Schnelle und leichtgewichtige Anwendungsentwicklung mit HTML5 und JEE REST

Alexander Schwartz

Berlin Expert Days (BedCon) 2013 / 4. April 2013

.msg systems

Was ich vorhabe

1. Worum es geht
2. Fachlicher Einstieg in das Beispiel
3. JEE Stack für REST auf dem Server
4. HTTP + JavaScript Stack auf dem Client
5. Testen der Anwendung
6. Detaillierung Server-Tests
7. Detaillierung Client-Tests
8. Zusammenfassung

Worum es hier geht

HTML+JavaScript und JEE+REST als Plattform

Browser als Client-Plattform

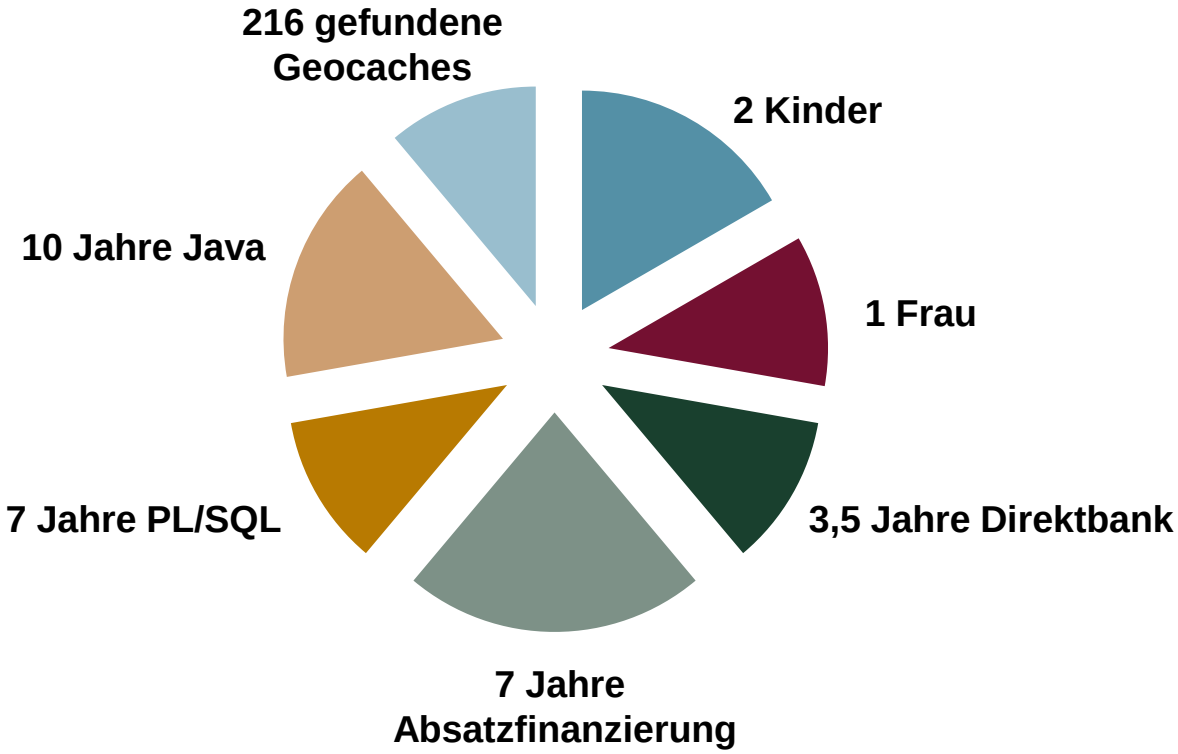
- HTML als Seitenbeschreibung
- CSS für ansprechendes (responsive) Design
- Widget z.B. via jQueryUI + Plugins
- Clientlogik via JavaScript
- Strukturierung mit Javascript MV*-Frameworks (z. B. KnockoutJS)
- Modularisierung und Nachladen von Resourcen (z. B. requireJS)

JEE REST für serverseitige Services

- JAX-RS 1.0/2.0 als Teil von JEE 6/7
- Kommunikation via REST/JSON ideal für JavasScript
- Persistenz via JPA
- Validierung von Daten mit Bean Validation
- CDI für Erweiterungspunkte
- Integration in Enterprise IT dank vieler Java-Bibliotheken

