

Beyond Consumer-Driven Contract Testing



NICOLE RAUCH
software development &
development coaching

Soooo...

- ▶ We want to build this nice webapp / distributed system / these microservices
- ▶ We want to make sure “Frontend” and “Backend” play nicely together

Soooo...

- ▶ We want to build this nice webapp / distributed system / these microservices
- ▶ We want to make sure “Frontend” and “Backend” play nicely together
- ▶ HOW???

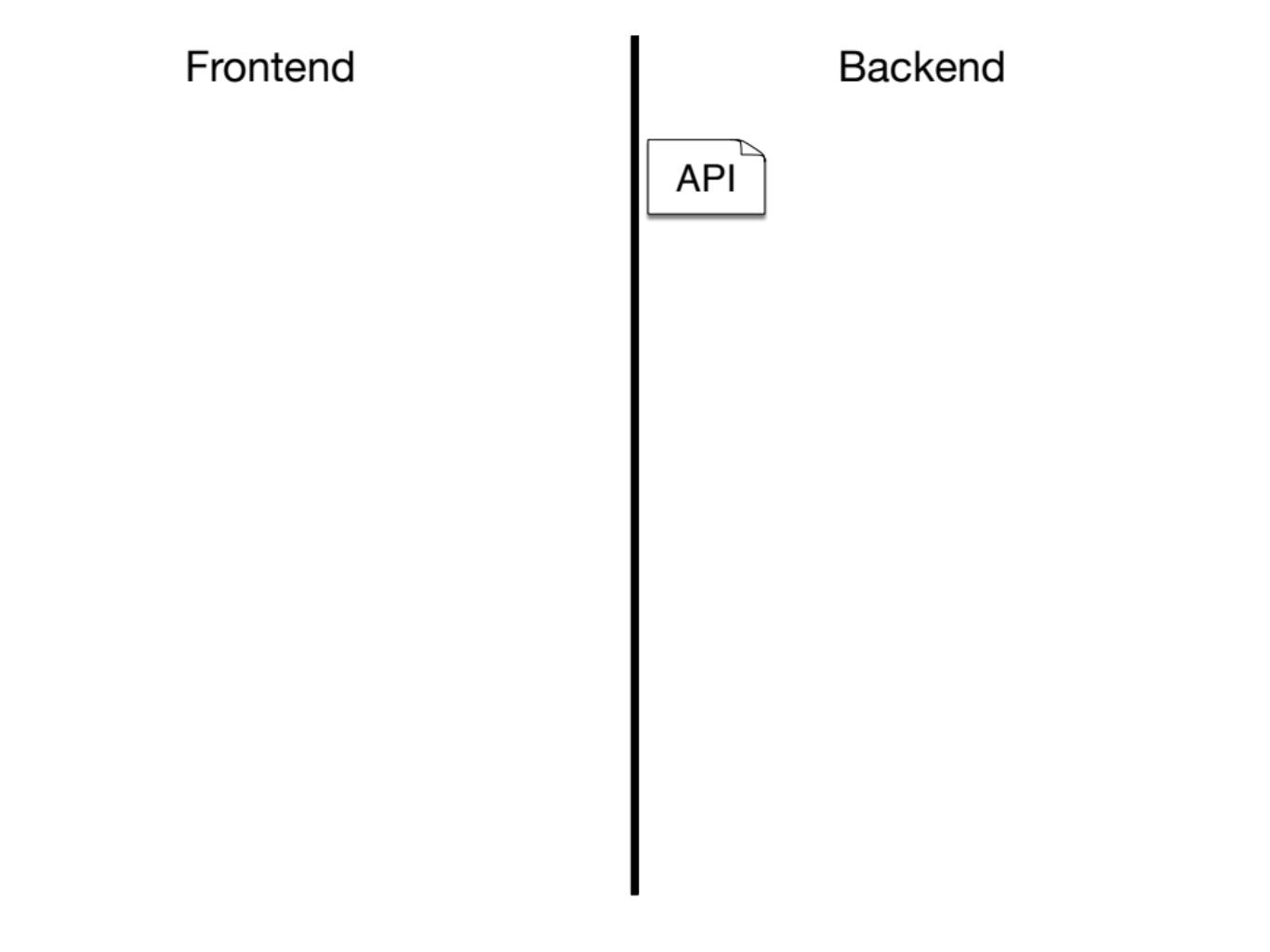
“Super-Naïve” Approach

Frontend

Backend

Frontend

Backend

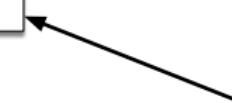


API

A diagram illustrating a system architecture. A thick vertical black line divides the space into two main sections: 'Frontend' on the left and 'Backend' on the right. In the 'Backend' section, there is a white rectangular box with a small triangular cutout at the top-right corner, containing the text 'API'. This represents a communication interface between the two parts of the system.

Frontend

Backend



Frontend

Backend



Frontend

Backend

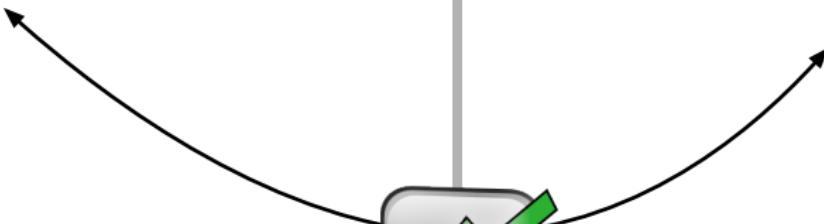


↑

But . . .

