchern Struktur anstatt Inhalte verifizieren!

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
	return builder
		.given("Es gibt zwei Gruesse")
		.uponReceiving("Abfrage aller Gruesse mit vorhandenen Daten")
			.path("/greetings")
			.method("GET")
		.willRespondWith()
			.status(200)
			.body(new PactDslJsonBody()
				.array("greetings")
					.object()
						.stringType("type")
						.stringType("phrase")
						.closeObject()
					.object()
						.stringType("type")
						.stringType("phrase")
						.closeObject()
			)
		.toPACT();
}
```

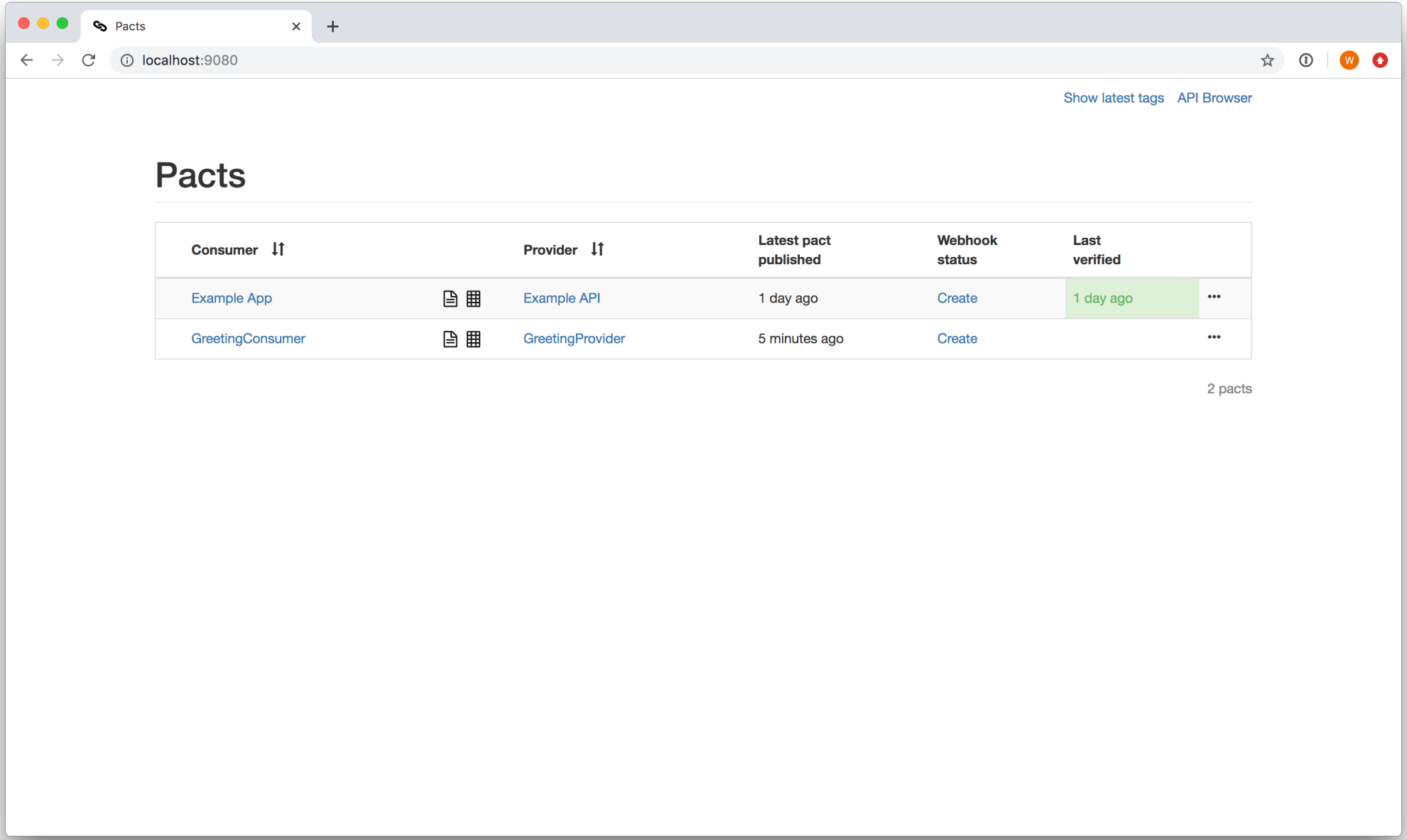
# PACT File mit Type-Matchern und Provider-States

```
{
  "description": "Abfrage eines existierenden Grusses",
  "request": {
    ...