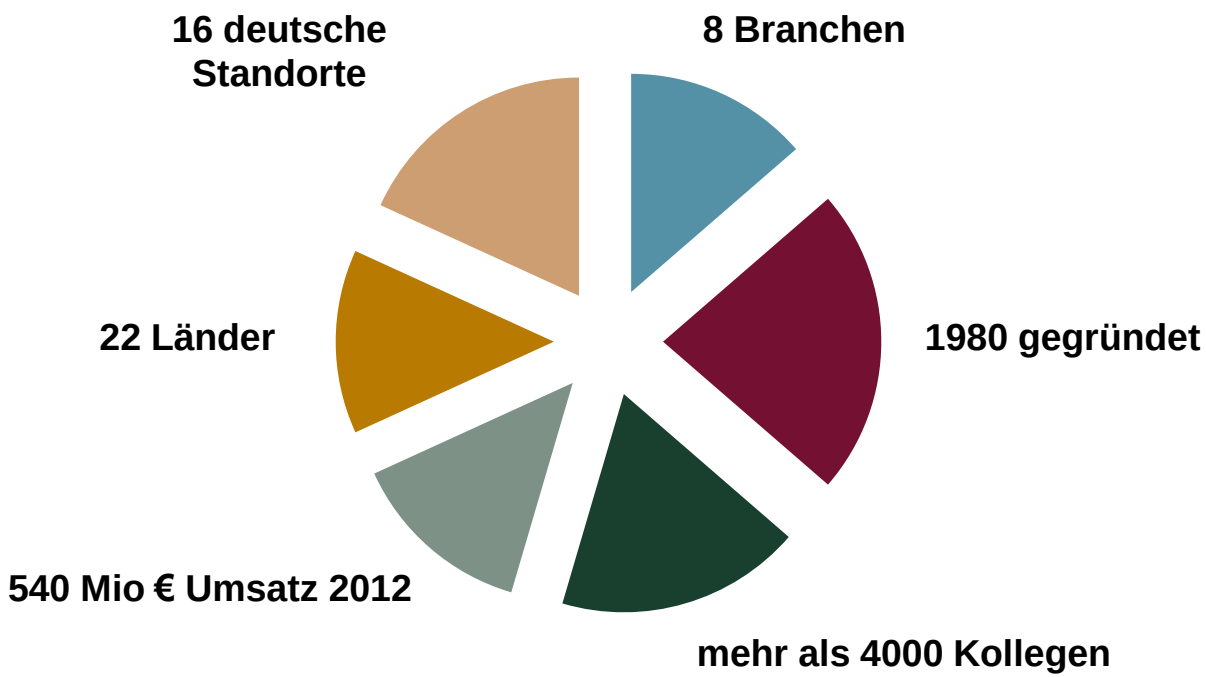
Alexander Schwartz

Lead IT Consultant im GB Travel und Logistics



Mein Sponsor und Arbeitgeber

msg systems ag



Die heutige Aufgabe

Online-Datenbank für (Raum-)Schiffsichtungen

User Story:

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

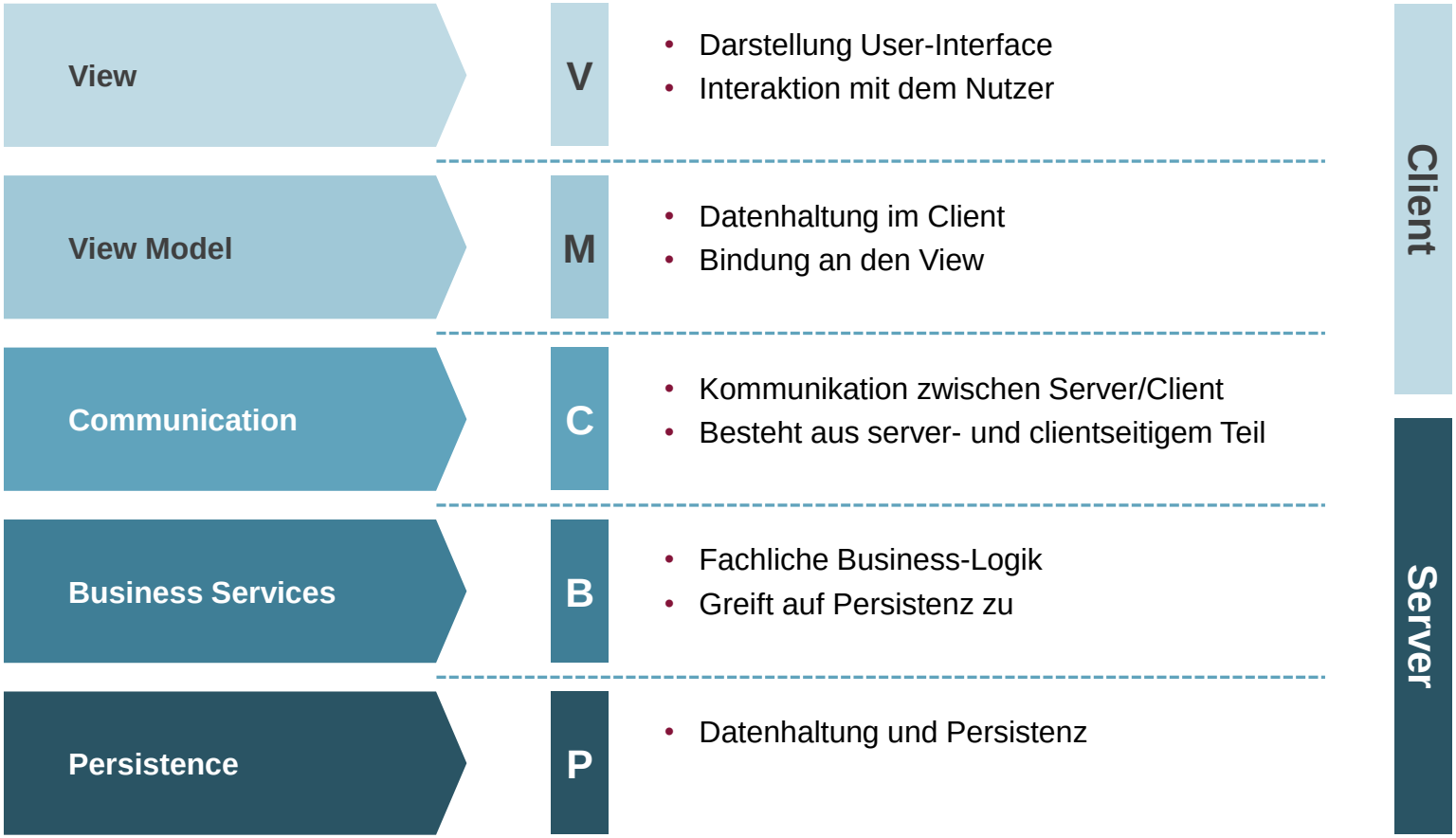
Hierzu erfasst er Schifftstyp, Datum, Zeitzone und eine Notiz.“

Garantiert kein Bezug zu einem Kundenprojekt!

