2.– 5. September 2013
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Aber eben lief das doch noch!

HTML5 Testing mit Jasmine BDD

Alexander Schwartz

msg systems ag

Was ich vorhabe

1. Worum es geht
2. Fachlicher Einstieg in das Beispiel
3. Testen mit Jasmine
4. Verwendung von Jasmine mit Sinon.JS
5. Ausführung im Browser und in Continuous Integration
6. Debugging von JavaScript-Tests
7. Zusammenfassung

Worum es hier geht

Automatisiertes Testen der HTML+JavaScript Client-Plattform

Browser als Client-Plattform

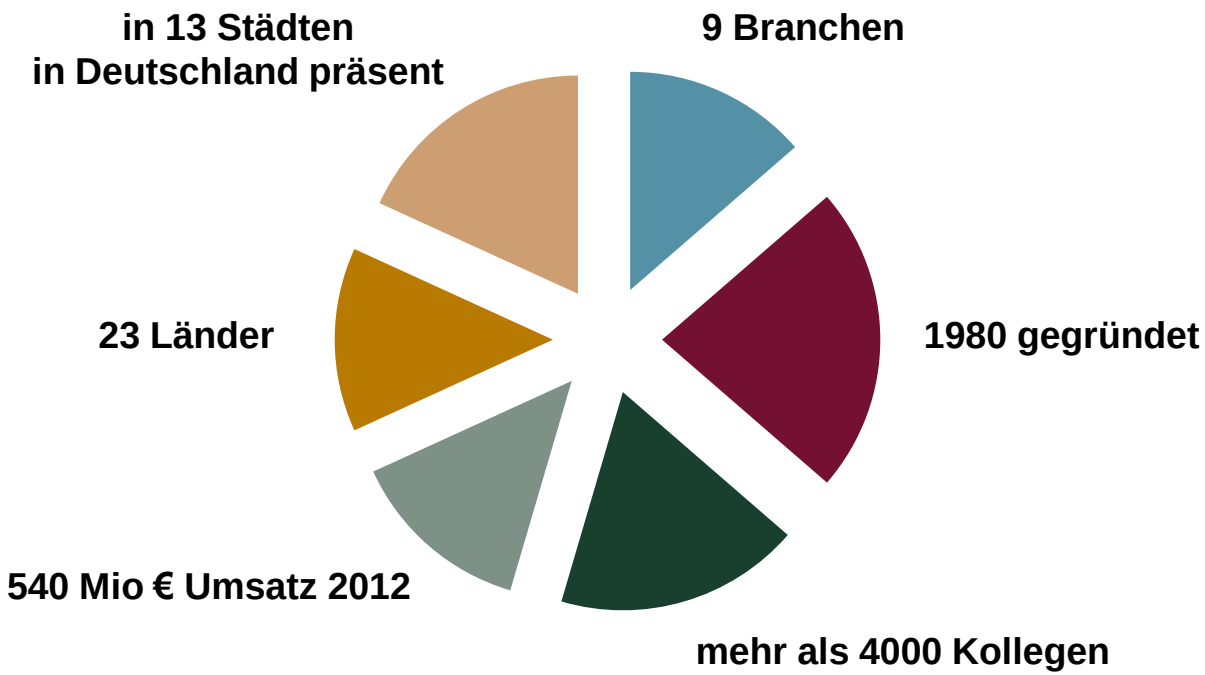
- HTML als Seitenbeschreibung
- CSS für ansprechendes (responsive) Design
- Widget z. B. via jQueryUI + Plugins
- Clientlogik via JavaScript
- Strukturierung mit Javascript MV*-Frameworks (z. B. KnockoutJS)
- Modularisierung und Nachladen von Resourcen (z. B. requireJS)

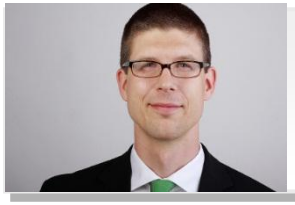
REST für Backend Services

- Kommunikation via REST/JSON ideal für JavaScript
- Verschiedene Implementierungen möglich
- JAX-RS 1.0 / JEE 6