Frontend

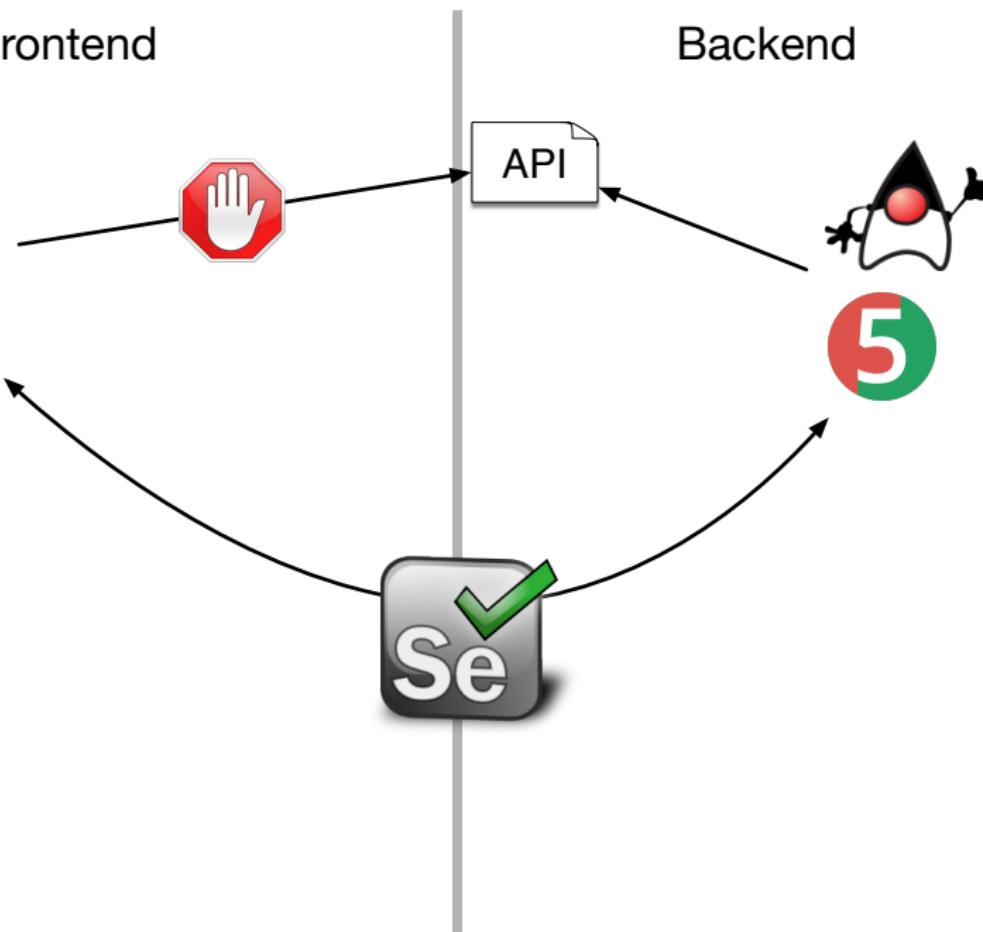
Backend



↗

Frontend

Backend



Frontend

Backend



Frontend

Backend



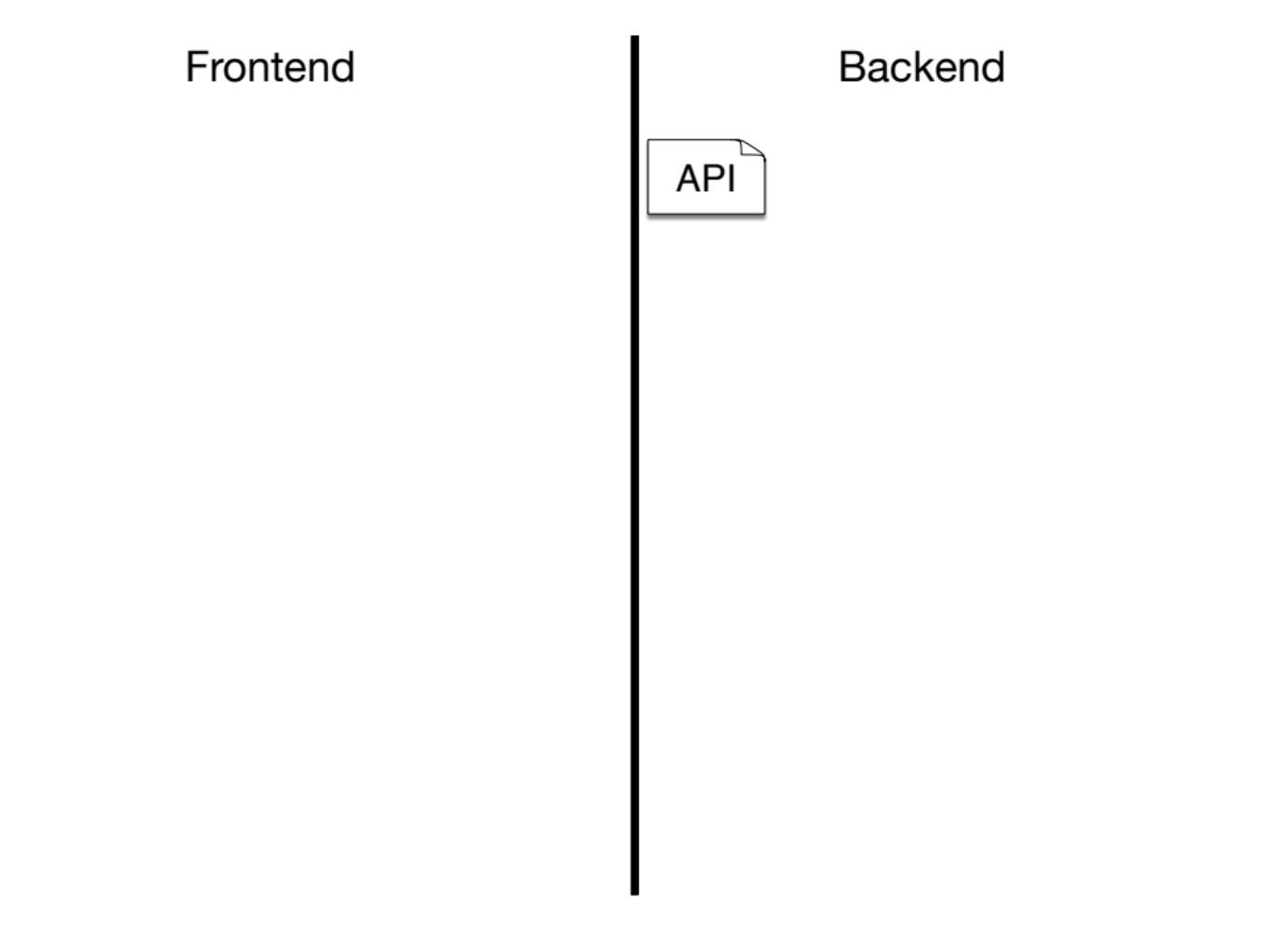
“Still Quite Naïve” Approach

Frontend

Backend

Frontend

Backend



API

A diagram illustrating a system architecture. A thick vertical black line divides the space into two main sections: 'Frontend' on the left and 'Backend' on the right. In the 'Backend' section, there is a white rectangular box with a small triangular cutout at the top-right corner, containing the text 'API'. This represents a communication interface between the two parts of the system.

Frontend

Backend



SINON.JS



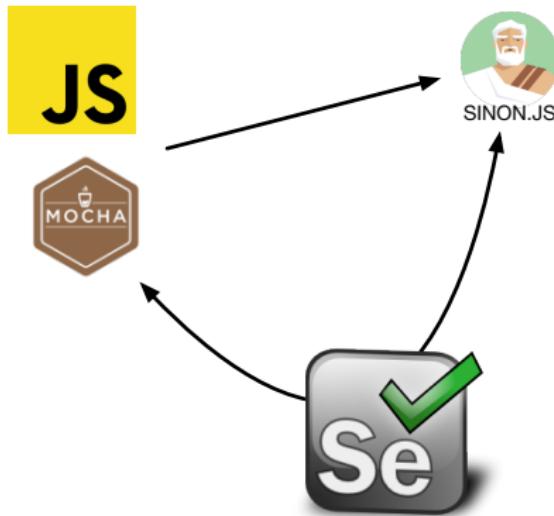
Frontend



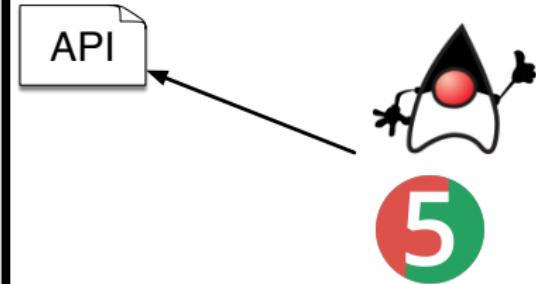
Backend



Frontend

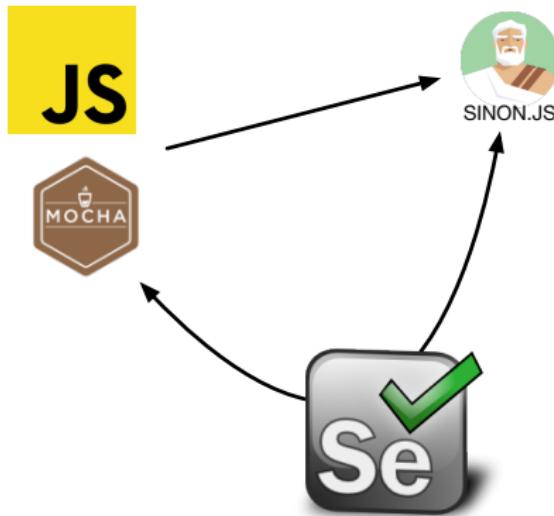


Backend

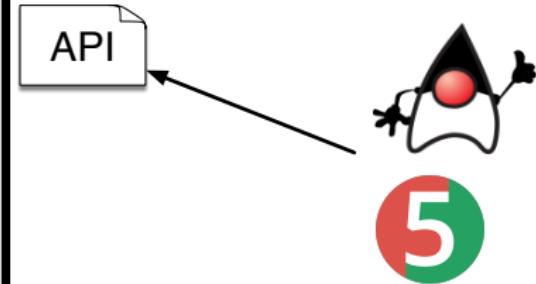


But . . .