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

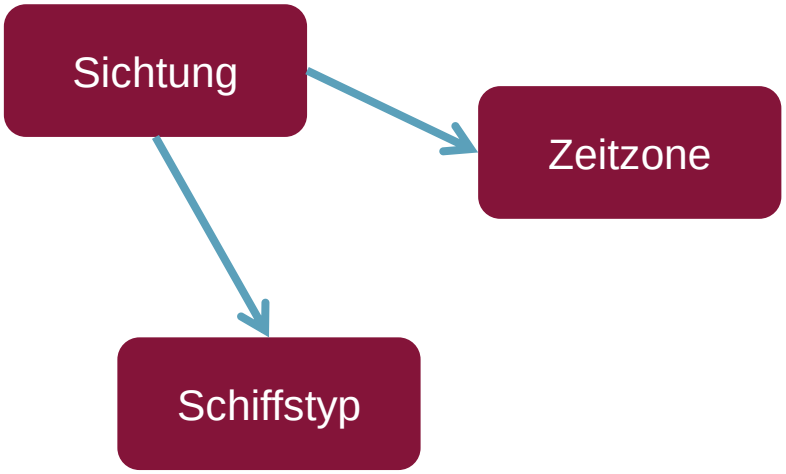
Unser Wegweiser

Eine klare Schichtentrennung hilft als Wegweiser durch die Architektur



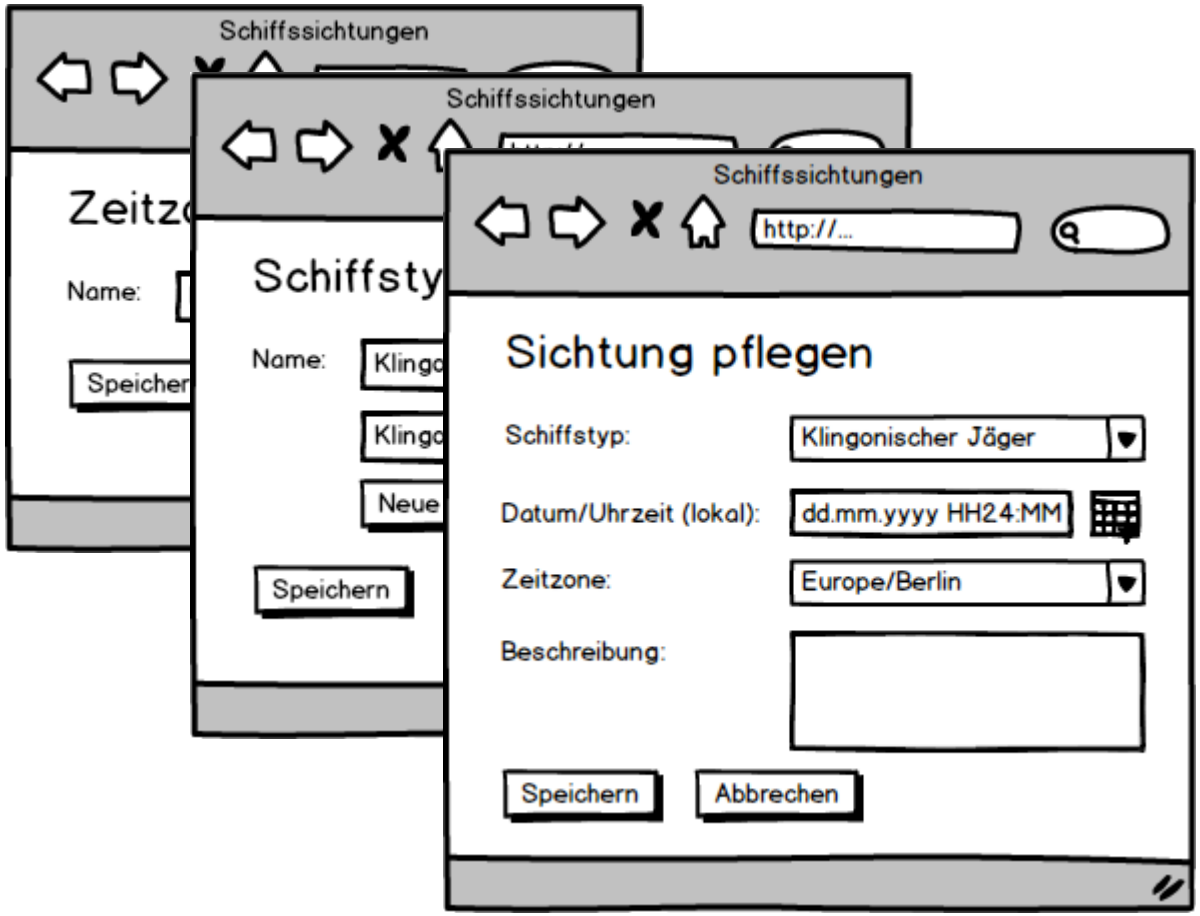
Was es im Inneren zusammenhält

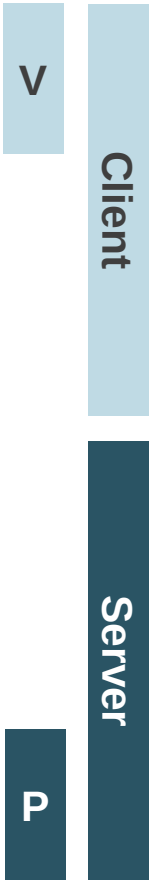
Domain-Model für Struktur hilft Fachabteilung und Entwicklern



Wie´s für den Kunden aussieht

Mockups klären früh das Maskenlayout und verringern Nacharbeiten





Los geht‘s – die erste Entität

Standard-JPA-Entität für „Zeitzone“

- IDs werden automatisch generiert
- Validierung über Bean-Validation
- Basis-Klasse AbstractEntity für Optimistic Locking

```
@Entity
public class Timezone extends AbstractEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long timezoneId;

  @NotEmpty
  private String timezoneName;

…

}
```

```
@MappedSuperclass
public class AbstractEntity {

  @Version
  private Integer version;

…

}
```

Entität mit Übersetzung

Komplexe Entität „Schiffstyp“

- Übersetzbares Element als Entität Translation
- CascadeType.ALL für gemeinsame Speicherung
- Orphan Removal für Housekeeping
- Translation enthält eine Map mit Übersetzungen

```
@Entity
public class Vessel extends AbstractEntity {

  @Id
  @GeneratedValue
        (strategy = GenerationType.IDENTITY)
  private Long vesselId;

  @OneToOne(fetch = FetchType.EAGER,
                cascade = CascadeType.ALL,
                orphanRemoval = true)
  @JoinColumn(name = "vesselName")
  private Translation vesselName;

…
}
```

```
@Entity
public class Translation extends AbstractEntity {

  @Id
  @GeneratedValue
            (strategy = GenerationType.IDENTITY)
  private Long translationId;

  @ElementCollection
  @CollectionTable(name = "Text", joinColumns =
                @JoinColumn(name = "translationId"))
  @MapKeyColumn(name = "textLanguage")
  @Column(name = "textString")
  private Map<Locale, String> texts;

…
}
```

Client

Server

P

Vorsicht beim (Daten-Model) Bau

Abbildung fachlicher Datentypen auf technische Datentypen

Fachlich	Java	JSON	JavaScript
Zeichenkette	String	String	String
Ganzzahl	Long	Number	Number
Dezimalbruch	BigDecimal	String *	String *
Datum	LocalDateTime	String **	String **

* JavaScript Number-Type unterstützt nur Fließkommazahlen, aber keine Dezimalzahlen. Es können dadurch Rundungsdifferenzen auftreten, die fachlich nicht gewünscht sind. Daher Fallback auf String.