Mein Sponsor und Arbeitgeber

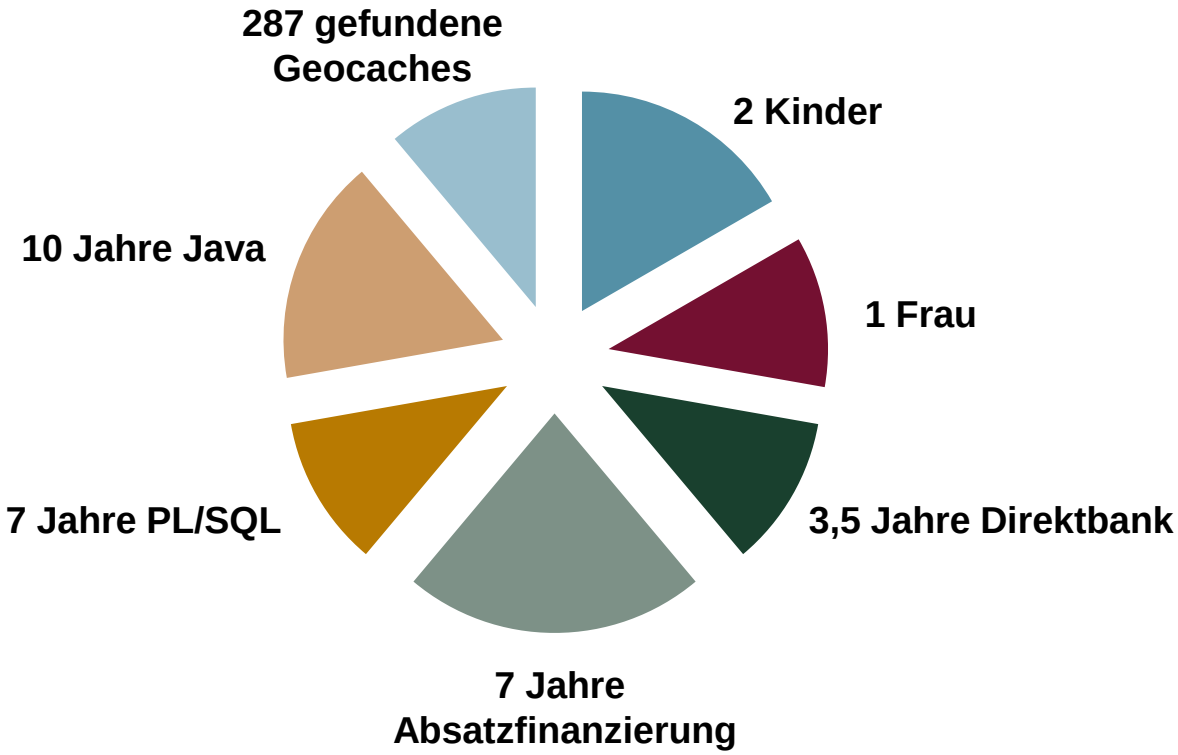
msg systems ag





Alexander Schwartz

Lead IT Consultant im GB Travel und Logistics



Die heutige Aufgabe

Online-Datenbank für Schiffsichtungen

User Story:

„Nutzer möchte die Sichtung eines Schiffes erfassen, um später sehen zu können, ob auch andere dieses Schiff gesehen haben.

