CDC

API Definition and Test with Consumer Driven Contracts

Werner Eberling

eMail: werner.eberling@mathema.de
Twitter: @Wer_Eb

https://docs.PACT.io

About the speaker

Werner Eberling

Principal Consultant / Autor

eMail: werner.eberling@mathema.de

Twitter: @Wer_Eb

OMG ... Who designed THIS API?

© Pixabay Licence

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

Maybe we should ask the clients first?

© Pixabay Licence

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

But wait, THAT was NOT the deal!!!

© Pixabay Licence

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

Let‘s make a contract

© Pixabay Licence

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

Design by Contract – the end of flexibility?

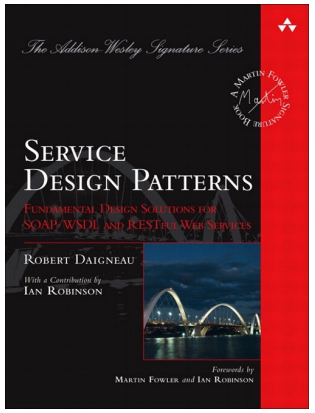
„be conservative in what you do,
be liberal in what you accept from others “

– RFC 761

Consumers as Tolerant Reader

- Consumer define what they do and need…

© Pixabay Licence



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

- … and ignore everything else.

© Pixabay Licence

But … what about the contract?

© Pixabay Licence

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

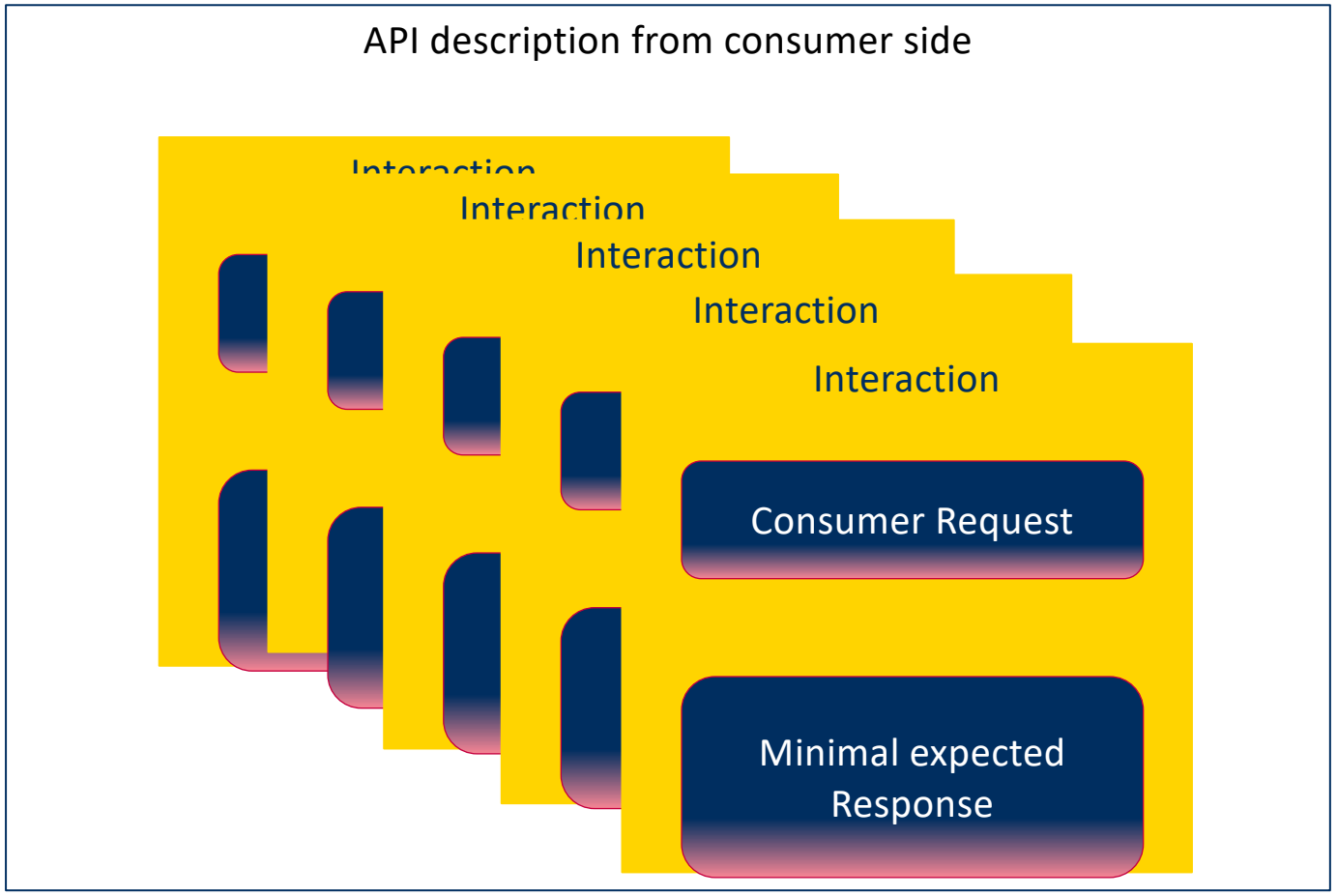
We need to test ... EVERYTHING!!!

How do we test in real life?

Burning down the house?

Or better simulate signal input...

Interactions over static interface definitions



https://docs.PACT.io

vs.