** „Standard“ bei JSON ist für Datumsangaben Sekunden seit 1970 und Zeit UTC. LocalDateTime lässt sich so nicht abbilden; Date in JavaScript wird in der Zeitzone des Browser dargestellt => keine Kontrolle durch Anwendung möglich möglich. Daher Fallback auf Datum als String im ISO-Format

Komplexe Entität „Sichtung“

- ManyToOne-Relation für Selectbox Vessel – ohne Cascade!
- Joda-Elemente für Zeiten; usertype für Joda-Persistenz

```
@Entity
public class Sighting extends AbstractEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long sightingId;

  private String sightingMemo;

  @ManyToOne
  @JoinColumn(name = "vesselId")
  private Vessel vessel;

  @Type(type = "org.jadira.usertype.dateandtime.joda.PersistentDateTimeZoneAsString")
  private DateTimeZone sightingTimezone;

  @Type(type = "org.jadira.usertype.dateandtime.joda.PersistentLocalDateTime")
  private LocalDateTime sightingDate;
…
}
```

Erster REST Endpoint

Java-Generics sind gut!

- DefaultRestEndpoint liefert CRUD Funktionalität
- Genug „Platz“ im Endpoint für spezifische Funktionalität (z.B. Aufruf von Business Services)
- Sicherstellung „Don‘t Repeat Yourself“

```
@Stateless
@Path("/timezone")
public class TimezoneEndpoint extends DefaultRestEndpoint<Timezone> {

}
```

```
@Produces({ "application/json", "text/xml" })
@Consumes({ "application/json", "text/xml" })
public abstract class
DefaultRestEndpoint<ENTITY extends HasId> {

< … 500 Zeilen Generics, Reflection, JPA Metamodel,
JavaDoc … Einblick auf der nächsten Folie …  >

}
```

Blick durch Schlüsselloch in DefaultRestEndpoint

```
public abstract class DefaultRestEndpoint<ENTITY extends HasId> {

…
 @GE
  @Pa
  publ
reque
   ENT

   if (r
     thr
   }

   pull

   retu
 }

…
```

```
public abstract class DefaultRestEndpoint<ENTITY extends HasId> {

…
 @PO
  @Pa
  pub
   bin

   em
   em

   Uri
   loc
   UR

   ret
      .
 }

…
```

```
public abstract class DefaultRestEndpoint<ENTITY extends HasId> {

….

 @POST
  @Path("/qbe")
  public List<ENTITY> queryByExample(ENTITY entity) {
    Session session = (Session) em.getDelegate();
    Example example = Example.create(entity).enableLike(MatchMode.ANYWHERE)
        .ignoreCase();
    Criteria criteria = session.createCriteria(entity.getClass()).add(example);
    addSubCriteria(criteria, entity);

    if (criteria.list().isEmpty()) {
      return null;
    } else {
      pullByJsonView(criteria.list(), getExtendedView());
      return criteria.list();
    }
  }
…
```

Object Graph Traversal

Mit @JsonView-Annotationen kann die Sichtbarkeit eingeschränkt werden

```
@Stateless
@Path("/sighting")
public class SightingEndpoint extends DefaultRestEndpoint<Sighting> {

  @Override
  protected Class<?> getExtendedView() {
    return Sighting.Extended.class;
  }

  @Override
  @GET
  @Path("/{id:.+}")
  @JsonView(Sighting.Extended.class)
  public Response findById(@PathParam("id") S
          @Context Request request) {
    return super.findById(id, request);
  }

…
}
```

```
public abstract class
DefaultRestEndpoint<ENTITY extends HasId>
{

 @GET
  @JsonView(ListView.class)
  public Response listAll() {

…
}
```

```
public class Sighting extends AbstractEntity {

…
 @JsonView(Extended.class)
  public String getSightingMemo() {
    return sightingMemo;
  }

…
}
```

Client

C

Server

Object Graph Traversal Output

In der Liste sind weniger Attribute angezeigt, in der Einzelansicht mehr

- Fachliche Überlegung: kein sightingMemo in der Übersicht notwendig
- Technische Überlegung: kein version in der Übersicht notwendig
- ABER: @JsonView gibt‘s nur bei Jackson (nicht JEE Standard)

GET http://localhost:8080/rest-samples/rest/sighting


```
[{"sightingId":1,
  "vessel":{"vesselId":1,"vesselName":"Klingonischer Jäger"},
   "sightingTimezone":"Europe/Berlin",
   "sightingDate":"2013-04-11T11:00:00.000"},
 {"sightingId":2,
  "vessel":{"vesselId":1,"vesselName":"Klingonischer Jäger"},
   "sightingTimezone":"Europe/London",
   "sightingDate":"2013-04-11T22:00:00.000"}]
```


GET http://localhost:8080/rest-samples/rest/sighting/-1


```
{"version":1,
  "sightingId":1,
  "sightingMemo":"unheimlich!",
  "vessel":{"version":0,"vesselId":-1,"vesselName":"Klingonischer Jäger"},
  "sightingTimezone":"Europe/Berlin",
  "sightingDate":"2013-04-11T11:00:00.000"}
```