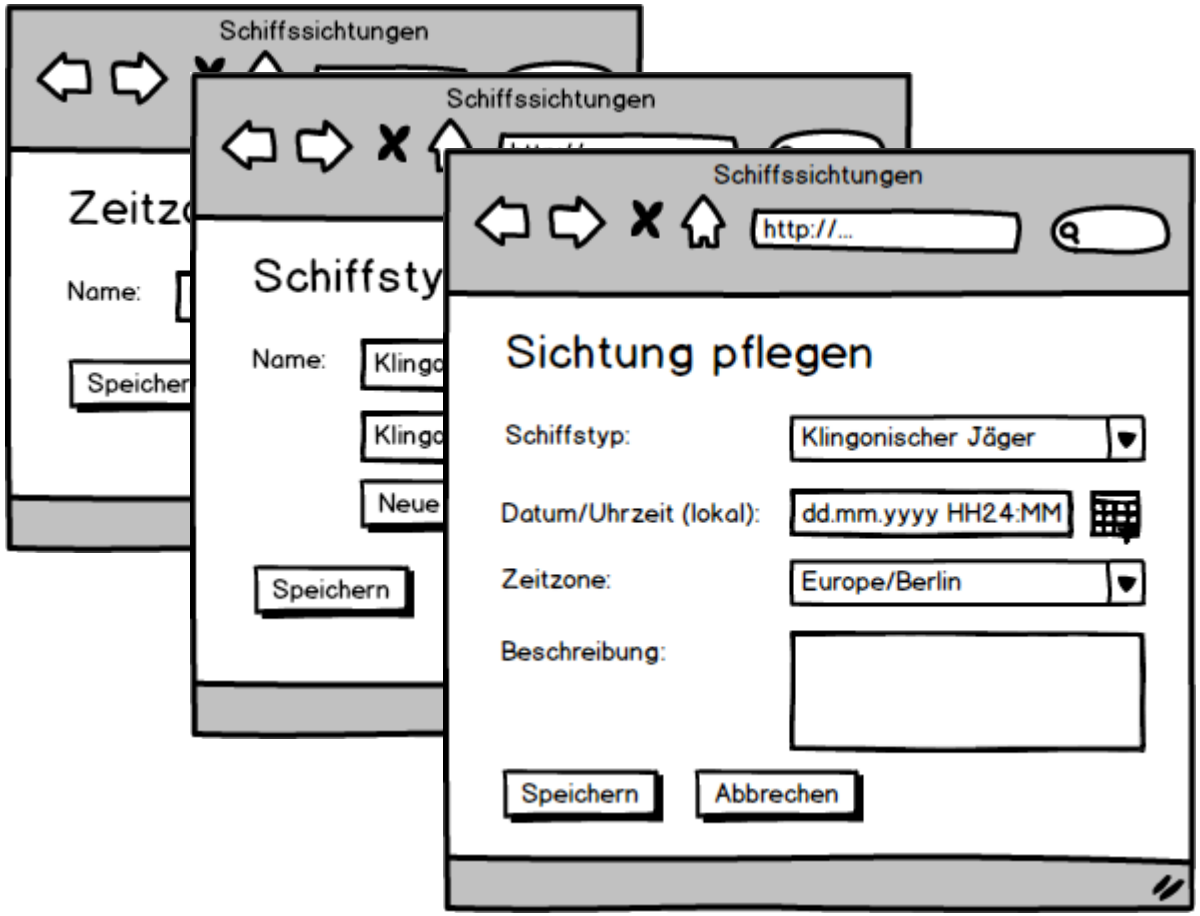
Hierzu erfasst er Schifftstyp, Datum, Zeitzone und eine Notiz.“

Garantiert kein Bezug zu einem Kundenprojekt!