Frontend



Backend



Frontend

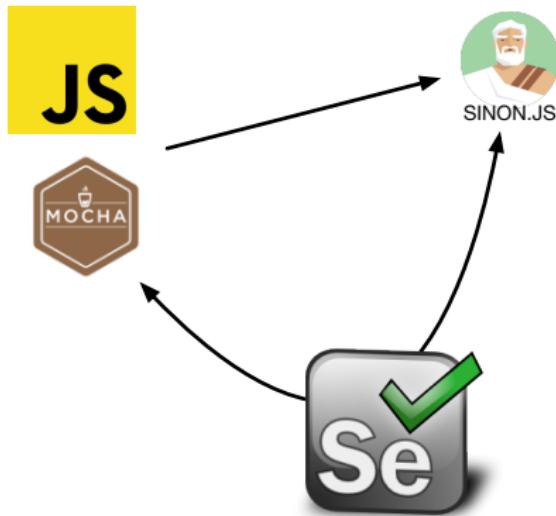
Backend



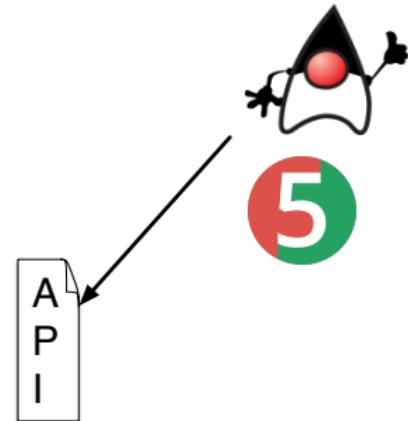
SINON.JS



Frontend

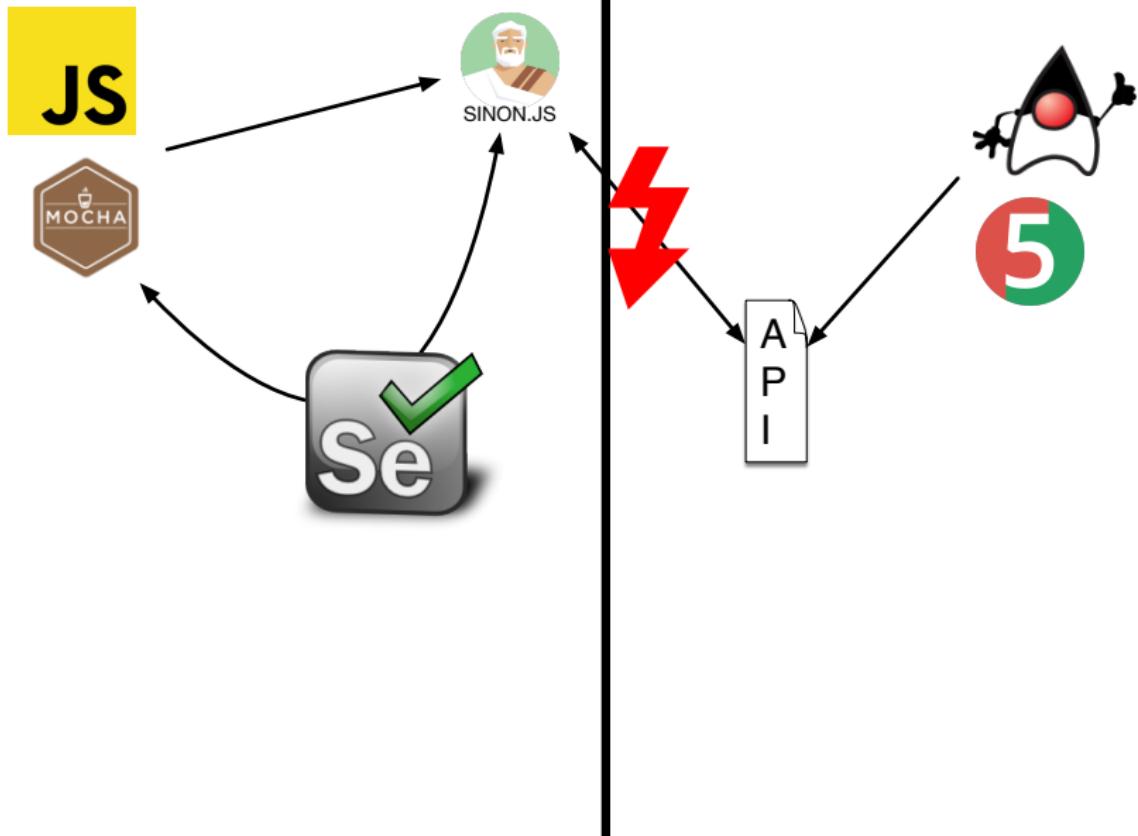


Backend



Frontend

Backend



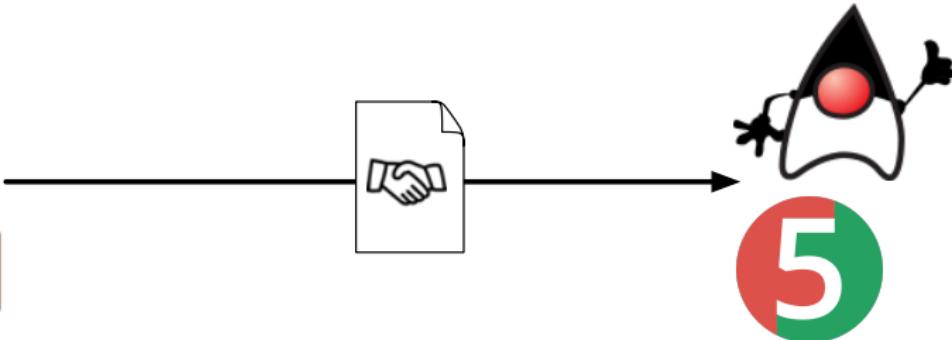
“Industry-Strength” Approach: Consumer-Driven Contract Testing