Client

C

Server

Erweiterungspunkt: Custom Serialisierer

DateTimeZone serialisieren (da kein Standardtyp)

```
public class DateTimeZoneDeserializer extends
JsonDeserializer<DateTimeZone> {

 @Override
  public DateTimeZone deserialize(JsonParser jparse,
      DeserializationContext context) throws IOException {
    String text = jparse.getText();
    if (text == null || text.trim().length() == 0) {
      return null;
    } else {
      return DateTimeZone.forID(text);
    }
  }
}
```

```
public class DateTimeZoneSerializer extends JsonSerializer<DateTimeZone> {

 @Override
 public void serialize(DateTimeZone value, JsonGenerator jgen,
      SerializerProvider provider) throws IOException {
    jgen.writeString(value.getID());
  }

}
```

Erweiterungspunkt: Custom Serialisierer

Registrieren der Serialisierer / De-Serialisierer

- Serialiserer für DateTimeZone und BigDecimal registrieren

```
@Provider
public class CustomObjectMapper implements ContextResolver<ObjectMapper> {
  @Override
  public ObjectMapper getContext(Class<?> type) {
    final ObjectMapper result = new ObjectMapper();
    SimpleModule module = new SimpleModule(getClass().getName(), new Version(1,
        0, 0, null))
        .addDeserializer(BigDecimal.class, new BigDecimalAmountDeserializer())
        .addSerializer(BigDecimal.class, new BigDecimalAmountSerializer())
        .addDeserializer(DateTimeZone.class, new DateTimeZoneDeserializer())
        .addSerializer(DateTimeZone.class, new DateTimeZoneSerializer());
    result.registerModule(module);
    result.configure(Feature.WRITE_DATES_AS_TIMESTAMPS, false);
    return result;
  }
}
```

Exception Handler

Für jede Exception kann (muss) ein eigener Handler definiert werden

```
@Provider
public class EntityNotFoundExceptionMapper implements
    ExceptionMapper<EntityNotFoundException> {

  @Inject
  private Localizer localizer;

  @Override
  public Response toResponse(EntityNotFoundException exception) {
    Map<String, String> responseObj = new HashMap<String, String>();
    responseObj.put("general", localizer.localize("error.entityNotFound"));
    return Response.status(Response.Status.NOT_FOUND).entity(responseObj)
        .build();
  }
}
```

```
public class Localizer {

  @Inject
  private HttpServletRequest httpServletRequest;

public String localize(String key) {
    try {
      ResourceBundle rb = ResourceBundle.getBundle("messages",
          httpServletRequest.getLocale());
      return rb.getString(key);
    } catch (MissingResourceException e) {
      return "{" + key + "}";
    }
  }
}
```

Bean Validation plus Exception-Handler

```
@Entity
public class Sighting extends AbstractEntity {
…
  @NotNull
  @Size(min = 2, message = "{sighting.memo.minLength}")
  private String sightingMemo;
…
}
```

```
#ValidationMessages_de.properties
sighting.memo.minLength=Muss mindestens {min} Zeichen lang sein
```

```
@Provider
public class ConstraintViolationExceptionMapper implements
    ExceptionMapper<ConstraintViolationException> {

  @Override
  public Response toResponse(ConstraintViolationException exception) {
    Map<String, Object> responseObj = new HashMap<String, Object>();

    for (ConstraintViolation<?> violation : exception.getConstraintViolations()) {
        // … < 40 Zeilen Code > …
    }

    return Response.status(Response.Status.BAD_REQUEST).entity(responseObj)
        .build();
  }
```

JEE Zwischenfazit

JEE kann REST!

- Querschnittsfunktionen lassen sich einfach von domänenspezifischen Elementen trennen
- APIs und Erweiterungspunkte für spezifische Implementierungen sind vorhanden
- JSON Serialisierung ist nicht in JEE 6.0 standardisiert
- Durch Generics, JPA Metamodel und Reflection gelingt es, das DRY Prinzip durchzuhalten

HTML Single Page Apps – aber bitte strukturiert und mit Modularisierung

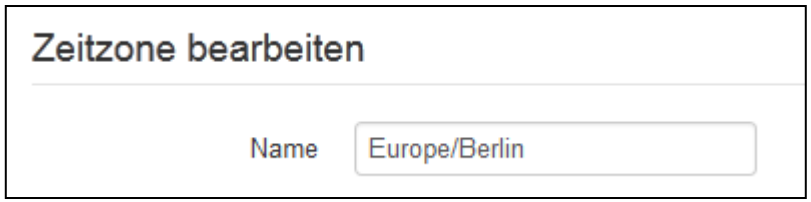
Anforderungen:

- HTML für den View
- View-Model für Datenhaltung
- Bi-direktionales Datenbinding View / View-Model
- Modularisierung von JavaScript-Bibliotheken, aber auch Templates und JavaScript-Code für Usecases der Anwendung
- Stateful-URLs innerhalb der Anwendung; Vor-/Zurück-Navigation

CSS-Frameworks wie Bootstrap erlauben ein schlankes HTML

Beispiel: Eingabefelder mit Label und Eingabefeld

```
<form class="form-horizontal" >
  <fieldset>
    <legend>Zeitzone bearbeiten</legend>
    <div class="row-fluid">
      <div class="control-group">
        <label class="control-label" for="bezeichnung">Name</label>
        <div class="controls">
          <input type="text" id="timezoneName" class="input-large" />
        </div>
      </div>
    </div>
  </fieldset>
</form>
```