https://github.com/ahus1/rest-samples

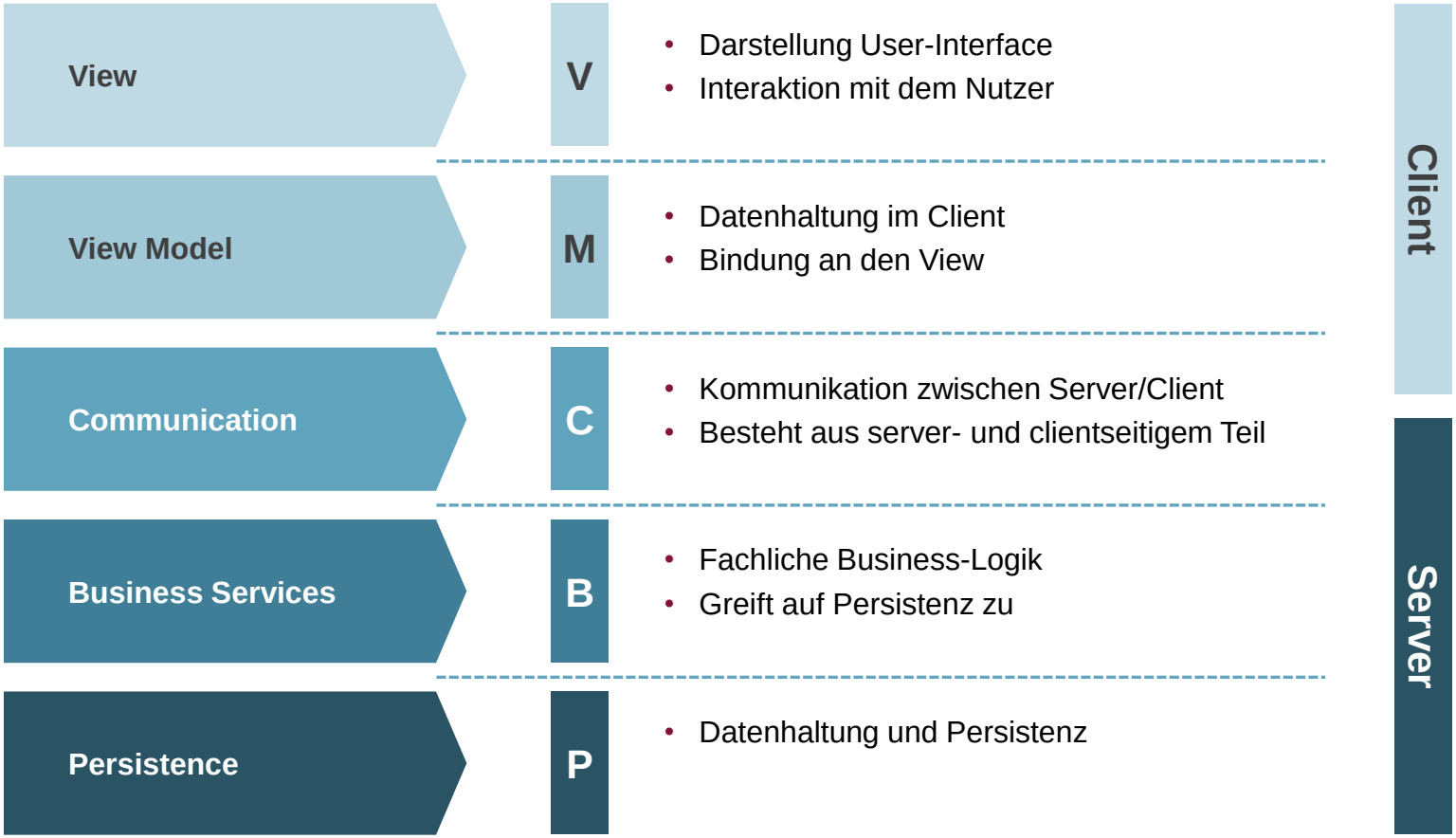
Wie´s für den Kunden aussieht

Ein erster Blick auf die Anwendung

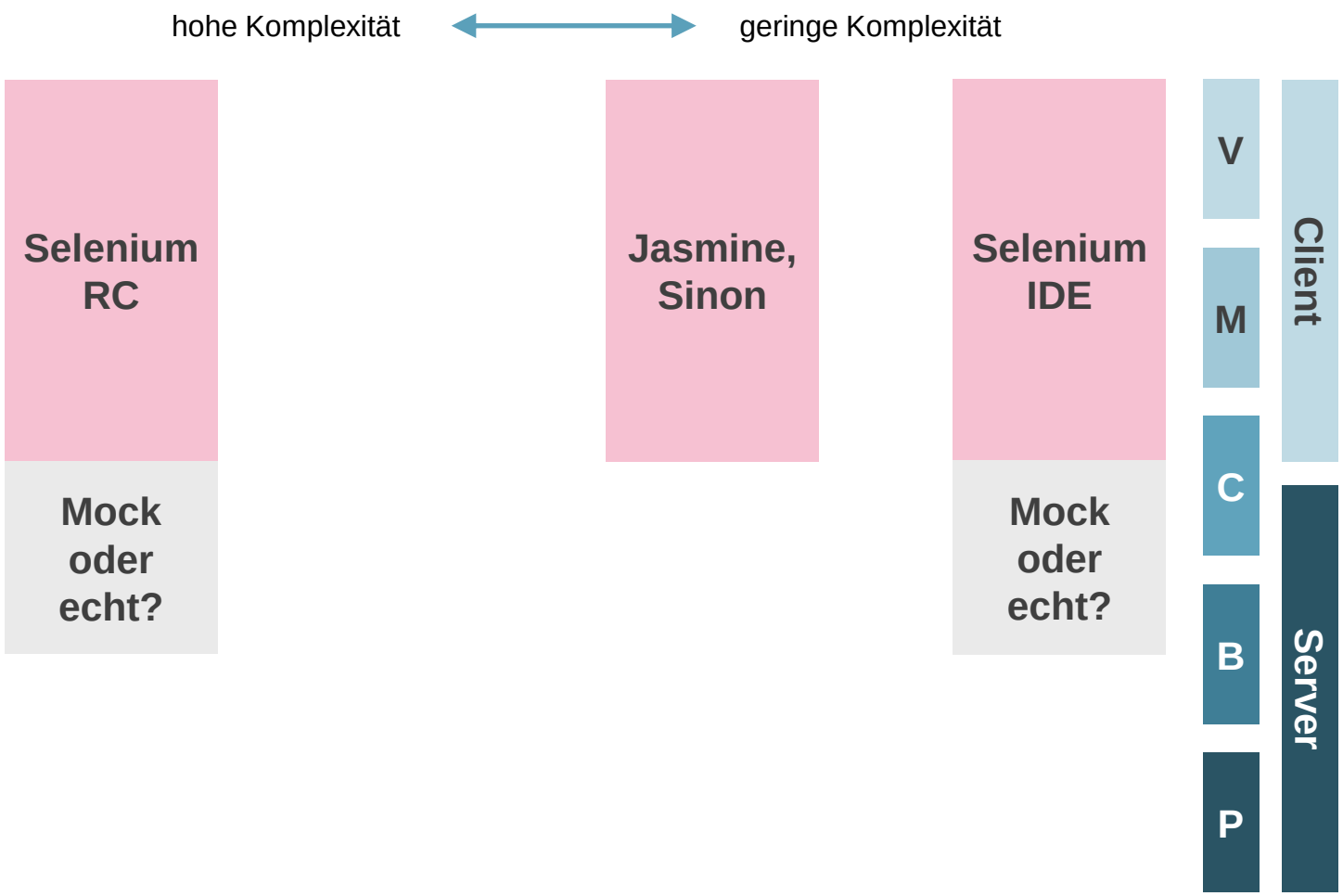


msg systems

Mit JavaScript den kompletten Client-Stack testen



Verschiedene Testframeworks decken verschiedene Bereiche ab



Jasmine Steckbrief

Jasmine ist ein JavaScript Testframework

Jasmine

http://pivotal.github.io/jasmine/

Test Suite

Test Case

Test Code mit Matcher

```
describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
```

Behaviour Driven Development (BDD) Testing:
Erstelle eine strukturierte Beschreibung des beobachtbaren Verhaltens der Software

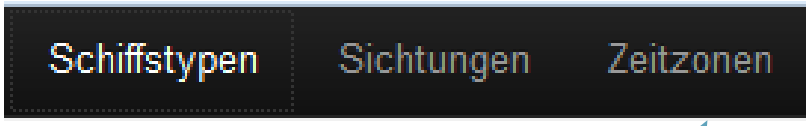
Test – Model Only

Prüfen, ob das Model sich wie beschrieben verhält

Umgesetzt mit KnockoutJS

```
// menu.js
 var Menu = function() {
    // Data
    var self = this;
    self.folders = ko.observableArray([ {
      name : 'Schiffstypen',
      link : '#/vessel/main'
    }, {
      name : 'Sichtungen',
      link : '#/sighting/main'
    }, {
      name : 'Zeitzonen',
      link : '#/timezone/main'
    } ]);
    self.folder = ko.observable();
  …
```

```
<!-- index.html -->
<ul class="nav" data-bind="foreach: menu.folders">
<li data-bind="css: { active: $data.link ==
$parent.menu.folder() }, attr: {id: $data.link} "><a
data-bind="text: $data.name, attr: {href:
$data.link}"></a></li>
</ul>
```



Ein Menü anzeigen – Trennung von View und View Model

```
describe("Menu", function() {

  it("has three menu items", function() {
    expect(menu.folders().length).toBe(3);
  });