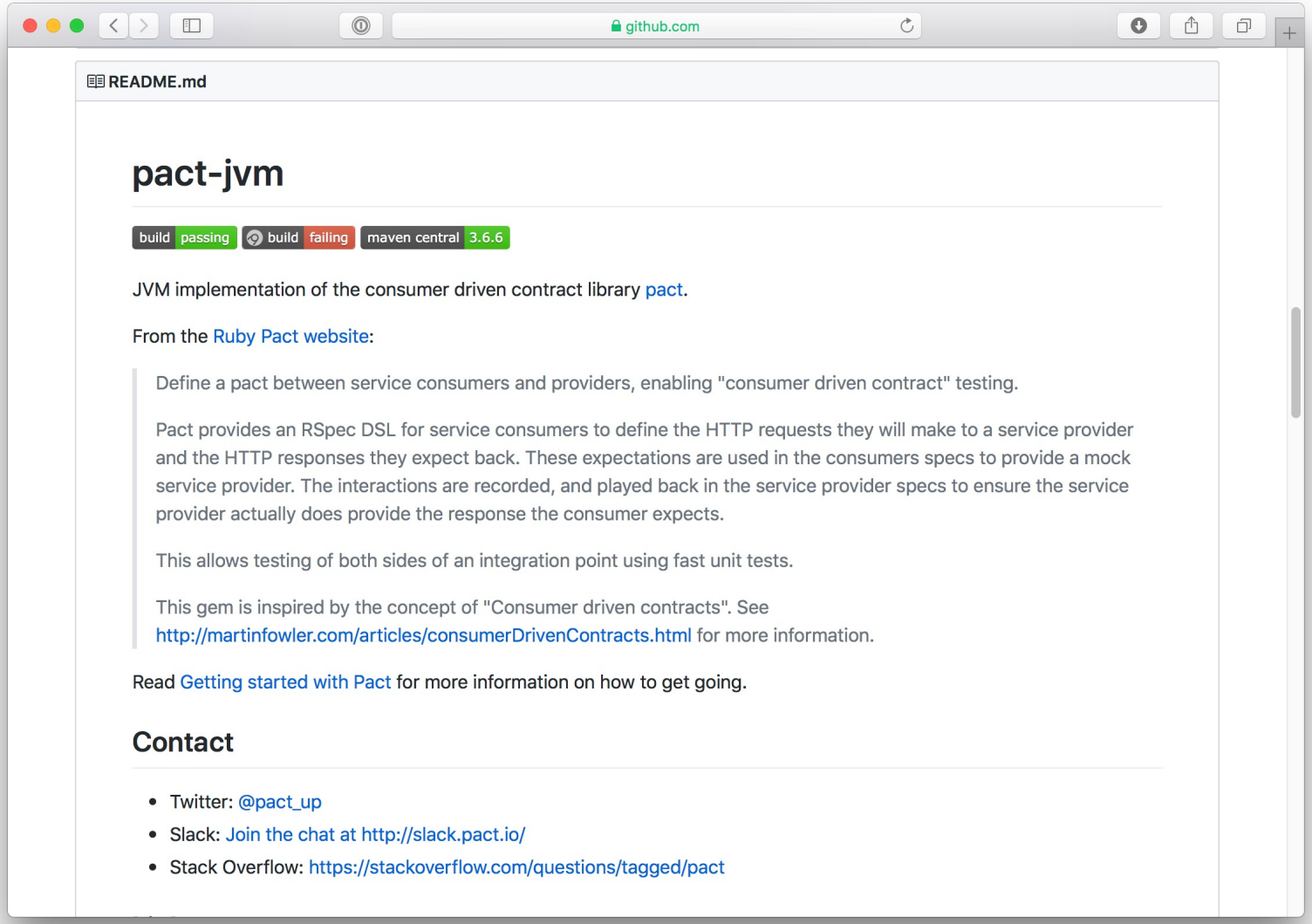
Spring Cloud Contract

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

Contract Testing using 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 using PACT – PACT-jvm

How does a contract look like?

```
{
  "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"
    }
  }
}
```

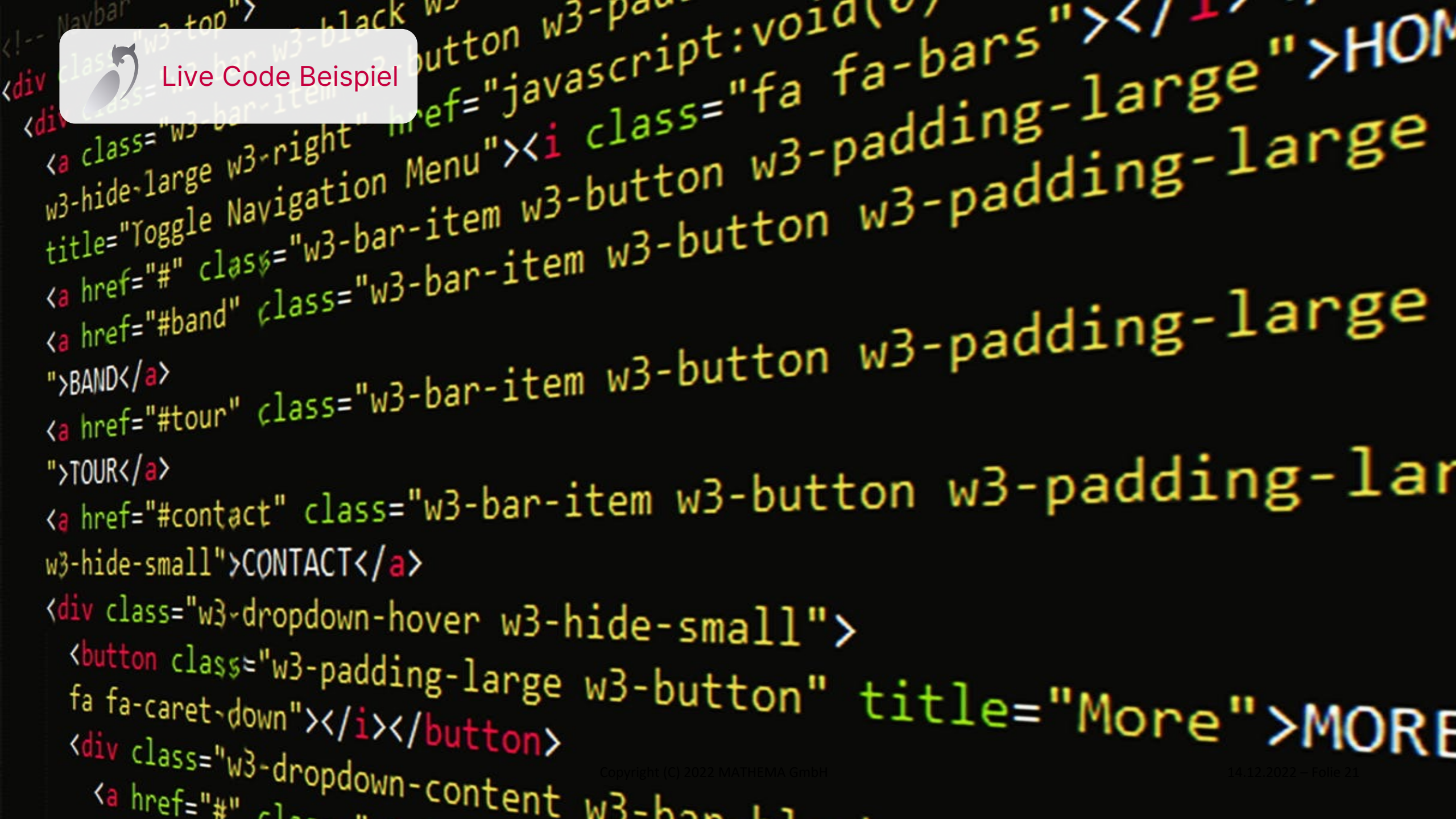
Bonusfeature: PACTs as Mockup for integration tests?

cdc-sample — pact-stub-service ./pacts/GreetingConsumer-GreetingProvider.json -p 9090 -h localhost — 124×23