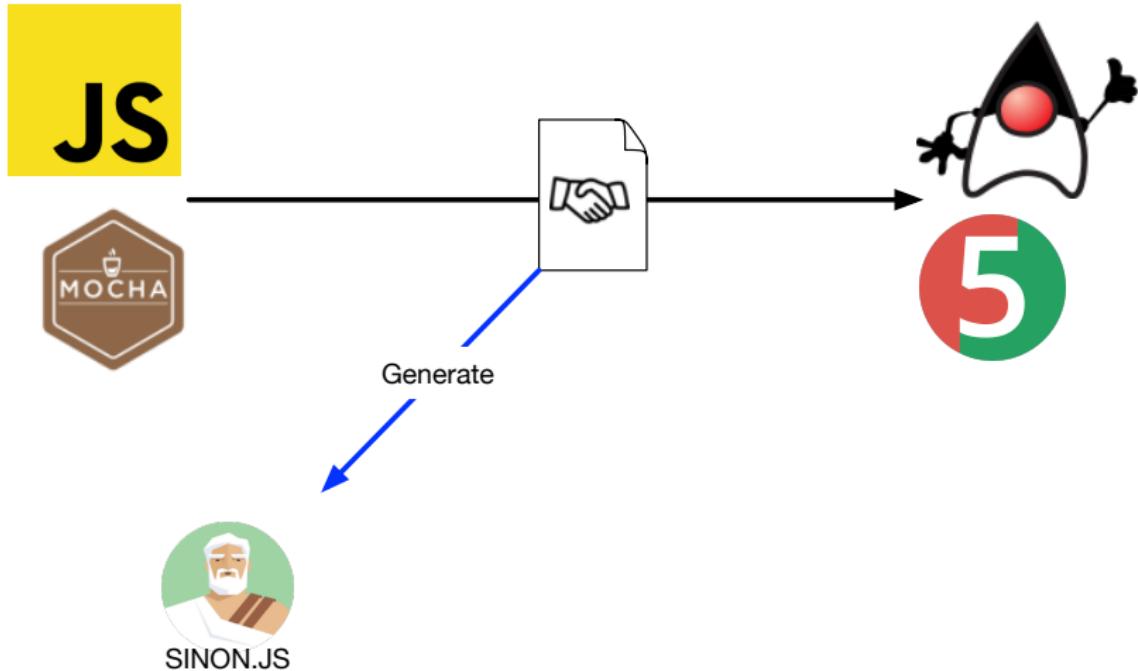
JS

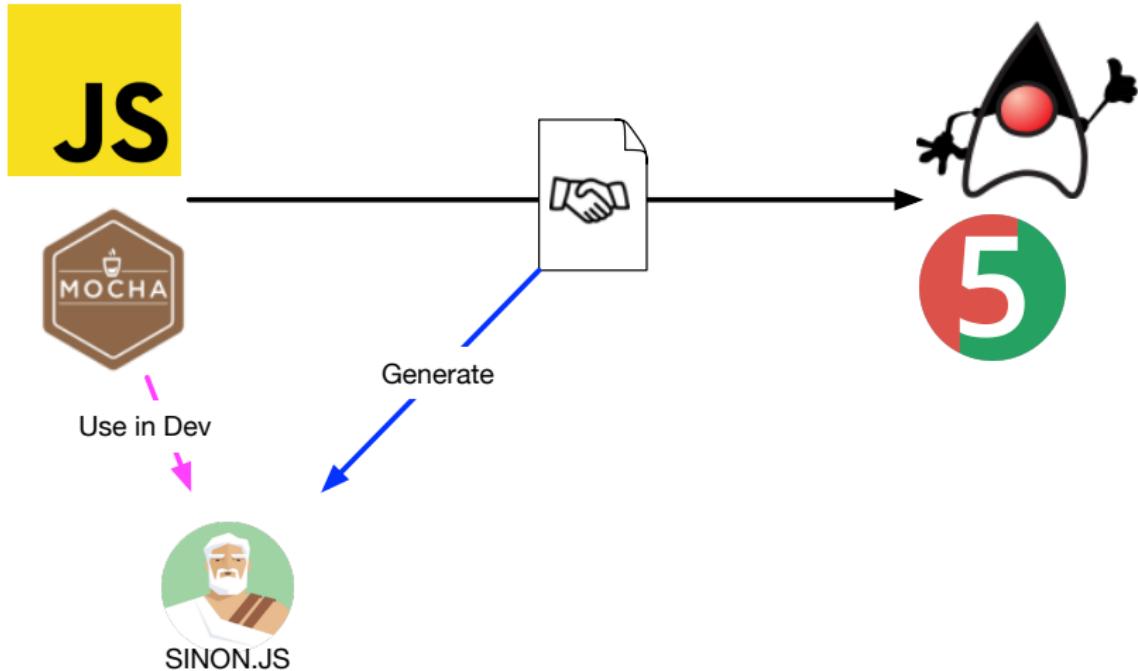


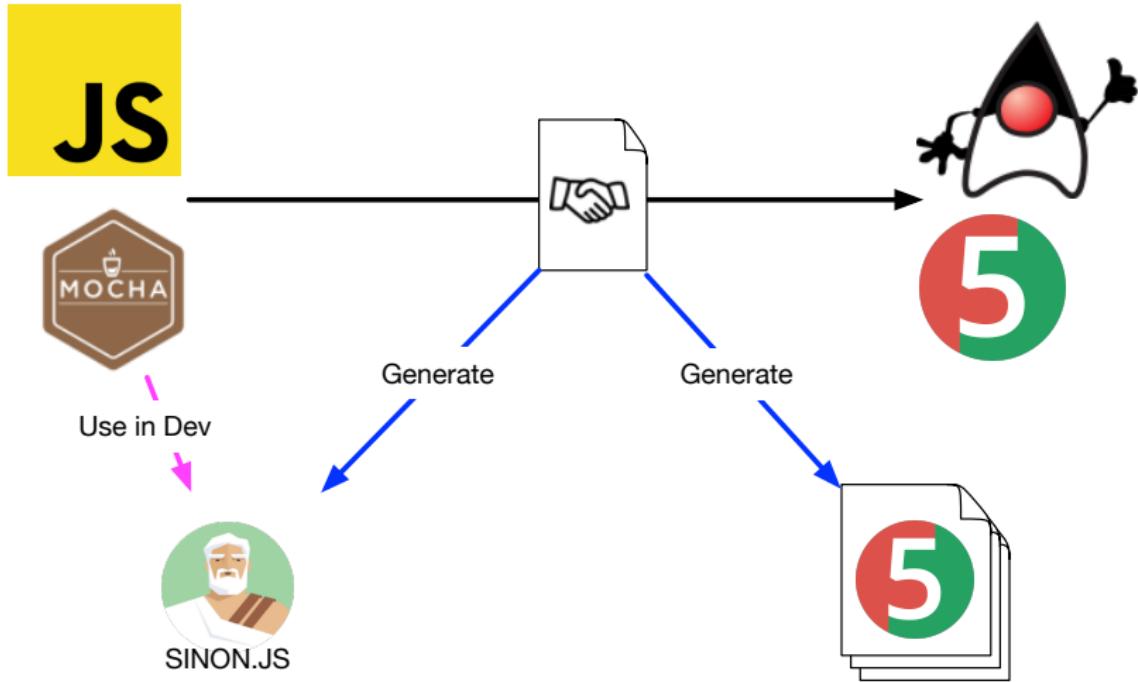
5

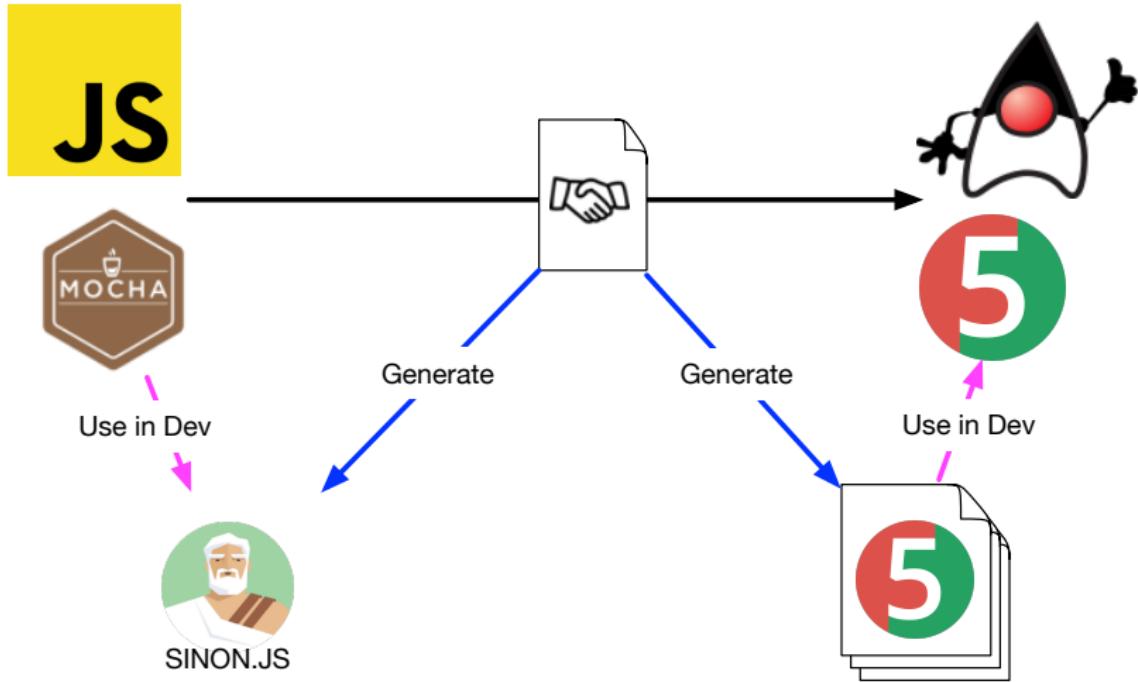
JS

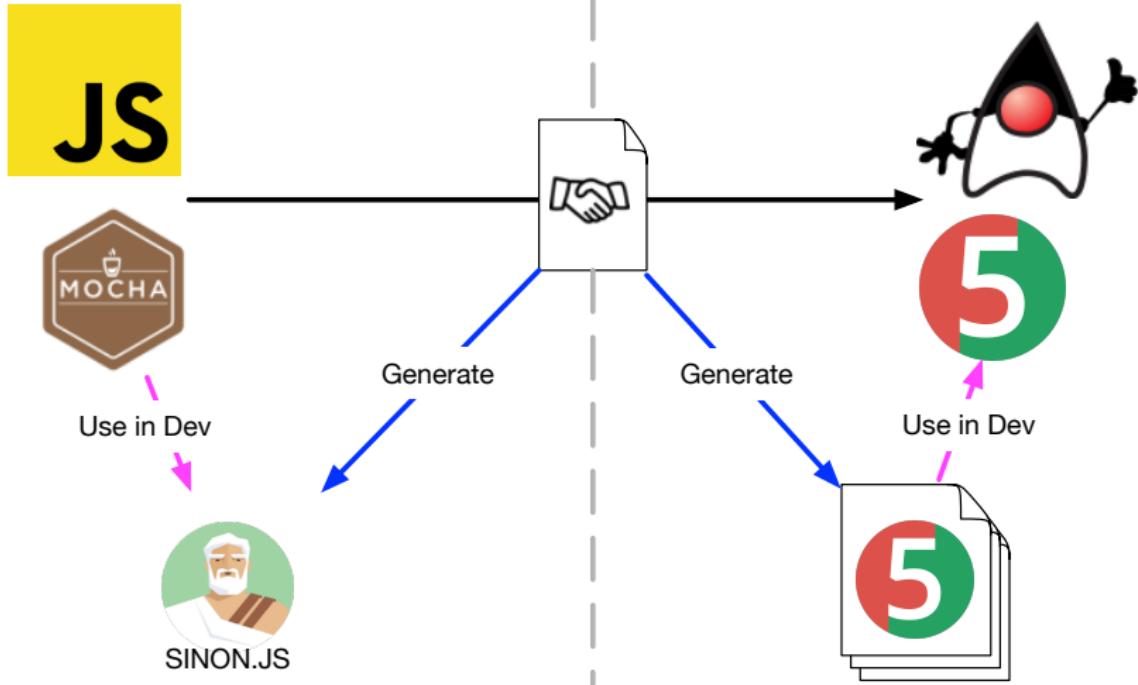












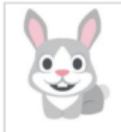
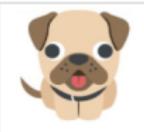
Welcome to our Pet Store

Add a New Pet

Pet name:

Pet price:

Species:



[Add Pet](#)

Pets in our Store



Dodo

[Sell Dodo](#)



Fifi

[Sell Fifi](#)



Minka

[Sell Minka](#)

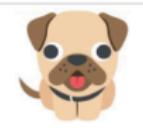
Welcome to our Pet Store

Add a New Pet

Pet name:

Pet price:

Species:



[Add Pet](#)

Pets in our Store

GET



Dodo

[Sell Dodo](#)



Fifi

[Sell Fifi](#)



Minka

[Sell Minka](#)

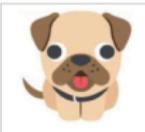
Welcome to our Pet Store

Add a New Pet

Pet name:

Pet price:

Species:



Add Pet



Pets in our Store

POST

Dodo

[Sell Dodo](#)

Fifi

[Sell Fifi](#)

Minka

[Sell Minka](#)

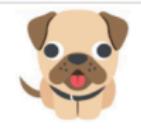
Welcome to our Pet Store

Add a New Pet

Pet name:

Pet price:

Species:



[Add Pet](#)

Pets in our Store



Dodo

[Sell Dodo](#)



Fifi

[Sell Fifi](#)



Minka

[Sell Minka](#)

DELETE

Example: Pet Store Contracts I

```
{  
    "description": "a request for all pets",  
    "providerState": "i have no pets",  
    "request": {  
        "method": "GET",  
        "path": "/pets",  
        "headers": { "Accept": "application/json" }  
    },  
    "response": {  
        "status": 200,  
        "headers": { "Content-Type": "application/json" },  
        "body": {  
            "tag": "Pets",  
            "pets": []  
        }  
    }  
}
```

Situation

- ▶ GET-Request

Situation

- ▶ GET-Request
- ▶ Does not depend on state

Situation

- ▶ GET-Request
- ▶ Does not depend on state
- ▶ Easy to handle with CDCT

Questions

- ▶ Completeness

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?