  it("has first item named 'Schiffstypen'", function() {
    expect(menu.folders()[0].name()).toBe("Schiffstypen");
  });

  it("can be switched to English", function() {
    // when
    runs(function() {
      menu.language("en");
    });

    // then
    waitsFor(function() {
      return menu.folders()[0].name();
    }, "translation to change to 'Vessels'", 750);
  });
…
```

Interaktion des Models mit dem Browser

```
describe("Link Interaction", function() {

    beforeEach(function() {
      // reset to start hash
      hasher.setHash("");
    });

    it("knowns the active entry", function() {
      // given
      var firstEntry = menu.folders()[0];
      var secondEntry = menu.folders()[1];
      expect(menu.isCurrentSubfolder(firstEntry.link)).toBe(false);

      // when
      hasher.setHash("vessel/main");

      // then
      expect(menu.isCurrentSubfolder(firstEntry.link)).toBeTruthy();
      expect(menu.isCurrentSubfolder(secondEntry.link)).toBeFalsy();
    });

    afterEach(function() {
      // reset to start hash
      hasher.setHash("");
    });
});
```

Interaktion mit dem View

- Zugriff via jQuery auf die aktuelle Seite
- Werte kommen hier noch aus dem Backend

```
describe("Manage Timezones Real Backend", function() {

  it("shows a list of two timezones at the start",
     function() {
       expect($("#timezoneList")).toBeVisible();
       expect($("#timezoneList > tbody > tr")[0]).toContainHtml(
          "Europe/Berlin");
     });

  it("has the first menu item highlighted at the start", function() {
    expect($("#subfolder0")).toHaveClass('active');
  });
…
```