  },
  "response": {
    "status": 200,
    "body": {
      ...
    },
    "matchingRules": {
      "body": {
        "$.greeting.type": {
          "matchers": [
            {
              "match": "type"
            }
          ],
          "combine": "AND"
        },
        "$.greeting.phrase": {
          "matchers": [
            {
              "match": "type"
            }
          ],
          "combine": "AND"
        }
      }
    }
  },
  "providerStates": [
    {
      "name": "Es gibt einen casual Gruss"
    }
  ]
}
```

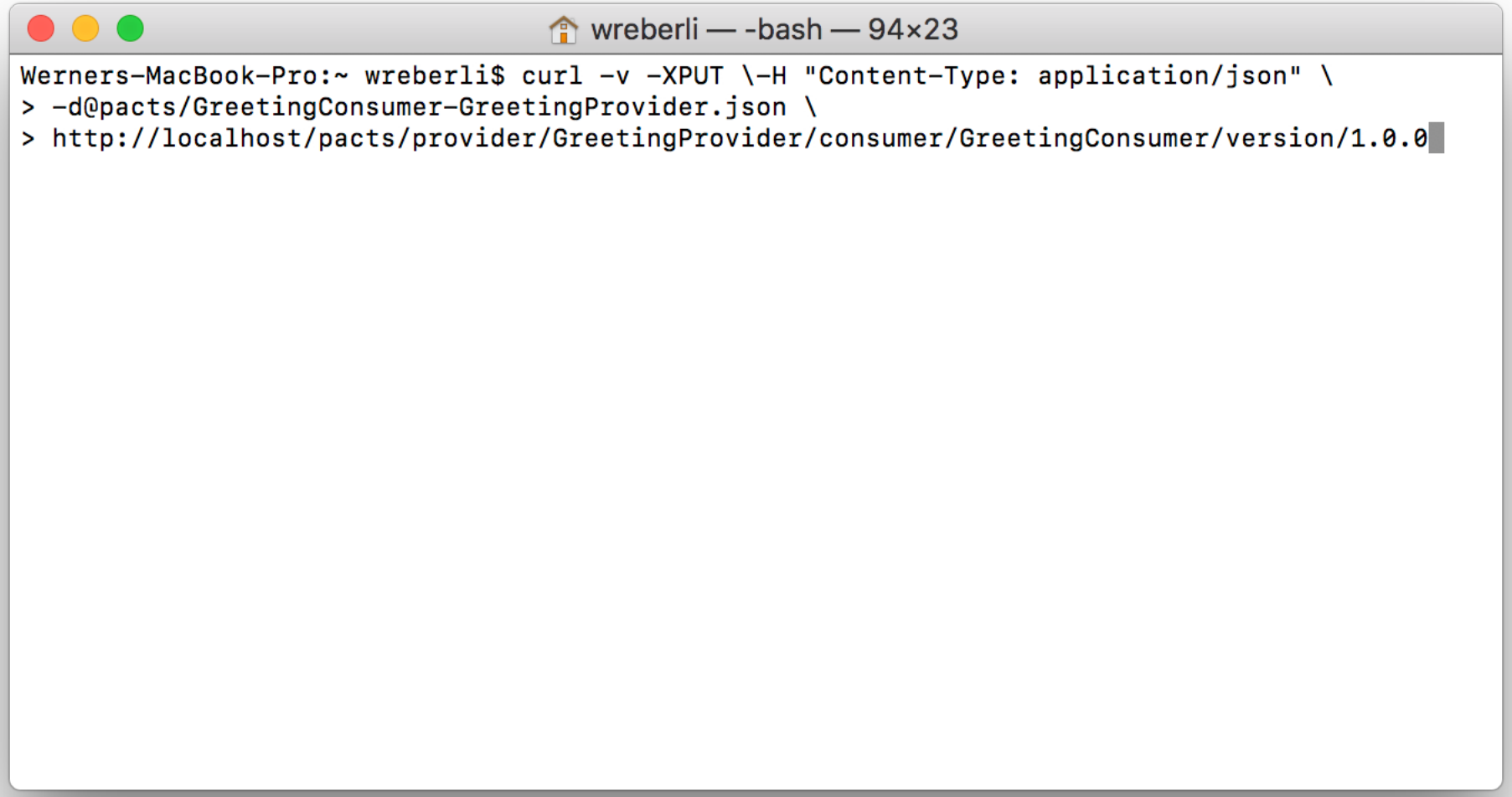
# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# PACTs Verwalten via PACT Broker



# PACT File in den Broker laden



# Gegen ein PACT File aus dem Broker verifizieren

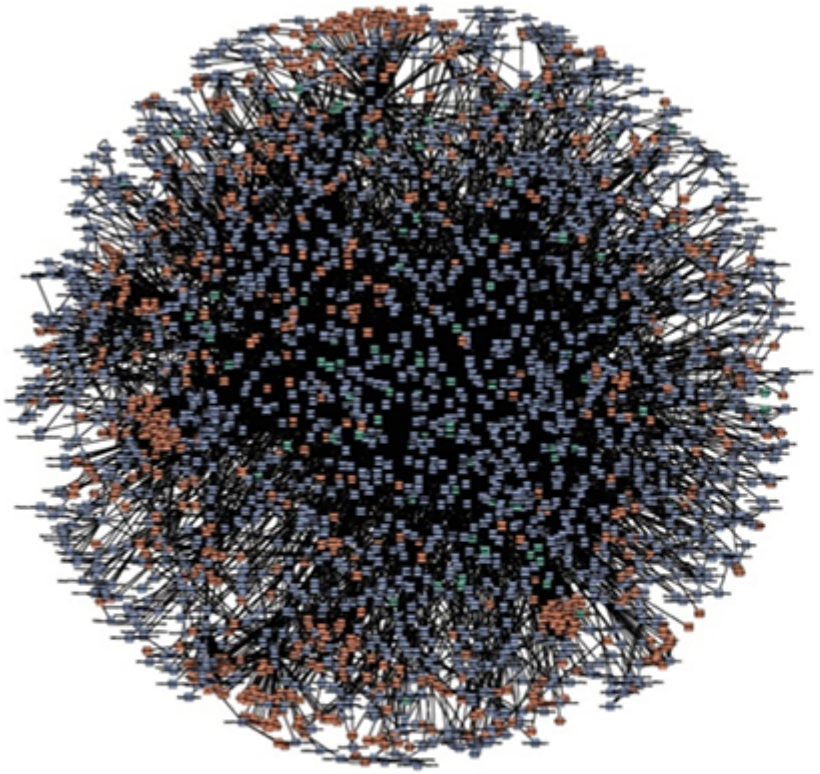
```java
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Provider("GreetingProvider")
@PactBroker(host="localhost", port="9080")
public class GreetingProviderVerifcationTest {

    @LocalServerPort
    int randomServerPort;

    @Autowired
    GreetingService greetingService;

    @TestTemplate