Example: Pet Store Contracts II

```
{  
  "description": "a request for all pets",  
  "providerState": "i have a list of pets",  
  "request": {  
    "method": "GET",  
    "path": "/pets",  
    "headers": { "Accept": "application/json" }  
  },  
  "response": {  
    "status": 200,  
    "headers": { "Content-Type": "application/json" },  
    "body": {  
      "tag": "Pets",  
      "pets": [  
        { "petName": "Fifi", "petType": "Dog" },  
        { "petName": "Minki", "petType": "Cat" }  
      ]  
    }  
  }  
}
```

Situation

- ▶ GET-Request that depends on state

Situation

- ▶ GET-Request that depends on state
- ▶ POST-/PUT-/DELETE-Requests

Situation

- ▶ GET-Request that depends on state
- ▶ POST-/PUT-/DELETE-Requests
- ▶ Difficult to handle with CDCT

Situation

- ▶ GET-Request that depends on state
- ▶ POST-/PUT-/DELETE-Requests
- ▶ Difficult to handle with CDCT
- ▶ Backend state needs to be established somehow

Situation

- ▶ GET-Request that depends on state
- ▶ POST-/PUT-/DELETE-Requests
- ▶ Difficult to handle with CDCT
- ▶ Backend state needs to be established somehow
- ▶ State checks need to be established somehow

Questions

- ▶ Completeness

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ Maintenance

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?

Questions

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?

We need the Functional Essence!



Sounds cool, but. . .

Sounds cool, but. . .

. . .what does that mean?

Functional Essence

- ▶ Fake Server

Functional Essence

- ▶ Fake Server
- ▶ Lightweight Implementation

Functional Essence

- ▶ Fake Server
- ▶ Lightweight Implementation
- ▶ Model

Functional Essence

- ▶ Fake Server
- ▶ Lightweight Implementation
- ▶ Model
- ▶ Rapid Prototype

Functional Essence

- ▶ Fake Server
- ▶ Lightweight Implementation
- ▶ Model
- ▶ Rapid Prototype
- ▶ Minimal Viable Product