Demo & Live Coding

Interaktion mit dem View

- Benutzerinteraktionen werden via jQuery ausgelöst

```
beforeEach(function() {

    expect($("#timezoneList > tbody > tr").length).toBe(1);

    expect($("a[name=edit]").length).toBe(1);

    $("a[name=edit]").first().click();

    waitsFor(function() {
      return $("#timezoneName").is(":visible");
    }, "the time zone name should appear", 5000);

  });

it("show a timezone on editing", function() {

    expect($("#timezoneName")).toBeVisible();
    expect($("#timezoneName")).toHaveValue("Europe/Berlin");

  });
```

Mocking des Backends

Mit Sinon.JS das REST-Backend simulieren

- Abfangen von JavaScript-Zugriffen auf den Server werden abgefangen
- Antworten können simuliert werden

```
// create a fake server with some responses
server = sinon.fakeServer.create();

var timezoneList = ko.toJSON(getJSONFixture('timezoneList.json'));

server.respondWith("GET", "rest/timezone", [ 200, {
  "Content-Type" : "application/json"
}, timezoneList ]);

server.respondWith("DELETE", "rest/timezone/1", [ 204, null, "" ]);
```

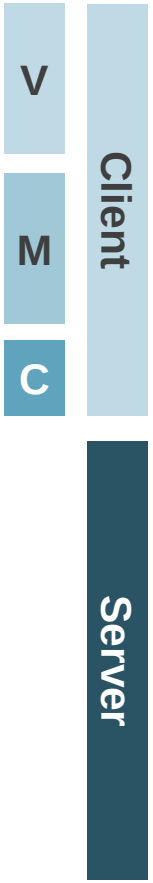
Demo & Live Coding

Mocking des Backends

Mit Sinon.JS asynchrone Verarbeitung synchronisieren

- Asynchrone Aufrufe werden eingefroren
- Antworten und deren Verarbeitung werden zu einem definierten Zeitpunkt ausgelöst
- Übersichtlicher und robuster Testfall, da Warteschleifen entfallen

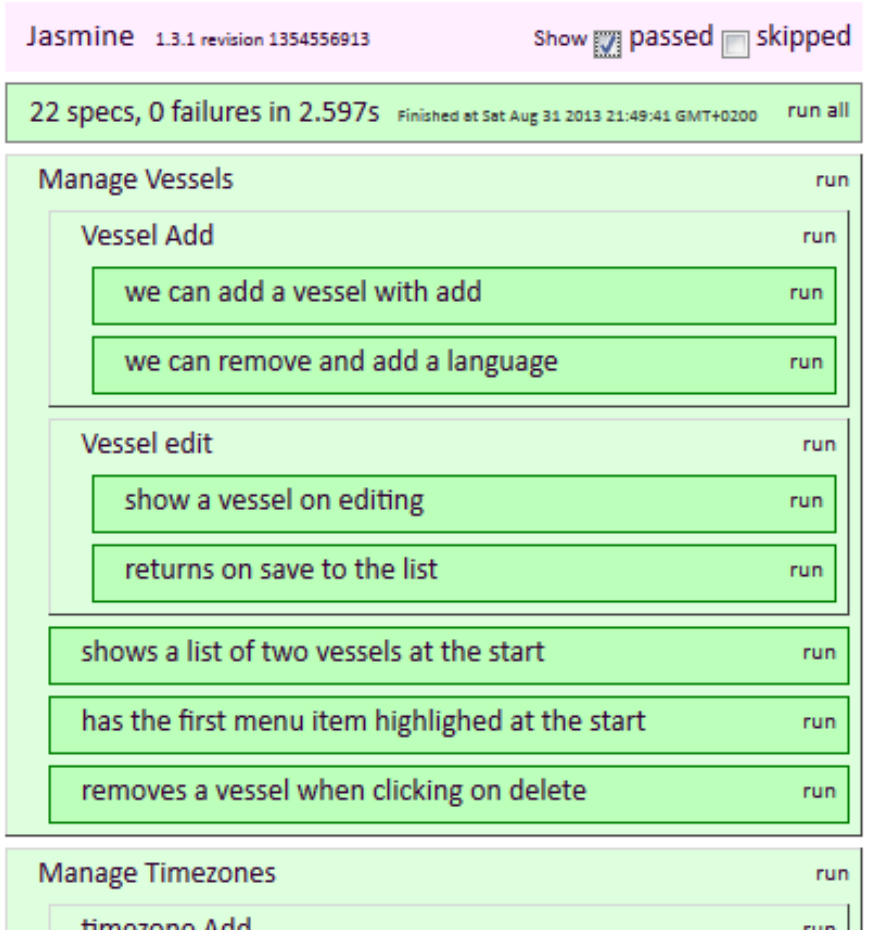
```
it("removes a timezone when clicking on delete",
      function() {

        // starting with two timezones
        expect($("#timezoneList > tbody > tr").length).toBe(2);

        // click on the first one
        $('#timezoneList a[name="delete"]').first().click();

        server.respond();

        // expect only the second to be left over
        expect($("#timezoneList > tbody > tr").length).toBe(1);
        expect($("#timezoneList > tbody > tr")[0]).toContainHtml(
            "Europe/London");

      });
```