V

Client

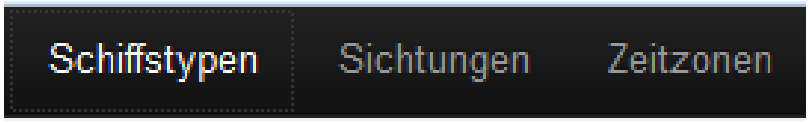
Server

Ein Menü anzeigen – Trennung View vom View Model

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>
```



V

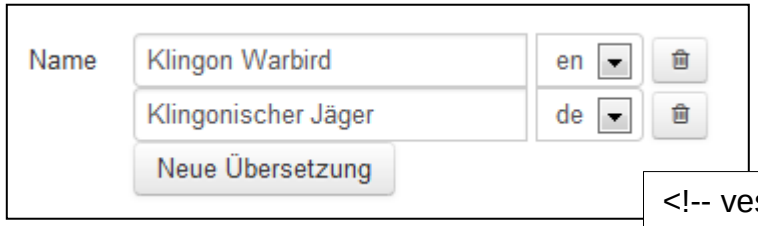
M

Client

Server

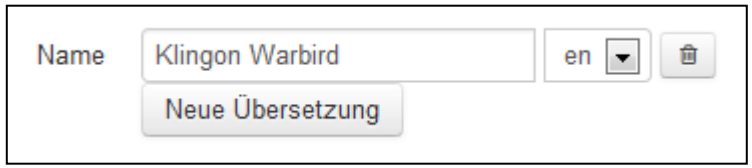
Bidirektionales Binding

Eine Texteingabe im Input-Feld aktualisiert das Modell – eine Änderung im Modell aktualisiert den View




```
<!-- vessel.html -->
<a class="btn btn-small" data-bind="click:
$parents[2].deleteLanguage">
    <i class="icon-trash"></i>
</a>
```

```
<!-- vessel.js -->
 self.deleteLanguage = function(language) {
          self.vessel().vesselName.remove(language);
 };
```



V
M
Client
Server

Modularisierung der Anwendung (Model) mit requireJS

- Jedes Modul erhält einen „Header“ mit Abhängigkeiten
- Für 3rd-Party-Bibliotheken werden im der Startprozedur Abhängigkeiten definiert
- requireJS sorgt für parallelen Download der JS-Module


```
<!– index.html -->
<script data-main="js/main"
src="js/libs/require.js"></script>
```

```
//main.js
require.config({
  shim : {
    jquery : {
      exports : "jQuery"
    },
    knockout : {
      deps : [ 'jquery' ]
    },
    'knockout.validation' : {
      deps : [ 'knockout' ]
    },…
```

```
<!-- vessel.js -->
define(
    [ 'knockout', 'jquery', 'mapping', 'hasher',
'crossroads', 'menu' ],
    function(ko, $, mapping, hasher, crossroads,
menu) {
    // logik…
 });
```

Modularisierung (Usecases)

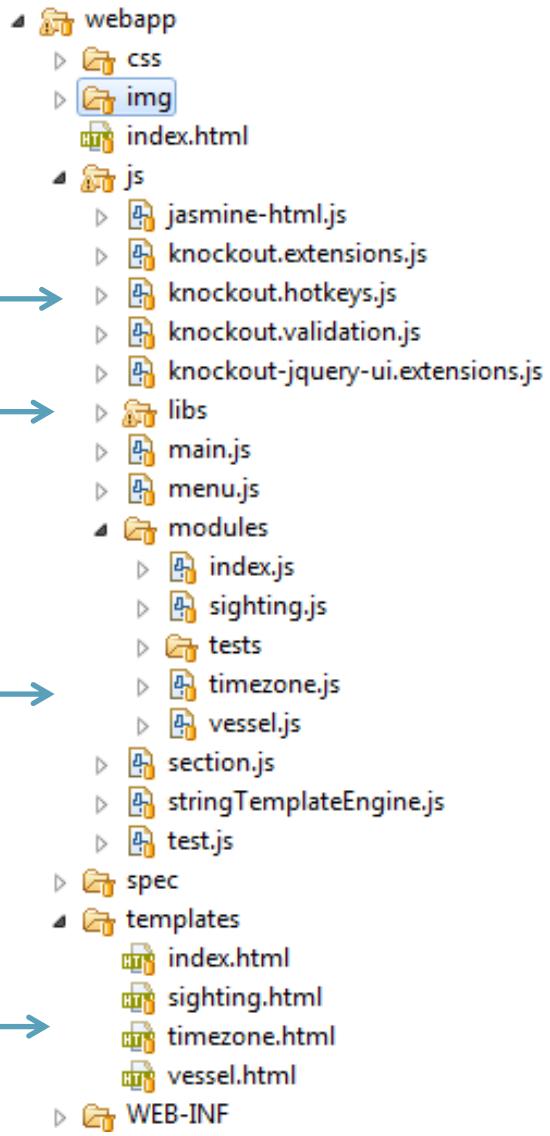
Modularisierung der Use-Cases

übergreifende Module

Bibliotheken

eine JS Datei pro Use Case

ein Template pro Use Case



V

M

Client

Server

Stateful URLs

URLs für den direkten Einstieg – und Vor-/Zurücknavigation

- HTML5 unterstützt das „History“ API, bei dem URLs sich ändern können, auch wenn die Seite gleich bleibt
- Zusatzinformationen zu einer Seite können per JavaScript in der Browser-Historie abgelegt werden

Ohne History-API:

- Workaround: URL hinter dem Hash-Tag

  http://localhost:8080/rest-samples/#/vessel/edit/1
  http://localhost:8080/rest-samples/#/vessel/main
  http://localhost:8080/rest-samples/#/timezone/edit/1

- Unterstützt z.B. durch Crossroads.js

```
crossroads.addRoute(/vessel\/edit\/(.+)$/, function(id) {
        $.get("rest/vessel/" + id, function(data) {
          self.vessel(mapping.fromJS(toViewModel(data)));
          self.vesselList(null);
        });
      });
```

V

Client

Server

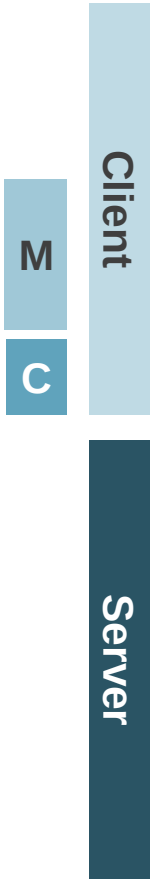
Das Model im Backend kann weitgehend beibehalten werden

- Laden eines Elements