Functional Essence Serves Many Purposes

- ▶ To learn about the domain

Functional Essence Serves Many Purposes

- ▶ To learn about the domain
- ▶ To discuss with domain experts

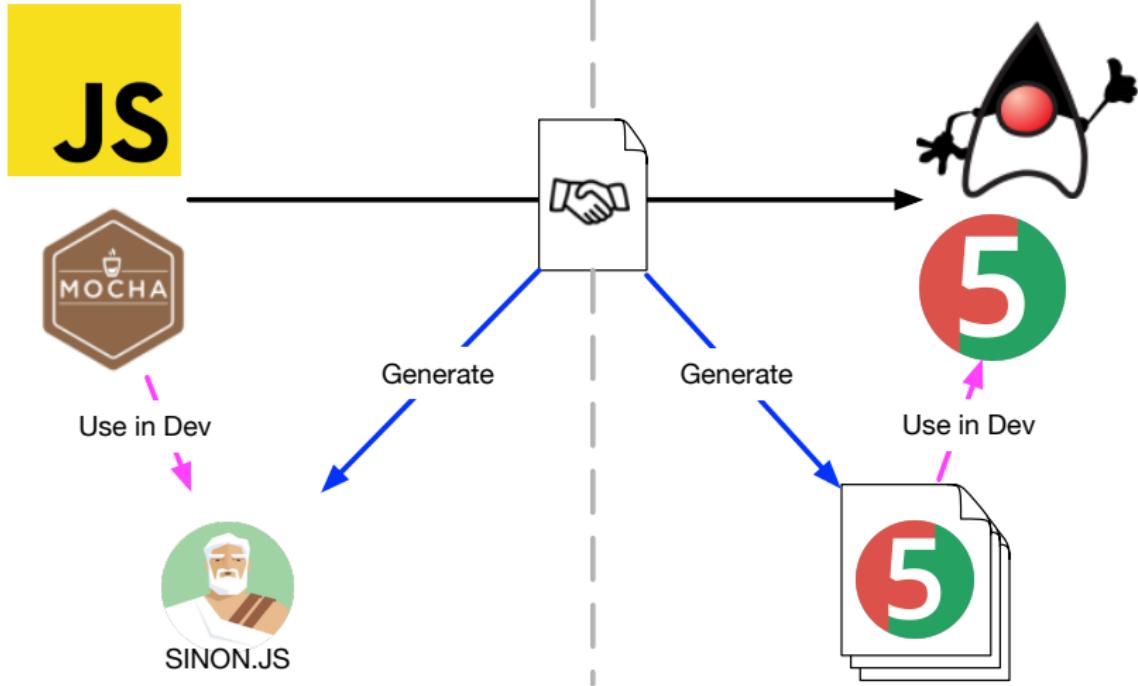
Functional Essence Serves Many Purposes

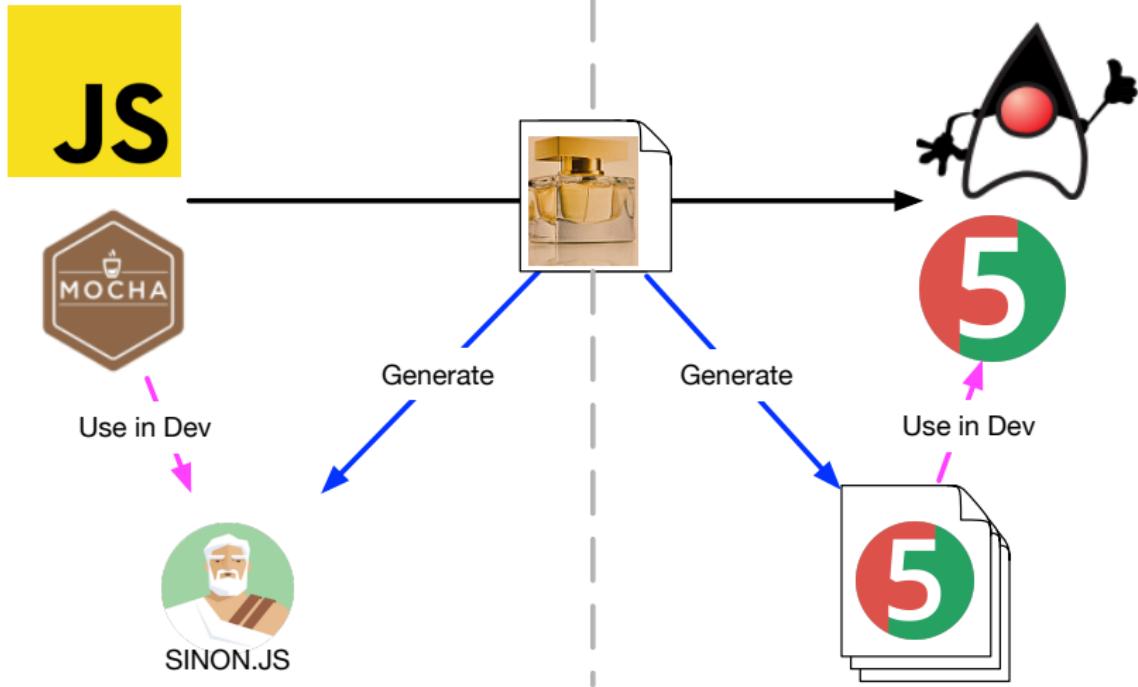
- ▶ To learn about the domain
- ▶ To discuss with domain experts
- ▶ To validate assumptions at an early stage

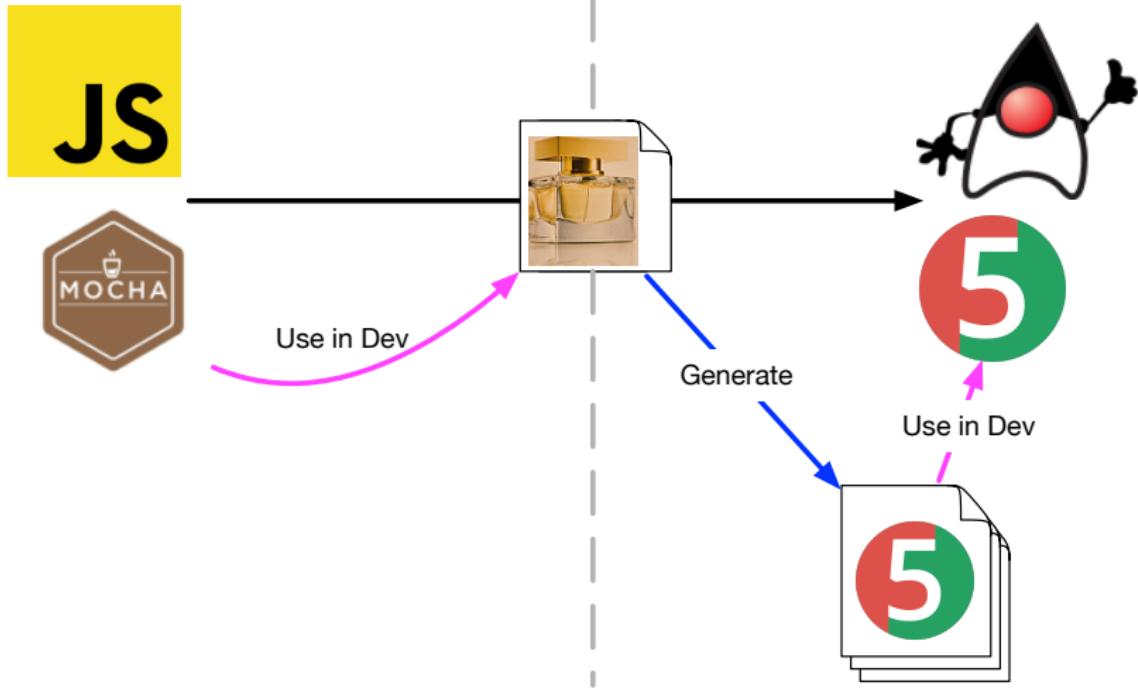
Functional Essence Serves Many Purposes

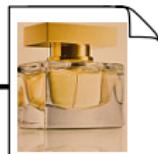
- ▶ To learn about the domain
- ▶ To discuss with domain experts
- ▶ To validate assumptions at an early stage

- ▶ To check the real implementation:









Use in Dev

Compare

Sounds cool, but. . .

Sounds cool, but...

...how can I build this?

How to Build a Functional Essence

- ▶ Implement the domain logic

How to Build a Functional Essence

- ▶ Implement the domain logic
- ▶ In the simplest possible way

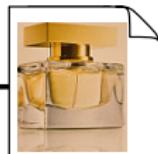
How to Build a Functional Essence

- ▶ Implement the domain logic
- ▶ In the simplest possible way
- ▶ In an arbitrary language

How to Build a Functional Essence

- ▶ Implement the domain logic
- ▶ In the simplest possible way
- ▶ In an arbitrary language





Use in Dev

Compare

The Pets Essence

```
class Pets {  
    constructor(){  
        this._pets = [];  
    }  
  
    getPets() {  
        return petSorter.sortPets(this._pets);  
    }  
  
    addPet(pet) {  
        this._pets.push(pet);  
        return 'Pet successfully added.';  
    }  
  
    removePet(pet) {  
        this._pets = this._pets.filter(  
            p => p.petName !== pet.petName || p.petType !== pet.petType);  
        return 'Pet successfully removed.';  
    }  
}
```

The Overall Essence

```
class Essence {  
    constructor() {  
        this._pets = new Pets();  
        this._somethingElse = new SomethingElse();  
    }  
  
    pets() {  
        return this._pets;  
    }  
  
    somethingElse() {  
        return this._somethingElse;  
    }  
}
```

The Essence App

```
let essence = new Essence();

router.get('/pets', (req, res) => {
  const pets = essence.pets().getPets();
  res.json({tag: 'Pets', pets});
});

router.post('/pets', (req, res) => {
  const message = essence.pets().addPet({ petName: req.body.petName,
    petPrice: req.body.petPrice, petType: req.body.petType });
  res.json({message});
});

router.delete('/pets', (req, res) => {
  const message = essence.pets().removePet({ petName: req.body.petName,
    petPrice: req.body.petPrice, petType: req.body.petType });
  res.json({message});
});
```

Important Addition

```
router.delete('/reset', (req, res) => {
  essence = new Essence();
  res.json({message: 'All pets successfully removed.'});
});
```

Important Addition

```
router.delete('/reset', (req, res) => {
  essence = new Essence();
  res.json({message: 'All pets successfully removed.'});
});
```

Also for the real backend!

Isn't That The Same Code As The Backend?

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:
- ▶ Can be in another language

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:
- ▶ Can be in another language
- ▶ Can use simpler datatypes (e.g. List vs. HashMap)

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:
- ▶ Can be in another language
- ▶ Can use simpler datatypes (e.g. List vs. HashMap)
- ▶ No performance-tuning (e.g. slower but simpler algorithms)

Isn't That The Same Code As The Backend?