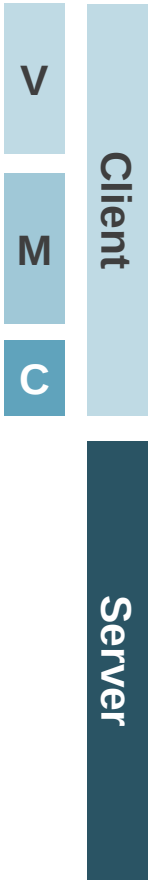


Jasmine ist schnell und einfach

Ein Browser-Reload lässt die Tests erneut laufen

- 22 Tests
- 2,6 s Gesamtlaufzeit
- Cross-Browser
- Unabhängig vom Backend
- Dokumentation aus fachlicher Sicht
- Beliebige Schachtelung
- Chrome etwas schneller als Firefox





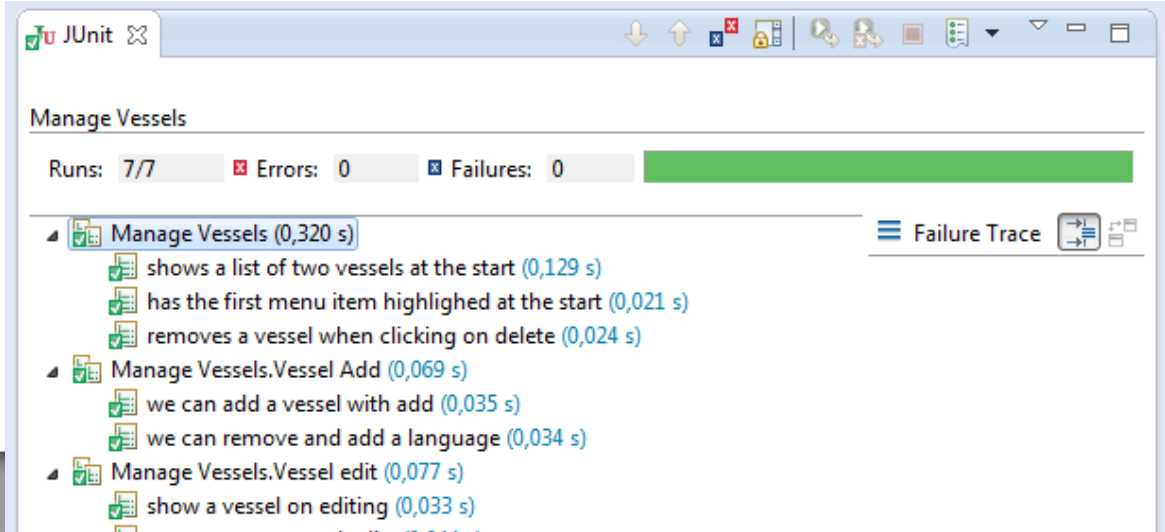
msg systems

Mit PhantomJS auf der Kommandozeile testen

- Integration in CI Server möglich
- Jasmine Tests erstellen XML entsprechend JUnit (für Jenkins, Eclipse et al)

V
M
C
Client
Server

```
$ phantomjs phantomjs-testrunner.js \
    http://localhost:8080/rest-samples/?spec=
```