```
$.get("rest/vessel/" + id, function(data) {
    self.vessel(mapping.fromJS(toViewModel(data)));
    self.vesselList(null);
});
```

- Das Model kann ggf. für den View punktuell transformiert werden, z.B. HashMap als Array

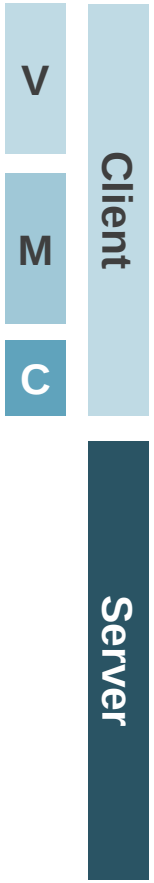
```
    function toViewModel(data) {
      data = jQuery.extend(true, {}, data);

      var text = [];
      $.each(data.vesselName, function(key, value) {
        text.push({
          textLanguage : key,
          textString : value
        });
      });
      data.vesselName = text;

      return data;
    }
```



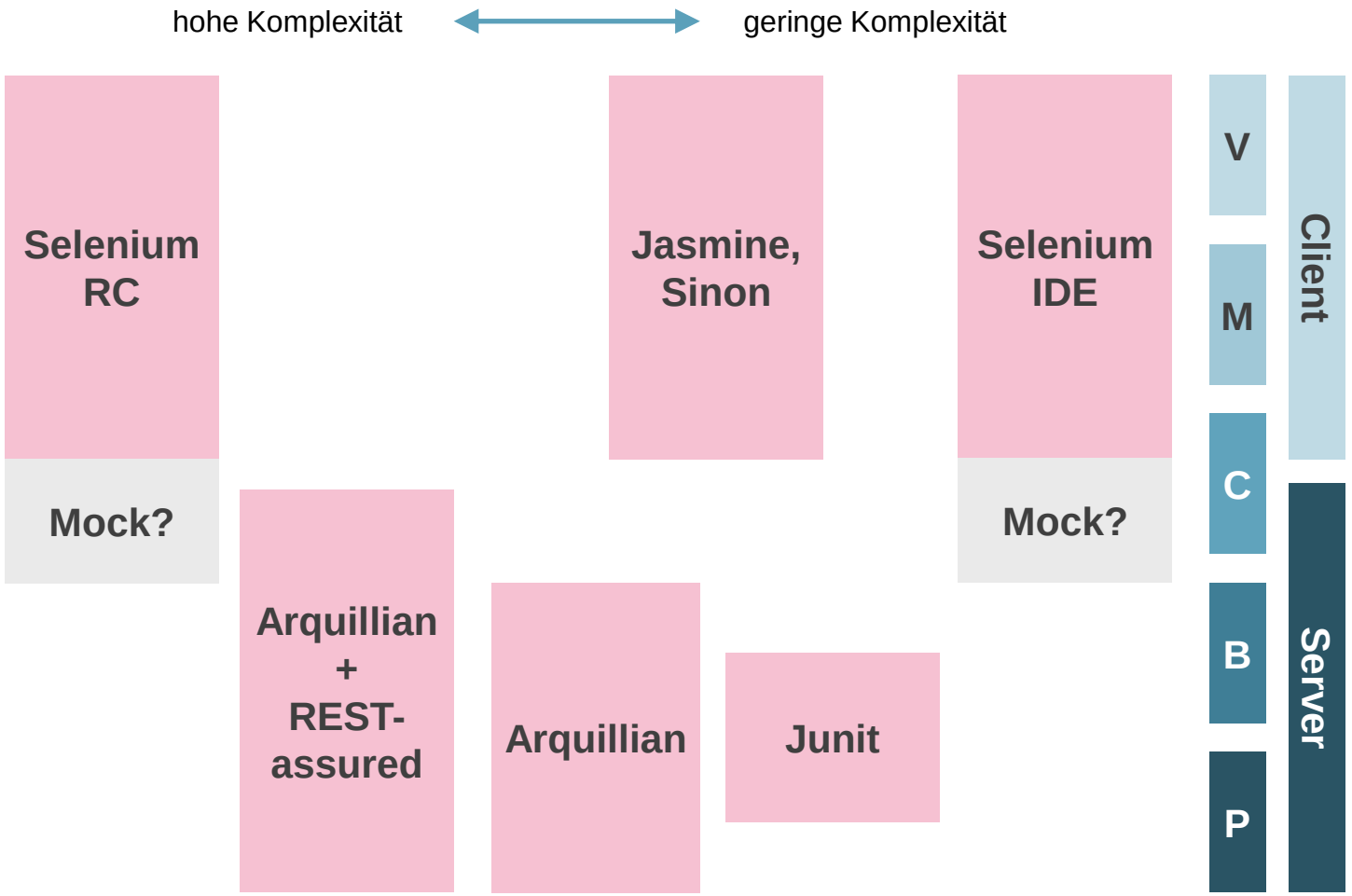
HTML/JS Zwischenfazit

HTML + JavaScript können strukturiert werden!

- Querschnittsfunktionen lassen sich einfach von domänenspezifischen Elementen trennen
- APIs und Erweiterungspunkte für spezifische Implementierungen sind vorhanden
- Die Zahl der Bibliotheken und APIs ist deutlich größer als im JEE Bereich
- Ist der Rahmen vorbereitet, können einfach neue Use Cases dazuentwickelt werden (je eine JS + HTML Datei pro Use Case)
- Code-Änderungen sind nach einem Browser-Refresh sofort sichtbar



Verschiedene Testframeworks decken verschiedene Bereiche ab



Arquillian + REST assured

Test-Anwendungen automatisiert packen und laufen lassen

Arquillian:

- Automatisiertes Deployment von Teilen der Server-Anwendung (ggf. ergänzt um Mock-Komponenten)

REST-assured:

- Testen der REST-services mit Fluent API