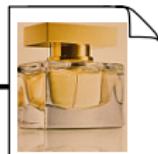
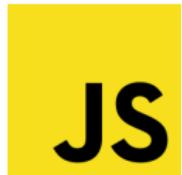
- ▶ Maybe... but usually not:
- ▶ Can be in another language
- ▶ Can use simpler datatypes (e.g. List vs. HashMap)
- ▶ No performance-tuning (e.g. slower but simpler algorithms)
- ▶ No technical overhead (e.g. database)

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:
- ▶ Can be in another language
- ▶ Can use simpler datatypes (e.g. List vs. HashMap)
- ▶ No performance-tuning (e.g. slower but simpler algorithms)
- ▶ No technical overhead (e.g. database)
- ▶ Some parts can be left out

Isn't That The Same Code As The Backend?

- ▶ Maybe... but usually not:
- ▶ Can be in another language
- ▶ Can use simpler datatypes (e.g. List vs. HashMap)
- ▶ No performance-tuning (e.g. slower but simpler algorithms)
- ▶ No technical overhead (e.g. database)
- ▶ Some parts can be left out
- ▶ ...



Use in Dev

Compare

How does the Comparison work?





Make same Change





Ask same Question





Get same Answer



How can we Implement the
Comparison?

Detour: Quick Check / Property-Based Testing

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

```
prop_RevRev :: [Int] -> Bool  
prop_RevRev xs = reverse (reverse xs) == xs
```

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

```
prop_RevRev :: [Int] -> Bool  
prop_RevRev xs = reverse (reverse xs) == xs
```

```
*Main> quickCheck prop_RevRev  
+++ OK, passed 100 tests.
```

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

```
prop_RevRev :: [Int] -> Bool  
prop_RevRev xs = reverse (reverse xs) == xs
```

```
*Main> quickCheck prop_RevRev  
+++ OK, passed 100 tests.
```

```
prop_RevId :: [Int] -> Bool  
prop_RevId xs = reverse xs == xs
```

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

```
prop_RevRev :: [Int] -> Bool  
prop_RevRev xs = reverse (reverse xs) == xs
```

```
*Main> quickCheck prop_RevRev  
+++ OK, passed 100 tests.
```

```
prop_RevId :: [Int] -> Bool  
prop_RevId xs = reverse xs == xs
```

```
*Main> quickCheck prop_RevId  
*** Failed! Falsifiable (after 7 tests and 3 shrinks):  
[0,1]
```

Detour: Quick Check / Property-Based Testing

- ▶ User specifies properties
- ▶ Tool generates examples
- ▶ Checks properties against examples

```
prop_RevRev :: [Int] -> Bool  
prop_RevRev xs = reverse (reverse xs) == xs
```

```
*Main> quickCheck prop_RevRev  
+++ OK, passed 100 tests.
```

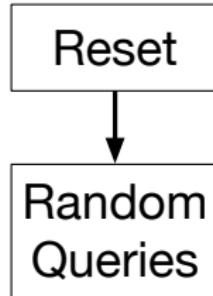
```
prop_RevId :: [Int] -> Bool  
prop_RevId xs = reverse xs == xs
```

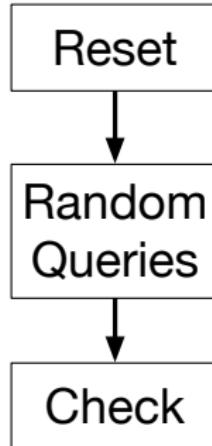
```
*Main> quickCheck prop_RevId  
*** Failed! Falsifiable (after 7 tests and 3 shrinks):  
[0,1]
```

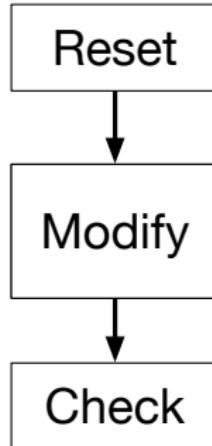
```
*Main> quickCheck prop_RevId  
*** Failed! Falsifiable (after 4 tests and 1 shrink):  
[1,0]
```

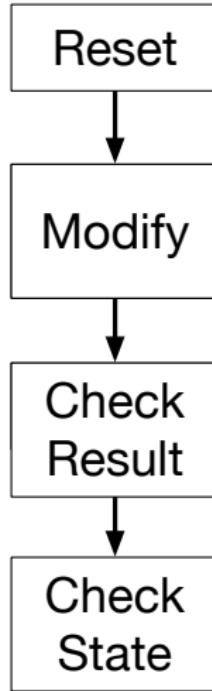
How to Mimic QuickCheck?

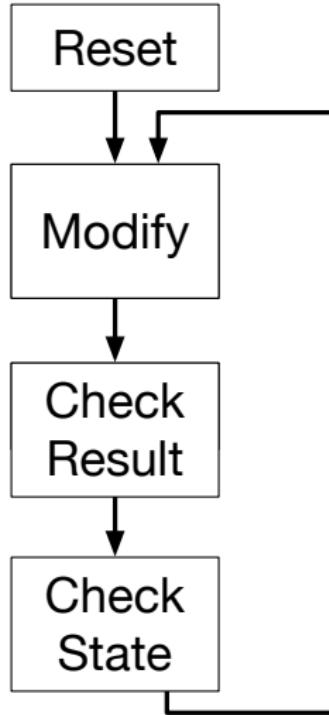
Reset

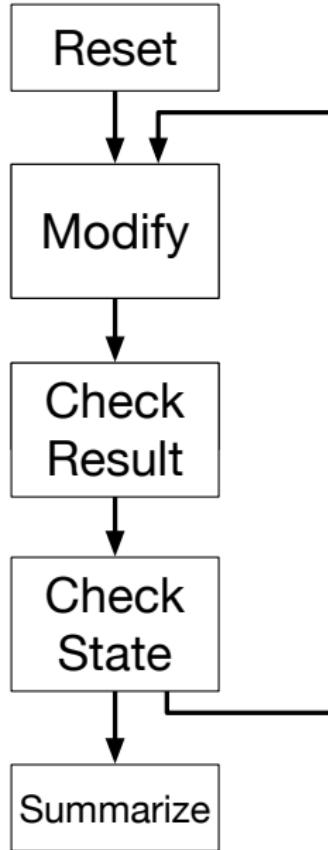


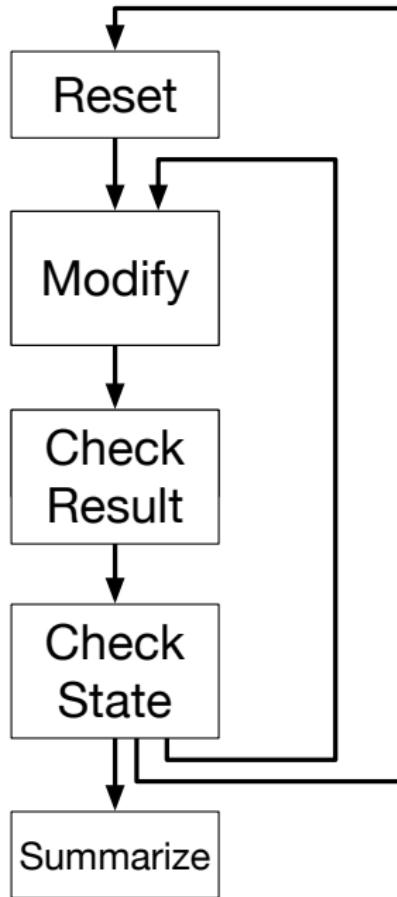






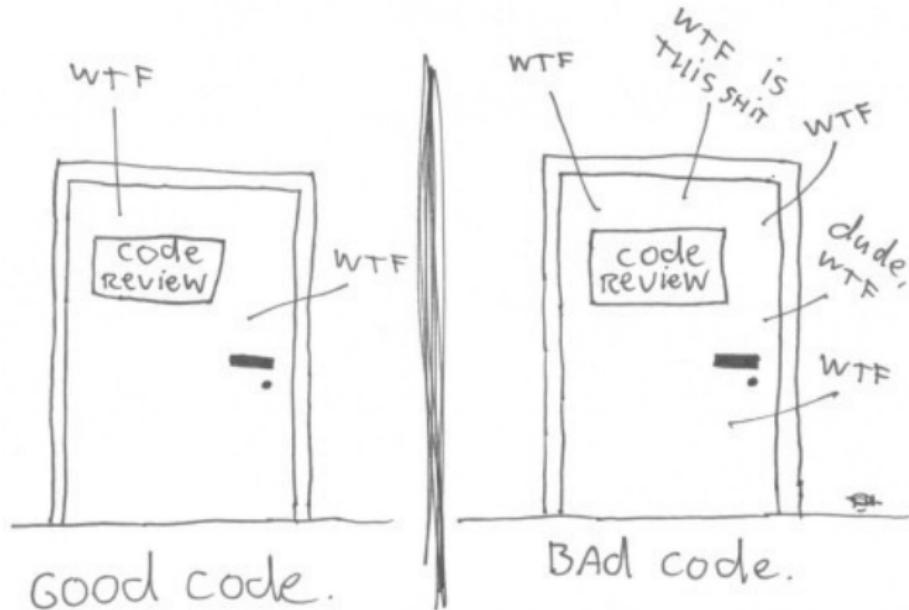






Comparator Implementation

The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



The Main Loop

```
const requests = [resets[0]()];
// initial reset

let count = 0;

while (count < 50) {
    count++;
    requests.push(chooseFrom(modifyingRequests()));
}

async.mapSeries(requests, requestAndCompare, (err, results) => {
    results.map(result => console.log(result));
    if (err) {
        console.log(err);
    }
});
```