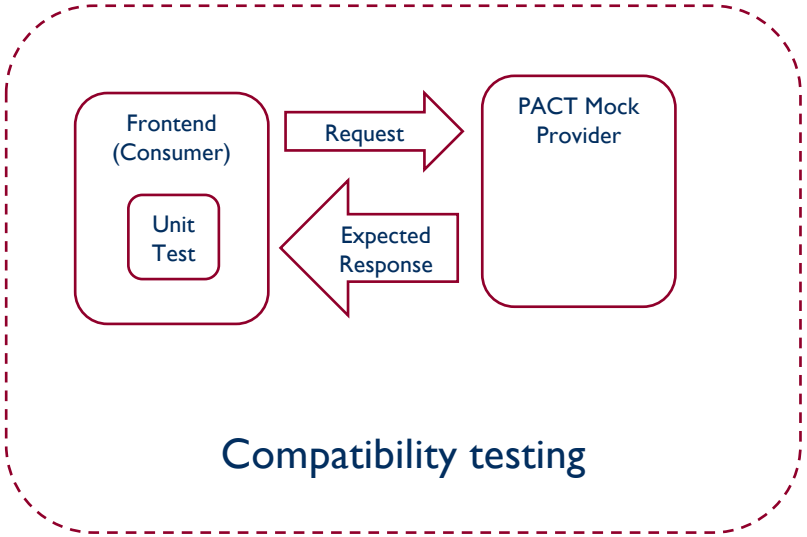
```
Werners-MBP:cdc-sample wreberli$ pact-stub-service ./pacts/GreetingConsumer-GreetingProvider.json -p 9090 -h localhost
INFO: Loading interactions from ./pacts/GreetingConsumer-GreetingProvider.json
I, [2019-05-08T15:35:28.019161 #58468]  INFO -- : Registered expected interaction GET /campus/greeting
D, [201
  "desc
  "requ
    "me
    "pa
  },
  "resp
    "st
    "he
    },
    "bo
      "
    }
  }
}
INFO  W
INFO  r
INFO  W
```

cdc-sample — pact-stub-service ./pacts/GreetingConsumer-GreetingProvider.json -p 9090 -h localhost — 124×31

```
INFO  WEBrick 1.4.2
INFO  ruby 2.3.7 (2018-03-28) [universal.x86_64-darwin17]
INFO  WEBrick::HTTPServer#start: pid=58468 port=9090
I, [2019-05-08T15:36:25.017595 #58468]  INFO -- : Received request GET /campus/greeting
D, [2019-05-08T15:36:25.017744 #58468] DEBUG -- : {
  "path": "/campus/greeting",
  "query": "",
  "method": "get",
  "headers": {
    "Host": "localhost:9090",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Upgrade-Insecure-Requests": "1",
    "Cookie": "_gid=GA1.1.1010395921.1557305790; _ga=GA1.1.1159609082.1557305790",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Saf
ari/605.1.15",
    "Accept-Language": "de-de",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "keep-alive",
    "Version": "HTTP/1.1"
  }
}
I, [2019-05-08T15:36:25.017946 #58468]  INFO -- : Found matching response for GET /campus/greeting
D, [2019-05-08T15:36:25.018067 #58468] DEBUG -- : {
  "status": 200,
  "headers": {
  },
  "body": {
    "greeting": "Hallo vom MATHEMA Campus!"
  }
}
```



Consumer checks compatibility against the PACT



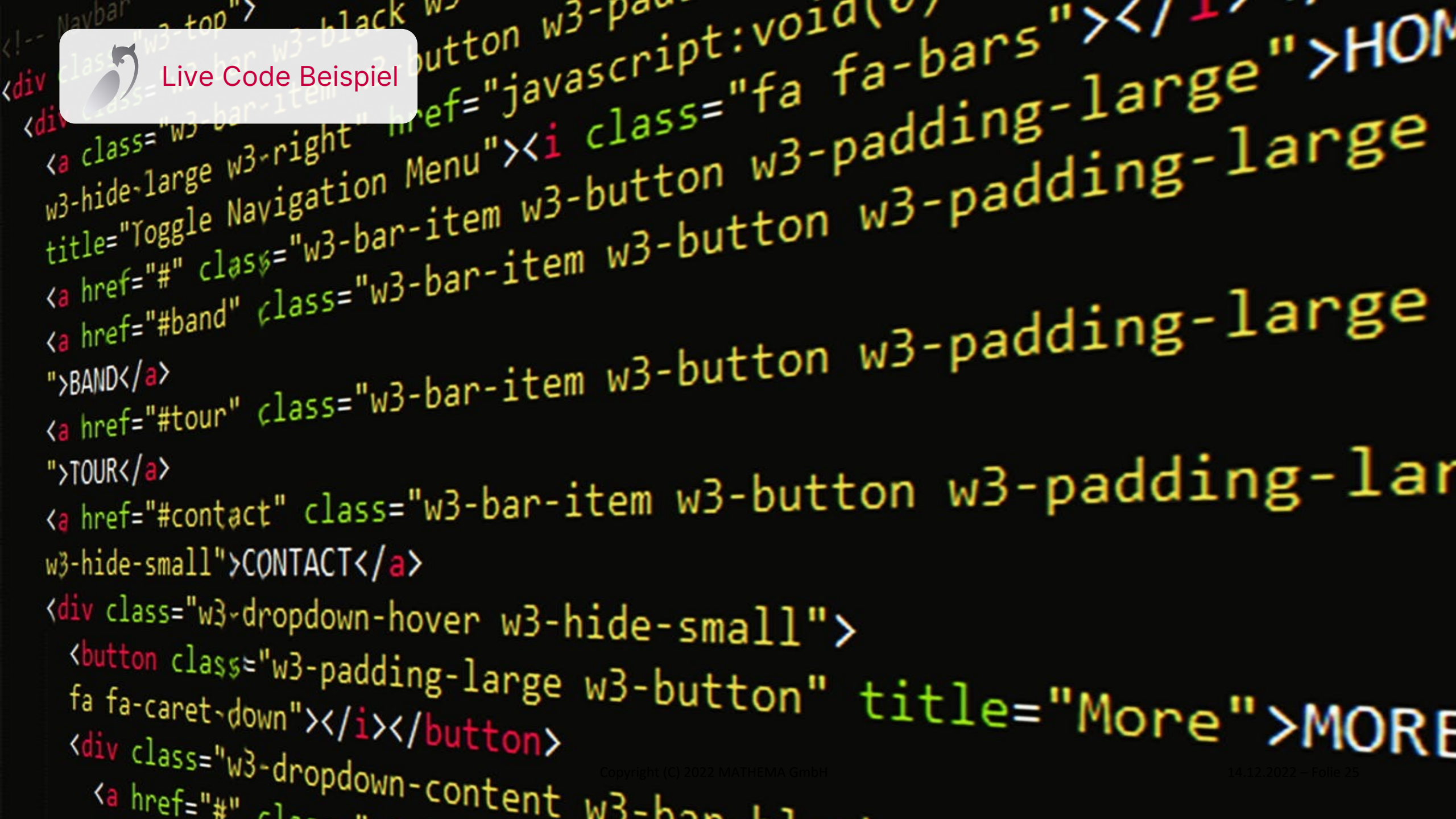
PACT creation using UnitTests

```
@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();
}
```

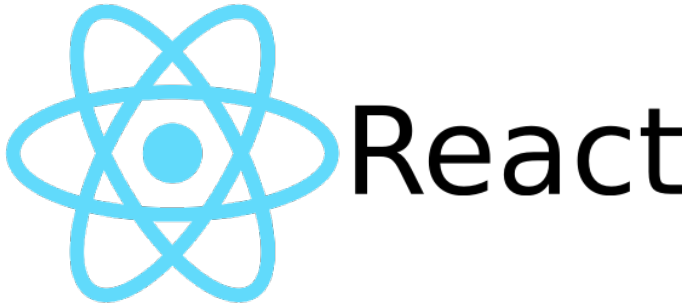
Consumer checks ist own compatibility in the same test

```
@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());
}
```

In a “real“ project we would use “real“ client classes, not HttpClient!