    @ExtendWith(PACTVerificationInvocationContextProvider.class)
    void PACTVerificationTestTemplate(PACTVerificationContext context) {
        System.setProperty("PACT.provider.version", "1.0.0");
        System.setProperty("PACT.verifier.publishResults", "true");
        context.verifyInteraction();
    }

    @BeforeEach
    void before(PACTVerificationContext context) throws Exception {
        context.setTarget(HttpTestTarget.fromUrl(new URL("http", "localhost", randomServerPort, "")));
        greetingService.reset();
    }
```

# Beispiel


HOME
w3-hide-small
w3-padding-large
MORE
Merchandise
Extras
Media

# Verträge auch bei Inter-Service-Kommunikation (I)

- Welcome to Deathstar Architecture



amazon.com

NETFLIX

https://www.appcentrica.com/the-rise-of-microservices

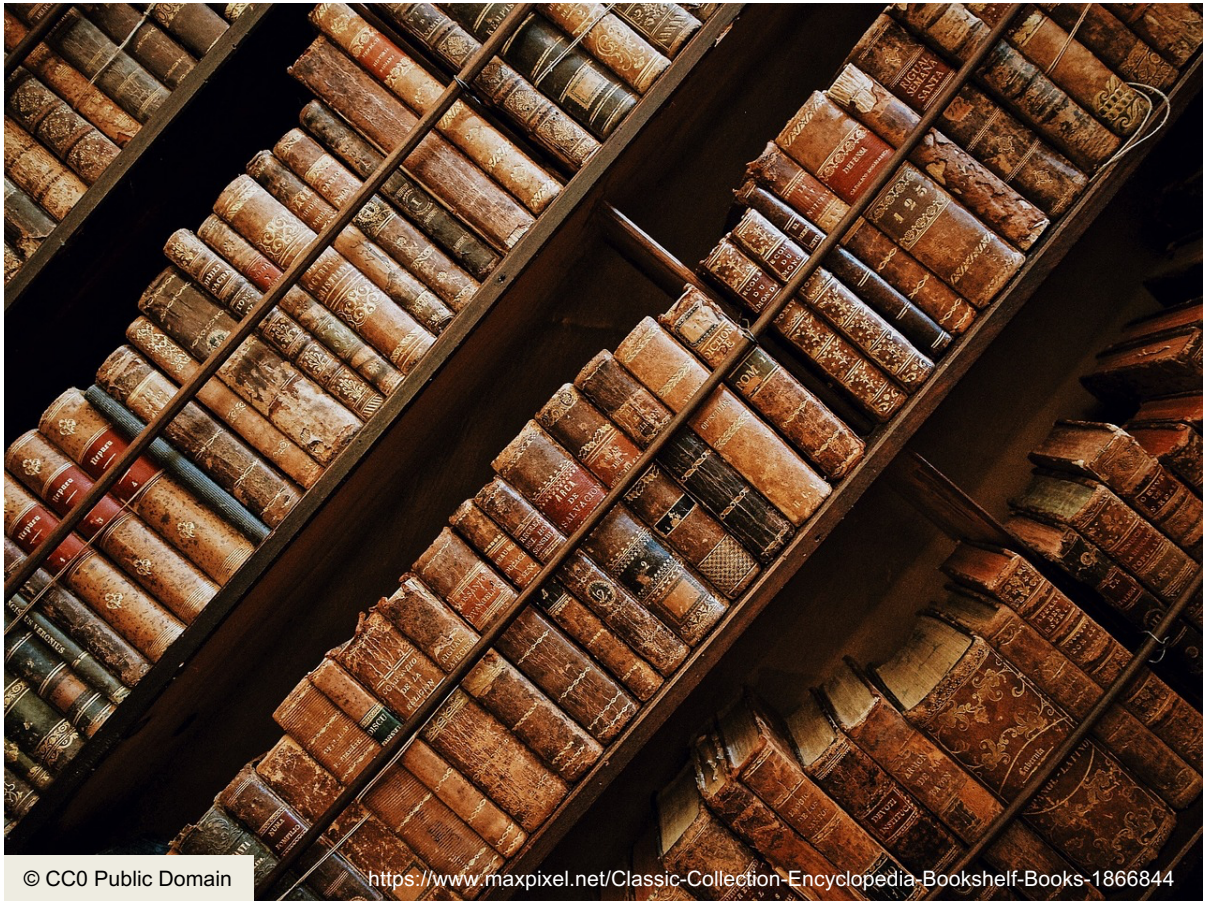
# Verträge auch bei Inter-Service-Kommunikation (I)

- Kommunikation zwischen Microservices ist eine Consumer – Provider Beziehung
- Über PACT Interaktionen kann ein Provider kompatible Änderungen absichern
- Über den PACT Broker kann ein Provider alle aktuell gültgen Verträge ermitteln ohne die Consumer direkt zu kennen