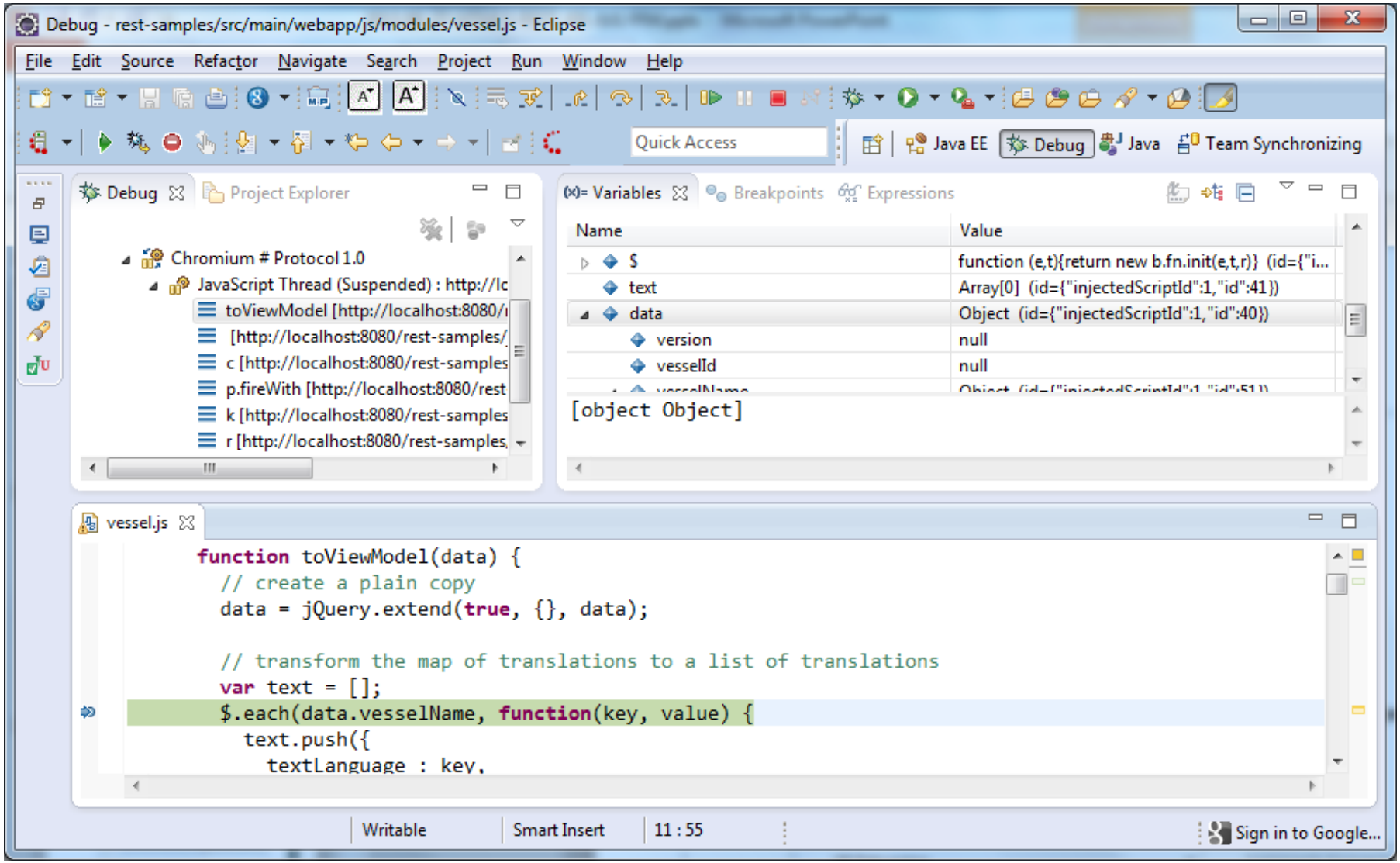


Debugging mit Chome

Wie Java und ein bisschen mehr?

- Verbinden mit einer laufenden Browser-Instanz
- Breakpoint setzen, Variablen sehen
- Hot Code Replacement

Debugging mit Chome



Zusammenfassung

Strukturierte Tests in JavaScript sind möglich

- Testfälle können genauso wie die Anwendung modular sein
- Ausführung ist schnell
- Turnaroundzeiten sind kurz
- JavaScript Entwickler können brauchen keine neuen Technologien kennenzulernen
- Tests sind automatisierbar, auch in Continuous Integration

Vielen Dank für Ihre Aufmerksamkeit

msg systems ag

Mergenthalerallee 73 - 75
65760 Eschborn

Telefon: +49 (171) 5 62 57 67
E-Mail: Alexander.Schwartz@msg-systems.com

www.msg-systems.com

msg systems