Frontends are build in JavaScript or TypeScript these days



Create and verify PACT using Karma

```
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);
      });
    });
});
```

Embed Mock Server into Karma testcase

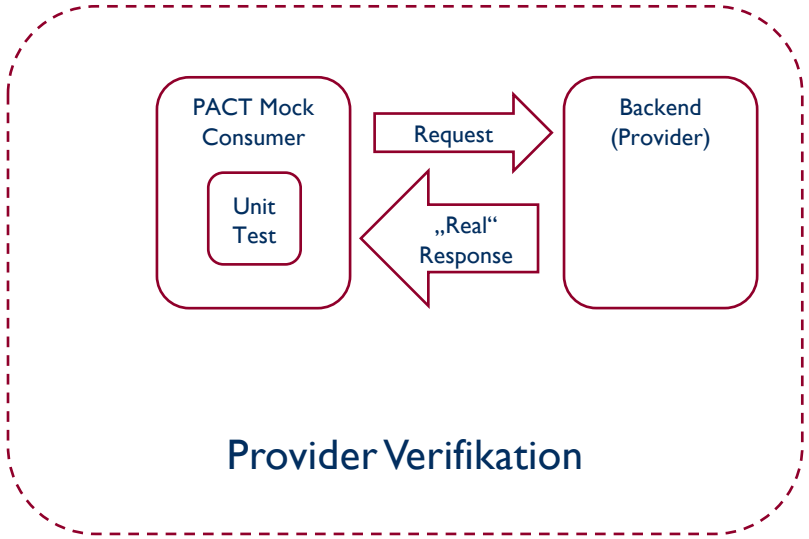
```
beforeAll(done => {
    provider = new PACTWeb({
      consumer: 'GreetingConsumer‘,
      provider: 'GreetingProvider‘,
      port: 1234,
      host: '127.0.0.1‘,
      log: path.resolve(process.cwd(), 'logs', 'PACT.log‘),
    });
    // required for slower CI environments
    setTimeout(done, 2000);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

greeting.service.PACT.spec.ts

karma.conf.js

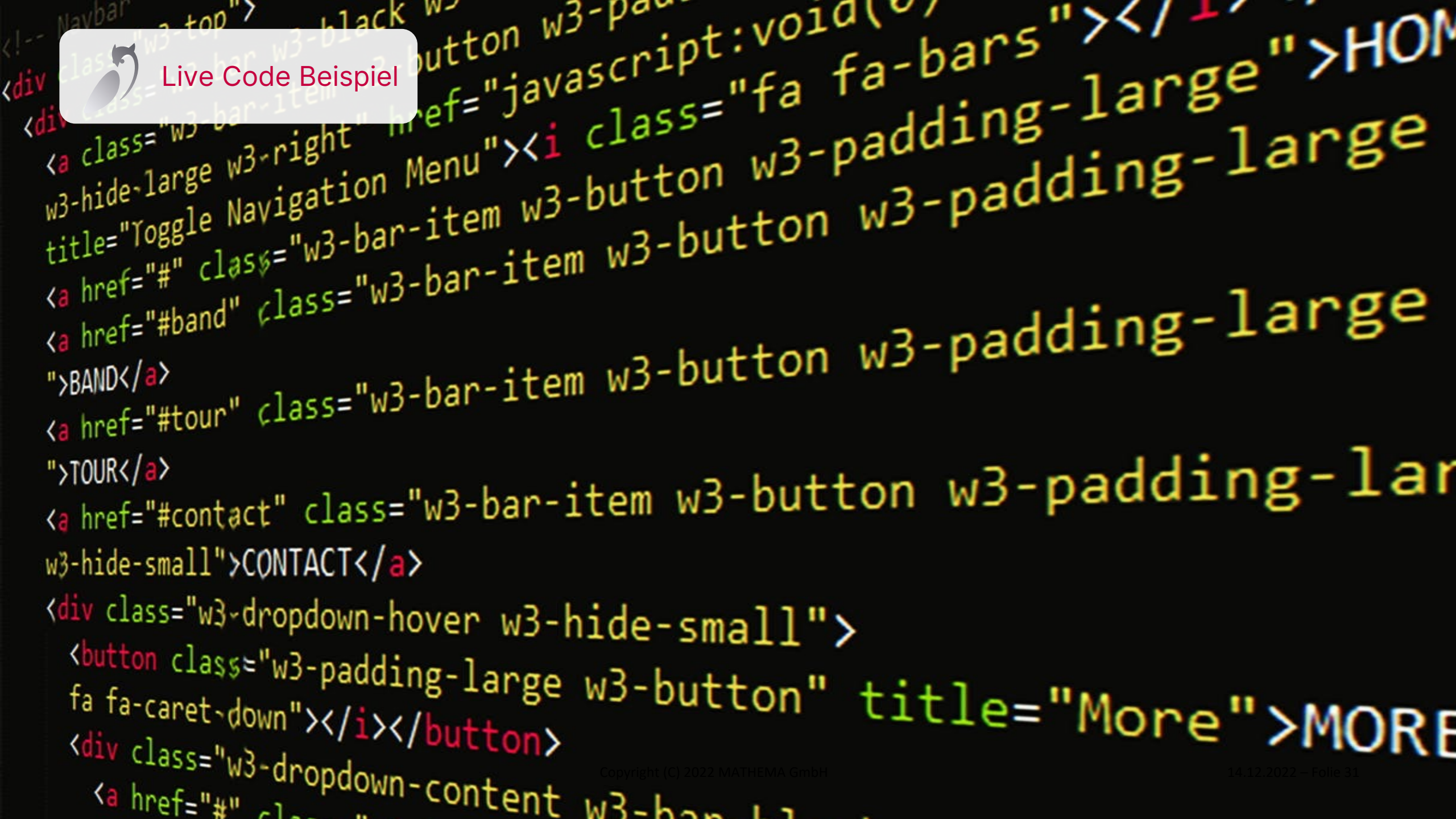
```
module.exports = function (config) {
  config.set({
    ... ,
    pact: [{
      cors: true,
      port: 8080,
      consumer: "GreetingConsumer",
      provider: "GreetingProvider",
      dir: "pacts",
      spec: 3
    }]
  });
};
```

Provider validates against PACT (I)

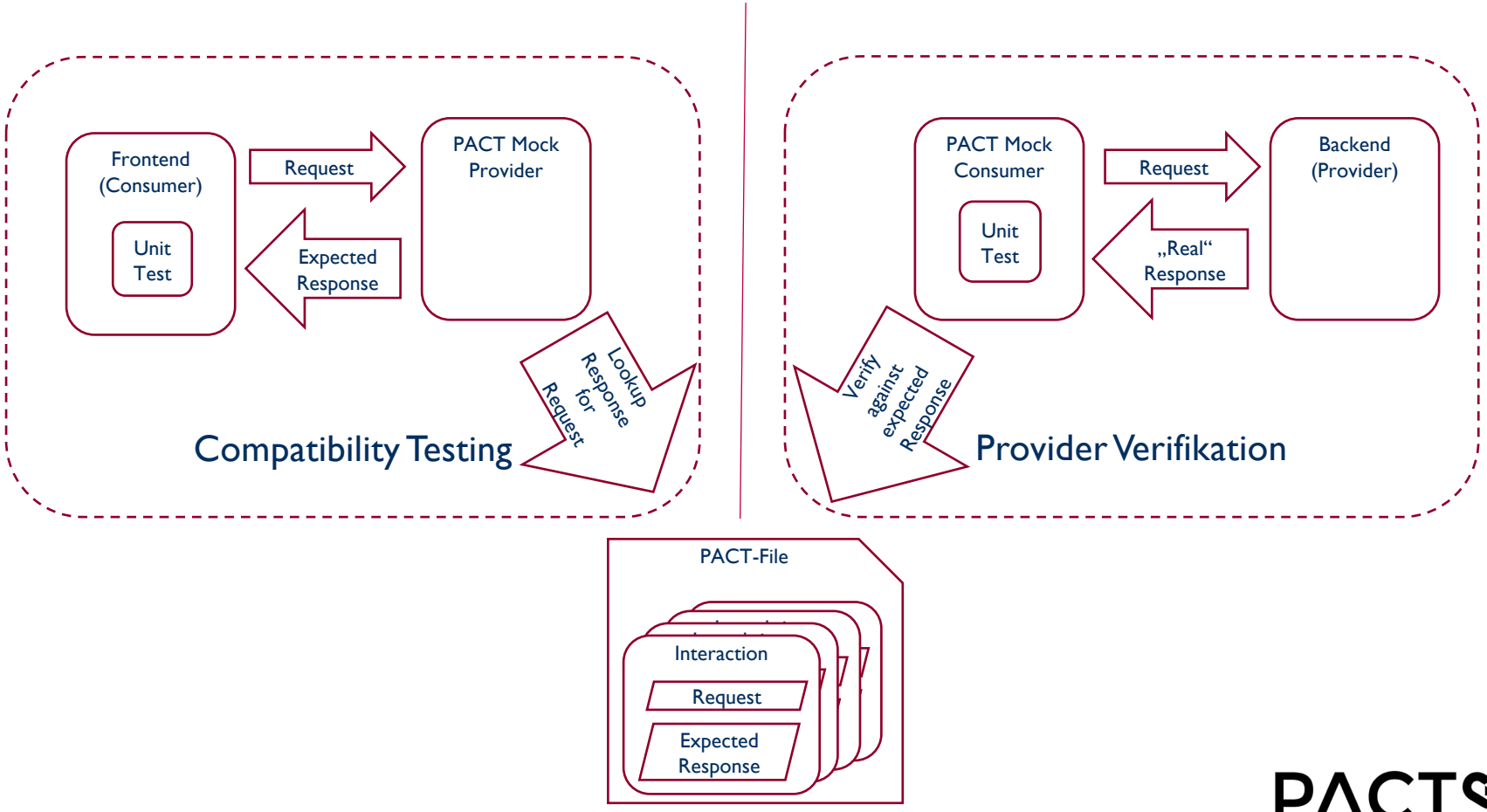


Provider validates against PACT