# Aber wir haben doch schon Swagger?!?

# Aber wir haben doch schon Swagger?!?

- Swagger beschreibt wie eine existierende Schnittstelle aussieht (Dokumentation)





https://www.maxpixel.net/Classic-Collection-Encyclopedia-Bookshelf-Books-1866844

# Aber wir haben doch schon Swagger?!?

- CDC beschreibt was der Consumer benötigt (Definition)

© Pixabay Licence

# Fazit

- Guter Weg zur Definition minimaler Schnittstellen
  - Der Client bekommt genau was er will (braucht?)
  - Design-by-Contract reloaded
- Entkoppelt die Entwicklung durch automatisierte „Integration ohne Integration“
- Als Mockup für komplexere Integrations-/Akzeptanztests nur bedingt geeignet
- Ideales Werkzeug bei automatisiertem Service Deployment

# Links

- PACT
  https://docs.PACT.io
- PACT JVM
  https://github.com/DiUS/PACT-jvm
- Mock Server
  https://github.com/PACT-foundation/PACT-mock_service
- PACT Broker
  https://github.com/pact-foundation/pact_broker
- Beispiele auf github
  https://github.com/wern/cdc-ws-hc-20

# Fragen?

MATHEMA

**Vielen Dank!**

werner.eberling@mathema.de
(@wer_eb)

PACT

https://docs.PACT.io