Request Generator

```
const resets = [
    // delete all pets
    () => ({url: '/reset', method: 'DELETE'}),
];

const modifyingRequests = [
    // addPet
    () => ({url: '/pets', method: 'POST', json: true, body: randomPet()}),
    // removePet
    () => ({url: '/pets', method: 'DELETE', json: true, body: randomPet()})
];

const comparisons = [
    // getPets
    () => ({url: '/pets', method: 'GET'}),
];
```

Pet Generator

```
const chooseFrom = arr => arr[Math.floor(arr.length * Math.random())];

const possibleNames = ['Minka', 'Fifi', 'Pucki'];
const possibleSpecies = ['Cat', 'Dog', 'Canary', 'Rabbit', 'Fish'];

const name = () => chooseFrom(possibleNames);
const price = () => Math.floor(150 * Math.random());
const species = () => chooseFrom(possibleSpecies);

const randomPet = () =>
    {petName: name(), petPrice: price(), petType: species()};
```

Revisiting the Main Loop

```
const requests = [resets[0]()];
// initial reset

let count = 0;

while (count < 50) {
    count++;
    requests.push(chooseFrom(modifyingRequests()));
}

async.mapSeries(requests, requestAndCompare, (err, results) => {
    results.map(result => console.log(result));
    if (err) {
        console.log(err);
    }
});
```

Request-Compare-Loop

```
const requestAndCompare = (request, mainCallback) => {

    console.log('Running the modification request:');

    runRequest(request, (err, result) => {
        const backendString = JSON.stringify(result.backend);
        const essenceString = JSON.stringify(result.essence);

        if (backendString !== essenceString) {
            mainCallback('Backend and Essence responses differ! Backend: '
                + backendString + ' - Essence: ' + essenceString);

        } else {

            console.log('Comparing all data:');
            compareEverything(mainCallback);
        }
    });
};
```

Query Submission

```
const backend = {baseURL: 'http://localhost:9090'};
const essence = {baseURL: 'http://localhost:8080'};

const merge = (req, server) =>
  Object.assign({}, req, {url: server.baseURL + req.url});

const requestFunctionFor = (req, server) =>
  callback => request(merge(req, server),
    (err, response) => callback(err, response.body));

const runRequest = (req, callback) => {
  console.log('Now checking:', req);
  async.parallel({
    backend: requestFunctionFor(req, backend),
    essence: requestFunctionFor(req, essence)
  }, callback);
};
```

Comparisons

```
function compareEverything(mainCallback) {
    async.map(comparisons, (itemFunc, callback) =>
        runRequest(itemFunc(), (err, res) => {
            if (res.backend === res.essence) {
                callback(null, null); // no differences
            } else {
                const formatDiff = formatter.formatDiff({
                    backend: JSON.parse(res.backend),
                    essence: JSON.parse(res.essence)
                });
                callback(null, formatDiff);
            }
        }),
        (err, results) => {
            const nonmatching = results.filter(x => x);
            mainCallback(nonmatching.length
                ? 'Backend and Essence differ in their data' : null);
        });
}
```

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves
 - ▶ We are responsible for completeness

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves
 - ▶ We are responsible for completeness
- ▶ Beyond CDCT:

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves
 - ▶ We are responsible for completeness
- ▶ Beyond CDCT:
 - ▶ QuickCheck approach: We don't write tests

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves
 - ▶ We are responsible for completeness
- ▶ Beyond CDCT:
 - ▶ QuickCheck approach: We don't write tests
 - ▶ We only specify the possible routes

Questions – and Answers

- ▶ Completeness
 - ▶ Did we really capture all requests (+ responses) in our contract?
 - ▶ All possible combinations with different backend states?
- ▶ CDCT:
 - ▶ We must write all tests ourselves
 - ▶ We are responsible for completeness
- ▶ Beyond CDCT:
 - ▶ QuickCheck approach: We don't write tests
 - ▶ We only specify the possible routes
 - ▶ Completeness is established over time

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test
 - ▶ We are responsible for the state's correctness

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test
 - ▶ We are responsible for the state's correctness
- ▶ Beyond CDCT:

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test
 - ▶ We are responsible for the state's correctness
- ▶ Beyond CDCT:
 - ▶ The state is established through API calls

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test
 - ▶ We are responsible for the state's correctness
- ▶ Beyond CDCT:
 - ▶ The state is established through API calls
 - ▶ The state is always reachable and correct*

Questions – and Answers

- ▶ State Validity
 - ▶ What are valid states in our stub?
 - ▶ Did we always establish a valid state in our stub?
 - ▶ How do we establish them (technically) before the request?
 - ▶ How do we validate them after the (modifying) request?
- ▶ CDCT:
 - ▶ We establish the state ourselves for each test
 - ▶ We are responsible for the state's correctness
- ▶ Beyond CDCT:
 - ▶ The state is established through API calls
 - ▶ The state is always reachable and correct*

*Assuming no coding errors!

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes
 - ▶ Code coverage helps identify missing routes

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes
 - ▶ Code coverage helps identify missing routes
 - ▶ Missing routes in Essence will be discovered because of errors

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes
 - ▶ Code coverage helps identify missing routes
 - ▶ Missing routes in Essence will be discovered because of errors
 - ▶ Missing logic in the Essence will be discovered

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes
 - ▶ Code coverage helps identify missing routes
 - ▶ Missing routes in Essence will be discovered because of errors
 - ▶ Missing logic in the Essence will be discovered
 - ▶ initially because of unexpected behaviour when prototyping

Questions – and Answers

- ▶ Maintenance
 - ▶ How can we keep track of our contracts and avoid redundancies?
 - ▶ How can we effectively maintain the contracts in case of changes?
- ▶ CDCT:
 - ▶ As difficult as maintaining normal tests
 - ▶ Code coverage helps identify uncovered areas
- ▶ Beyond CDCT:
 - ▶ We need to make sure that the comparator knows all routes
 - ▶ Code coverage helps identify missing routes
 - ▶ Missing routes in Essence will be discovered because of errors
 - ▶ Missing logic in the Essence will be discovered
 - ▶ initially because of unexpected behaviour when prototyping
 - ▶ later because of comparison errors

A Word of Warning

- ▶ This is not an easy solution. The world remains complicated.

A Word of Warning

- ▶ This is not an easy solution. The world remains complicated.
- ▶ This is not a golden hammer. Don't apply it everywhere.

A Word of Warning

- ▶ This is not an easy solution. The world remains complicated.
- ▶ This is not a golden hammer. Don't apply it everywhere.
- ▶ This is rather at the top of the test pyramid.

Thank you!

E-Mail info@nicole-rauch.de

Twitter @NicoleRauch

Web <http://www.nicole-rauch.de>



NICOLE RAUCH
software development &
development coaching

EventStorming · Domain-Driven Design
Training · Coaching · Facilitation
Software Craftsmanship
React.js and Redux
Functional Programming

<https://github.com/NicoleRauch/BeyondCDCT>

Credits

Essence: Mark Morgan - Perfume

<https://www.flickr.com/photos/markmorgantrinidad/5959461561>

WTF's per minute: Tom Holwerda

http://www.osnews.com/story/19266/WTFs_m