```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();
    }
```



Automate PACTs & validate indenpendentaly



PACTS
https://docs.PACT.io

Development Process based on CDC

- CDC decouples development of Consumer (e.g. Frontend) and Producer (e.g. Backend)
  - Definition of interactions between Consumer and Producer as PACT Files
  - Consumer development / test based on mockups and PACT compatbility tests
  - Consumer development / test based on PACT verification tests

- Automated integration testing „without real integration“!!!

PACT FOUNDATION

PACT for acceptance testing?

- PACT eases independent development of Consumer and Provider

  Simple mockups „for free“

- BUT:

  No state change supported!

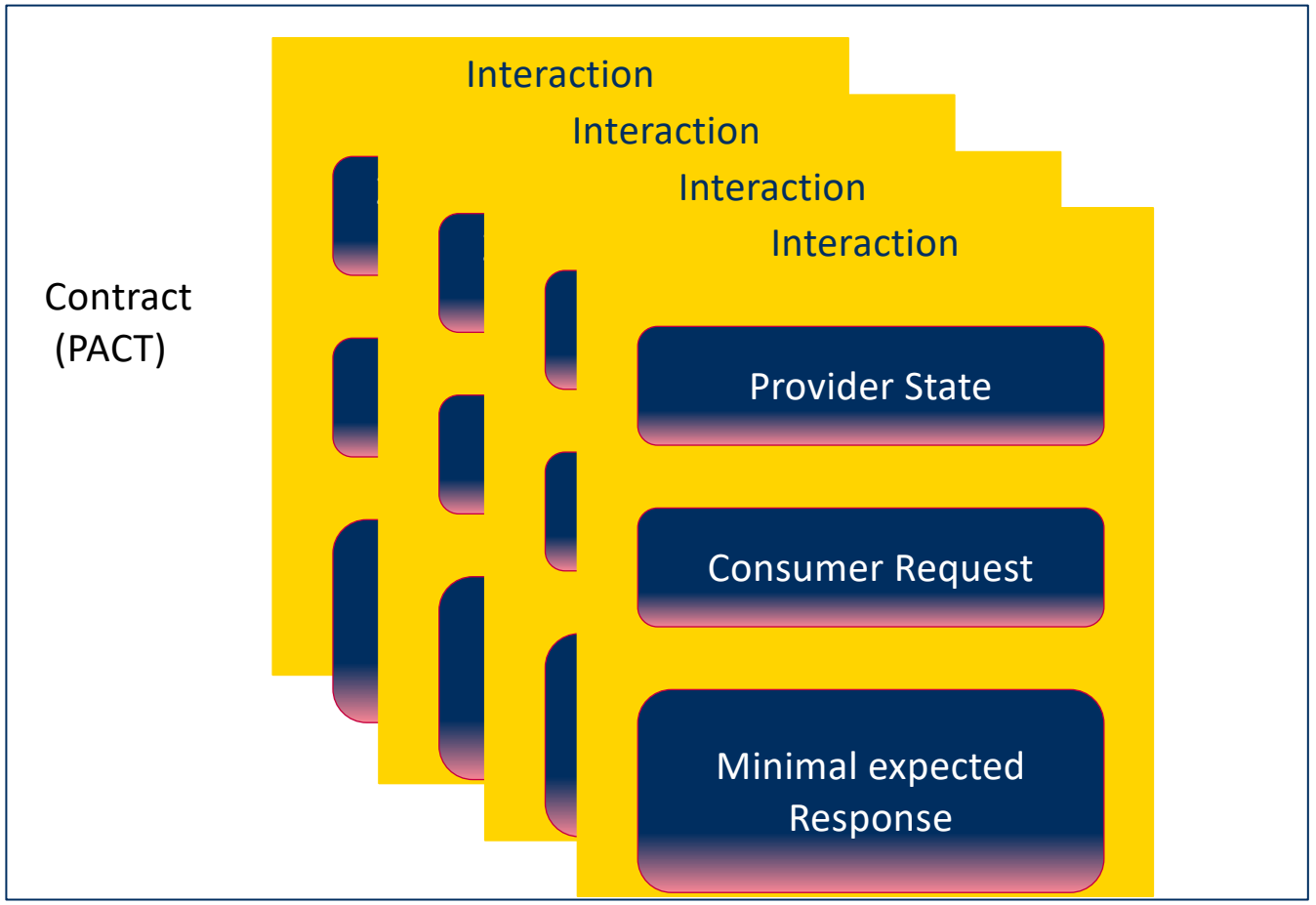
  Not suitable for complex frontend flows

  Absence of attributes not testable

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

States instead of complex flows (I)

States instead of complex flows(II)



PACT definition of Provider States