```
@RunWith(Arquillian.class)
public class MesseEndpointTest {

  @Deployment
  public static WebArchive createArchiveAndDeploy() {
    WebArchive war = ShrinkWrap.create(WebArchive.class,
        "arquillian-rest-demo.war");

    war.addPackages(…);
    war.addAsLibraries(…);
    return war;
  }

  @Test
  public void testReturnFilledList() throws Exception {

    Response r = given().log().all()
        .contentType(ContentType.JSON).expect()
        .body("[0].meBezeichnung", equalTo("Vessel 1"))
        .statusCode(Status.OK.getStatusCode()).when()
        .get("/rest-samples/rest/vessel");

    assertThat("only one element exists",
        r.body().jsonPath().getList("").size(), equalTo(1));
  }
```

Anforderungen beschreiben und Tests automatisieren auf dem Client

Kleine Tests:

- Unit-Tests für JavaScript-Code-Bibliotheken
- Ergänzt durch HTML-Fragmente
- DOM-Interaktion möglich durch jQuery Integration

Große Tests:

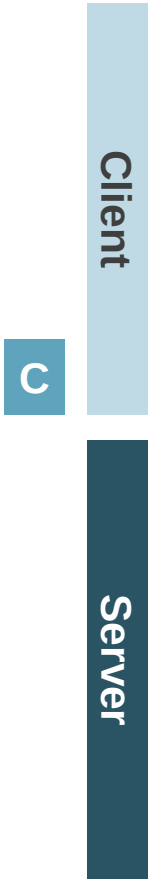
- Tests der vollständigen Use Cases


```
describe("Manage Vessels", function() {
  it("shows a list of two vessels at the start",
    function() {
      expect($("#vesselList")).toBeVisible();
      expect($("#vesselList > tbody > tr")[0]).toContainHtml("Vessel 1");
      expect($("#vesselList > tbody > tr")[1]).toContainHtml("Vessel 2");
    });
});
```

Backend kann via JavaScript simuliert werden

Frontend-Entwickler ist unabhängig vom Backend – und kann selber Testen

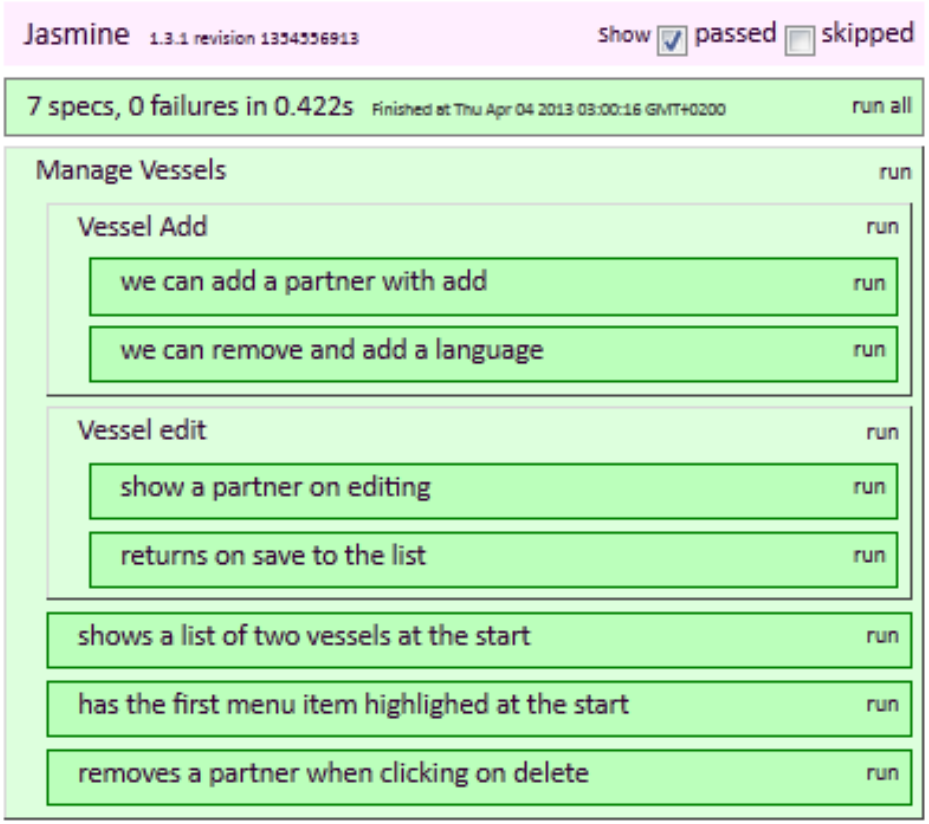
```
describe("Manage Vessels", function() {
    beforeEach(function() {
        server = sinon.fakeServer.create();

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

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

        server.respondWith("GET", "rest/vessel/1", [ 200, {
          "Content-Type" : "application/json"
        }, vessel ]);

     afterEach(function() {
       // disable fake server
       server.restore();
     });
 );
```

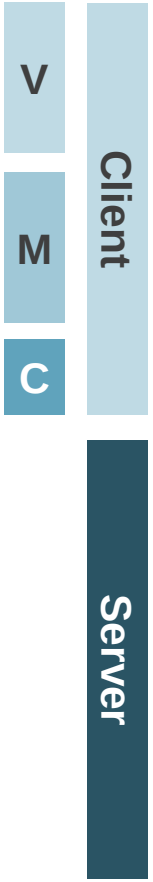


Jasmine ist schnell und einfach

Ein Browser-Reload lässt die Tests erneut laufen

- 7 Tests
- 760 ms Gesamtlaufzeit
- Cross-Browser
- Unabhängig vom Backend
- Dokumentation aus fachlicher Sicht
- Beliebige Schachtelung





Zusammenfassung

JEE REST Backend ergänzt sich mit HTML/JS Frontend

- Backend und Frontend können fachliche Komponenten und technische Komponenten von einander getrennt werden
- Steht der Rahmen, so können Entwickler an unabhängig an verschiedenen Use Cases arbeiten
- Steht der Rahmen, so können Entwickler mit geringen Vorkenntnissen in die Entwicklung einsteigen
- Frontend und Backend können separat entwickelt werden
- HTML+JS bietet schnelle Turnaround-Zeiten in der Entwicklung
- JEE REST bietet Integration in Enterprise IT und stabile Laufzeitumgebung
- Sowohl Frontend als auch Backend können einfach skalieren
- Durch neue HTML5 Features wie Local Storage und kann die Skalierung weiter erhöht werden

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