```
@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();
}
```

BUT: All the testdata!!!!

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

Structure vs. Content

- Acceptance testing deals with **Content** and **Business Logic**
- Compatibility tests are checking the **Structure**

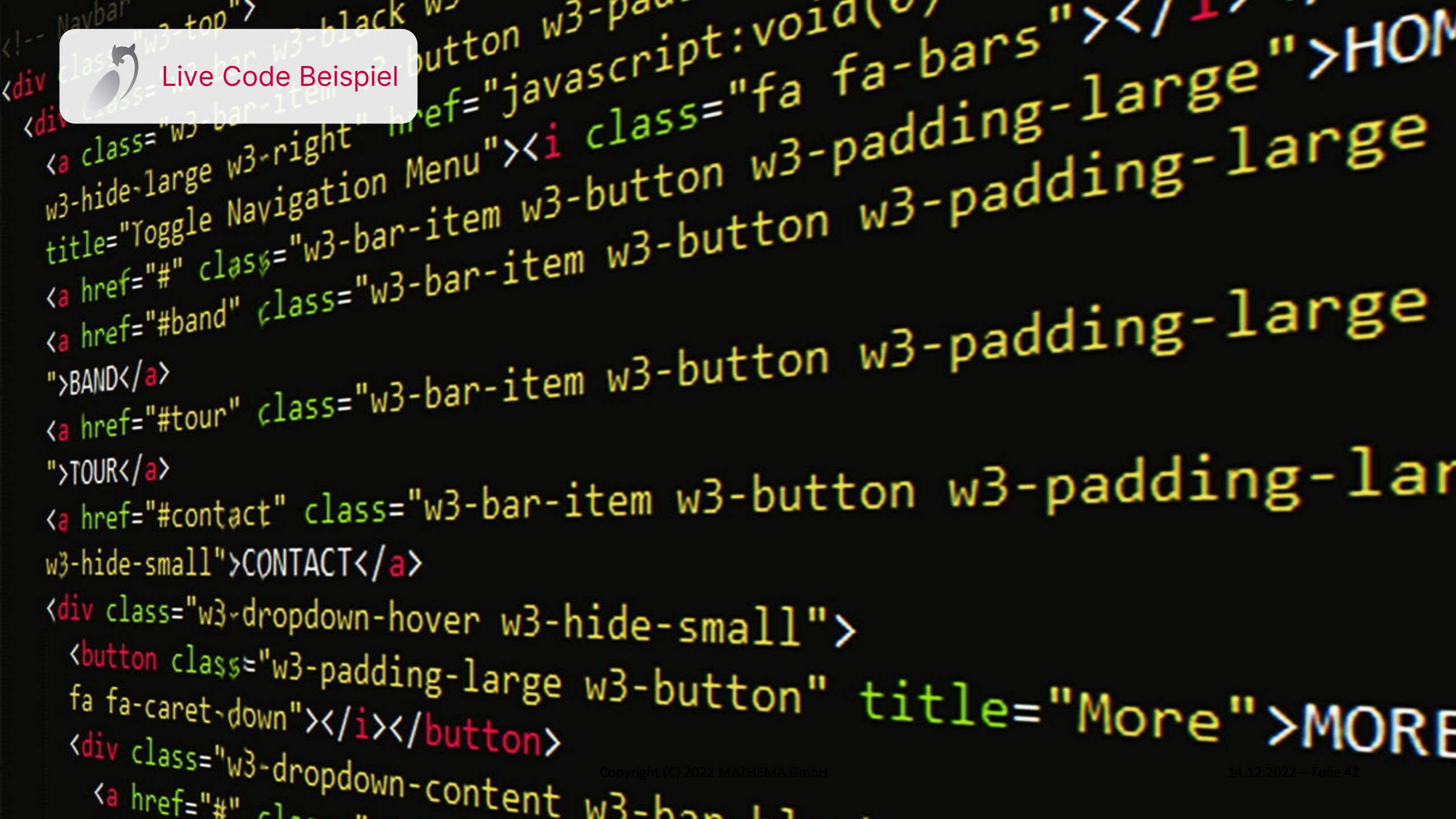
- CDC ensures **Compatibility, not Acceptance**!!!

Advanced verification using Matchers

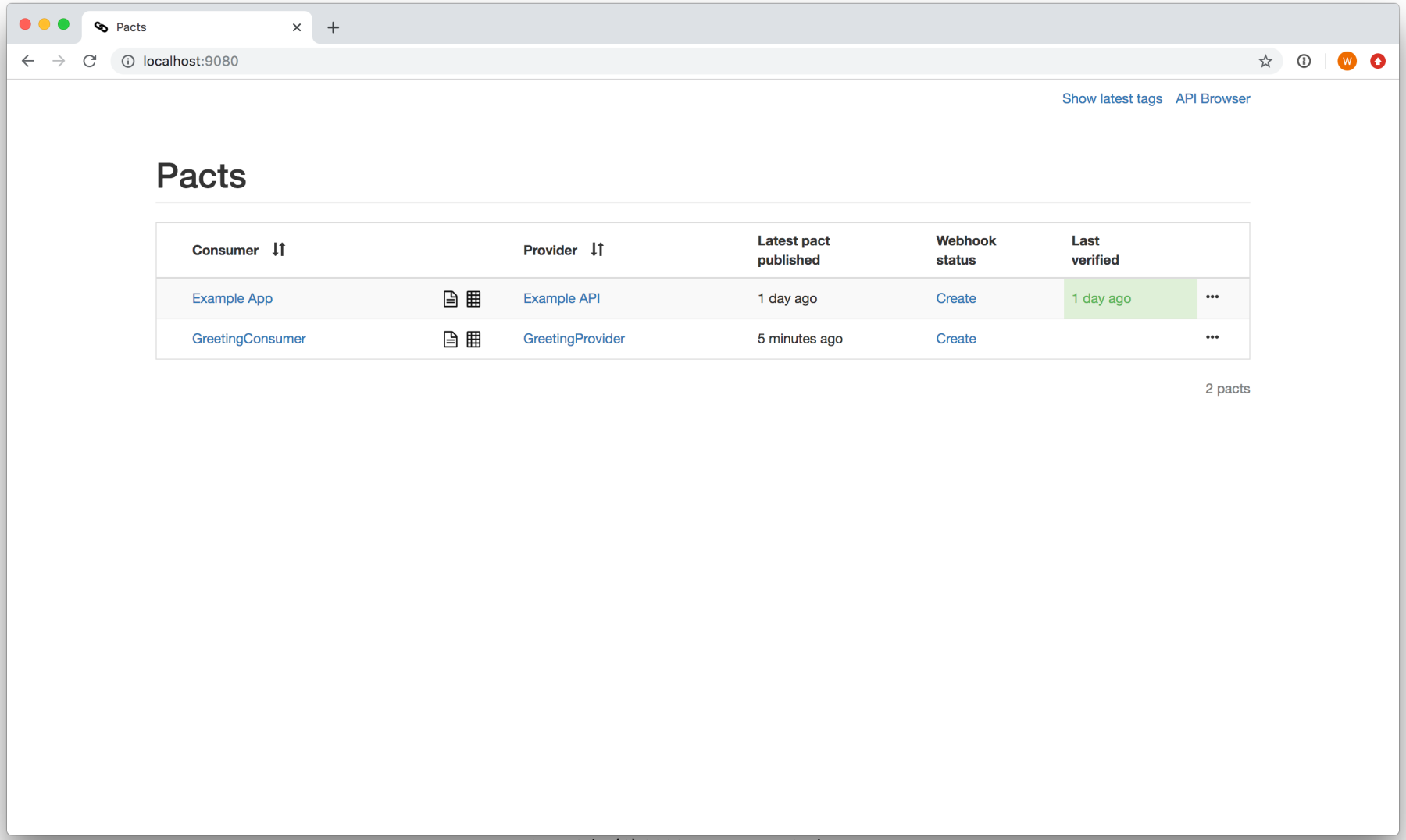
```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
        return builder
                .given("Es gibt zwei Gruesse")
                .uponReceiving("Abfrage aller Gruesse mit vorhandenen Daten")
                        .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 using Type-Matchers and 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"
    }
  ]
}
```



Organizing and exchanging PACTs via PACT Broker



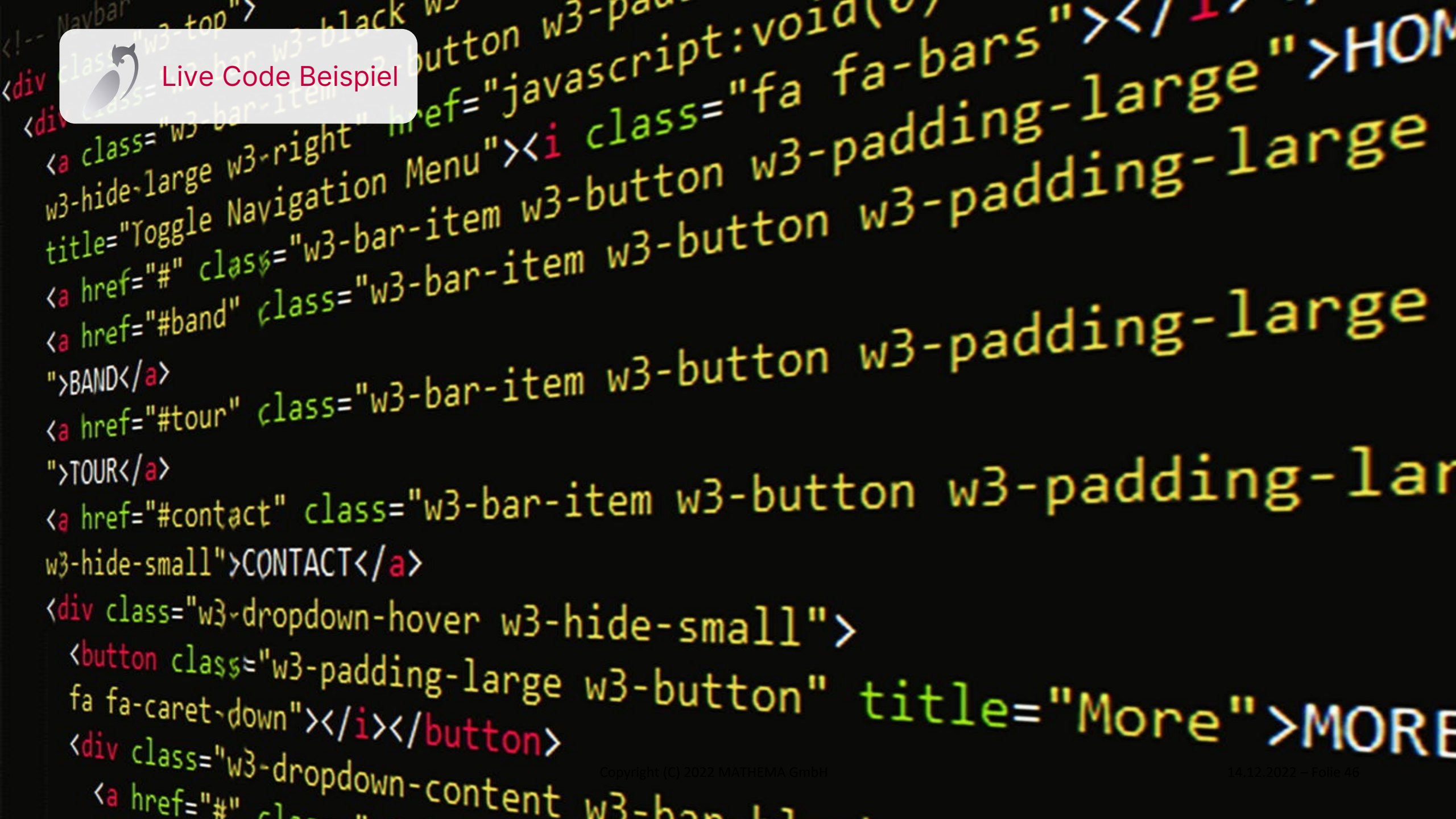
PACT File Upload

wreberli — -bash — 94×23

```
Werners-MacBook-Pro:~ wreberli$ curl -v -XPUT \-H "Content-Type: application/json" \
> -d@pacts/GreetingConsumer-GreetingProvider.json \
> http://localhost/pacts/provider/GreetingProvider/consumer/GreetingConsumer/version/1.0.0
```

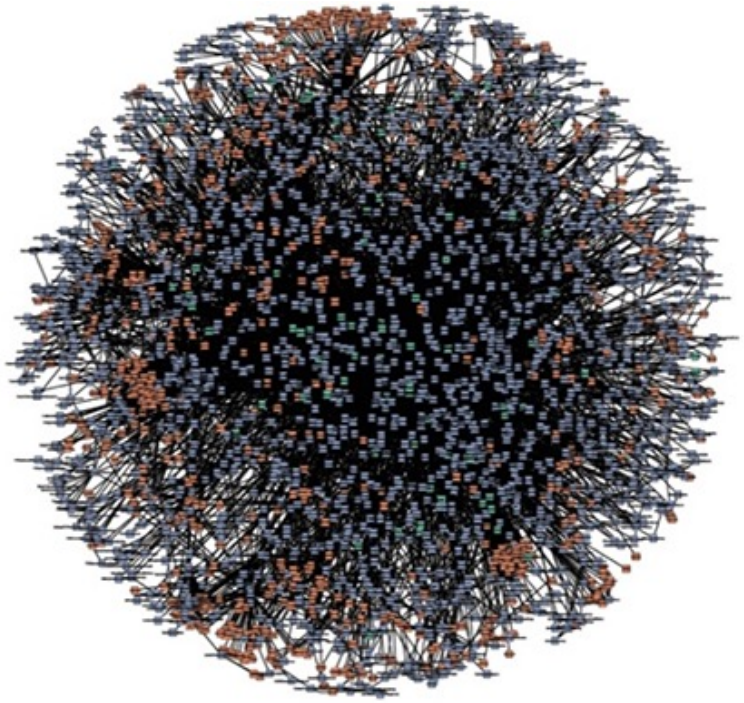
Verify against a PACT File stored in the Broker

```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();
    }
```



Using contracts for interservice communication (I)

- Welcome to Deathstar Architecture



amazon.com

NETFLIX

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

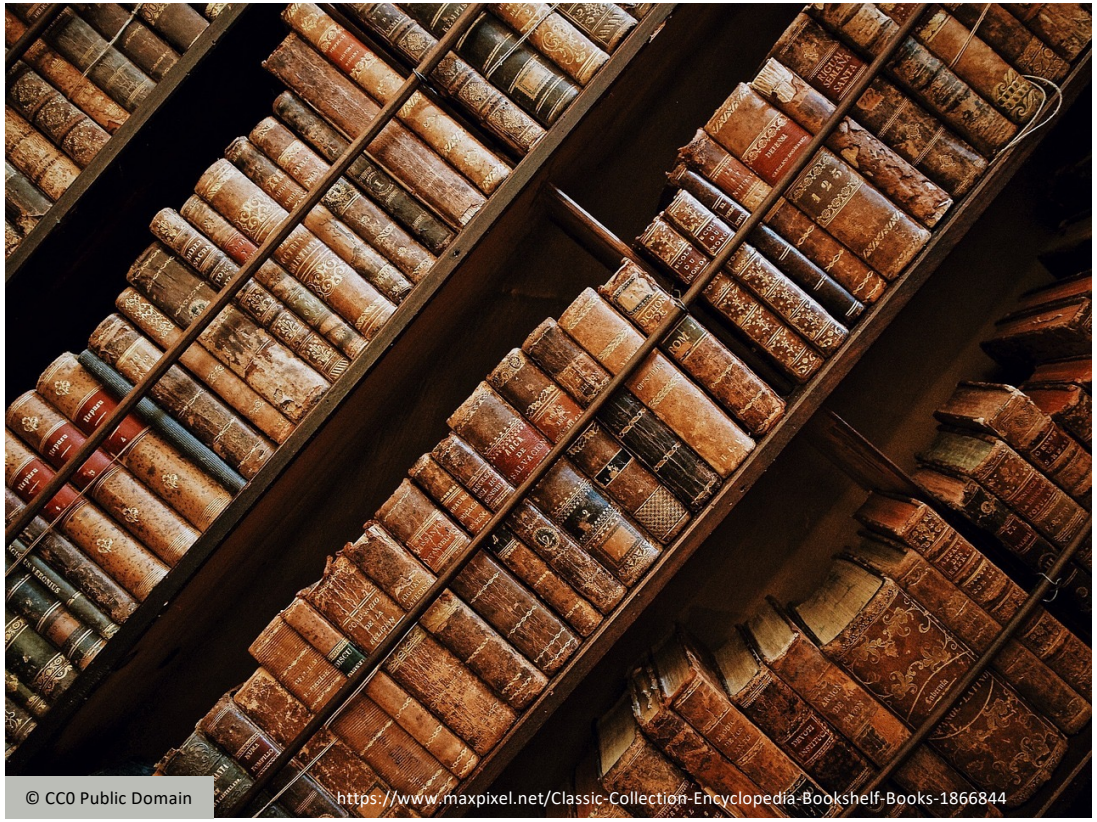
Using contracts for interservice communication (II)

- Communication between Microservices is a Consumer – Provider relationship
- PACT interactions help to check Provider updates for compatibility
- PACT Broker helps to identify all valid contracts without contacting any possible Consumer directly (automated process!!!)

But hey, we already have Swagger, right?!?

But hey, we already have Swagger, right?!?

- Swagger syntactically describes an existing Interface (Documentation)



But hey, we already have Swagger, right?!?

- CDC describes what a Consumers need (Definition)

© Pixabay Licence

Wrap-Up

- Great way to define minimal interfaces
  - Consumer get‘s (only) what (s)he really needs
  - Design-by-Contract reloaded
- Decoupling of development by automated „integration tests without integration“
- Ideal approach to ensure compatibility within automated service deployment
- Not meant to be used for mocking or acceptance testing!

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
- Examples on github
  https://github.com/wern/cdc-ws-hc-20

Thank you! Questions?

Werner Eberling

eMail: werner.eberling@mathema.de
Twitter: @Wer_Eb

Examples: https://github.com/wern/cdc-ws-hc-20

Examples?
Scan me ;)

www.mathema.de