



# Workshop - Konzepte moderner Web Applikationen

Jan Illetschko



# Ziele

- Unterschiede Anwendungsarchitekturen (Client-Server, 3-Tier, Micro Services)
- Funktionsschichten Web Anwendungen
- Transaktionen in zustandslosen, verteilten Anwendungen
  - alternative Konzepte
- UI Layout im mobilen Zeitalter
- Interface Contracts mittels REST



# Agenda

- Anwendungsarchitekturen
  - Client-Server
  - Multi-Tier Architekturen
  - Micro Services
- Micro Services
  - REST
  - Contract-driven Interface Design
- Browser Frontends
  - HTTP
  - Single Page Anwendungen
  - UI Layout und Responsive Design



# Anwendungs-Architekturen

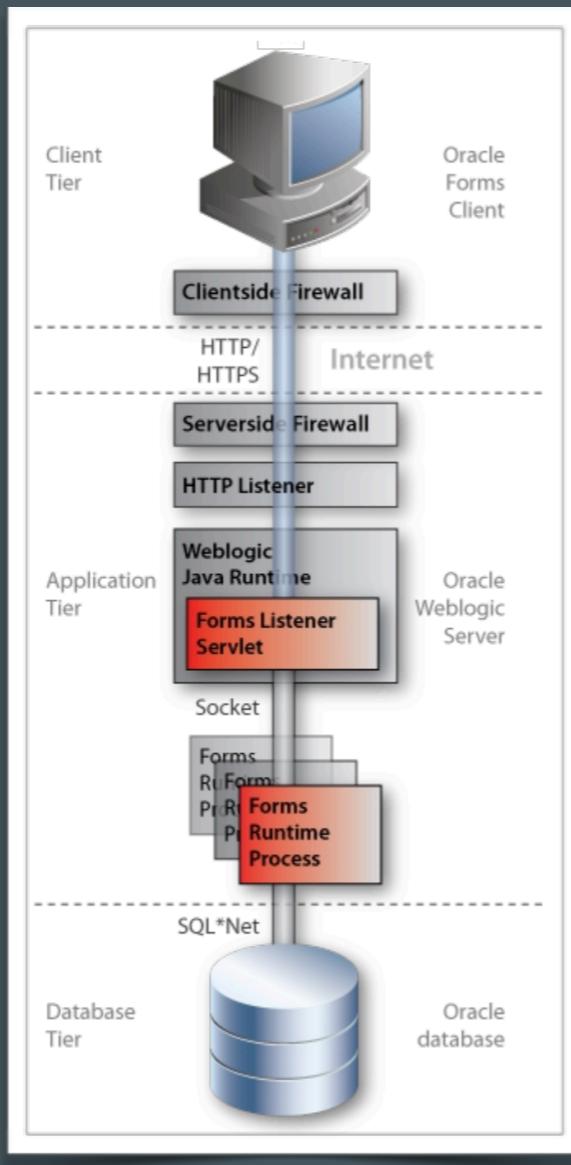
# Erwartungen an eine Anwendungsarchitektur

- Entwicklung
  - Erlernbarkeit
  - Zukunftssicherheit
  - Erweiterbarkeit
  - Wartbarkeit der Code-Basis
  - Flexibilität
  - Tool-Unterstützung
  - Modernes User-Interface
  - Community
  - Agilität
- Deployment
  - Breite Verfügbarkeit
- Betrieb
  - Skalierbarkeit
  - Hochverfügbarkeit
  - Managebarkeit



# Client Server

# Client-Server Überblick (Forms)



- Clients haben eine dedizierte Verbindung zur Datenbank
- Client Benutzer entsprechen dem Datenbankbenutzer
- Anwendungs-Logik liegt vorrangig in der Datenbank



# Flipchart Client Server Pros/Cons

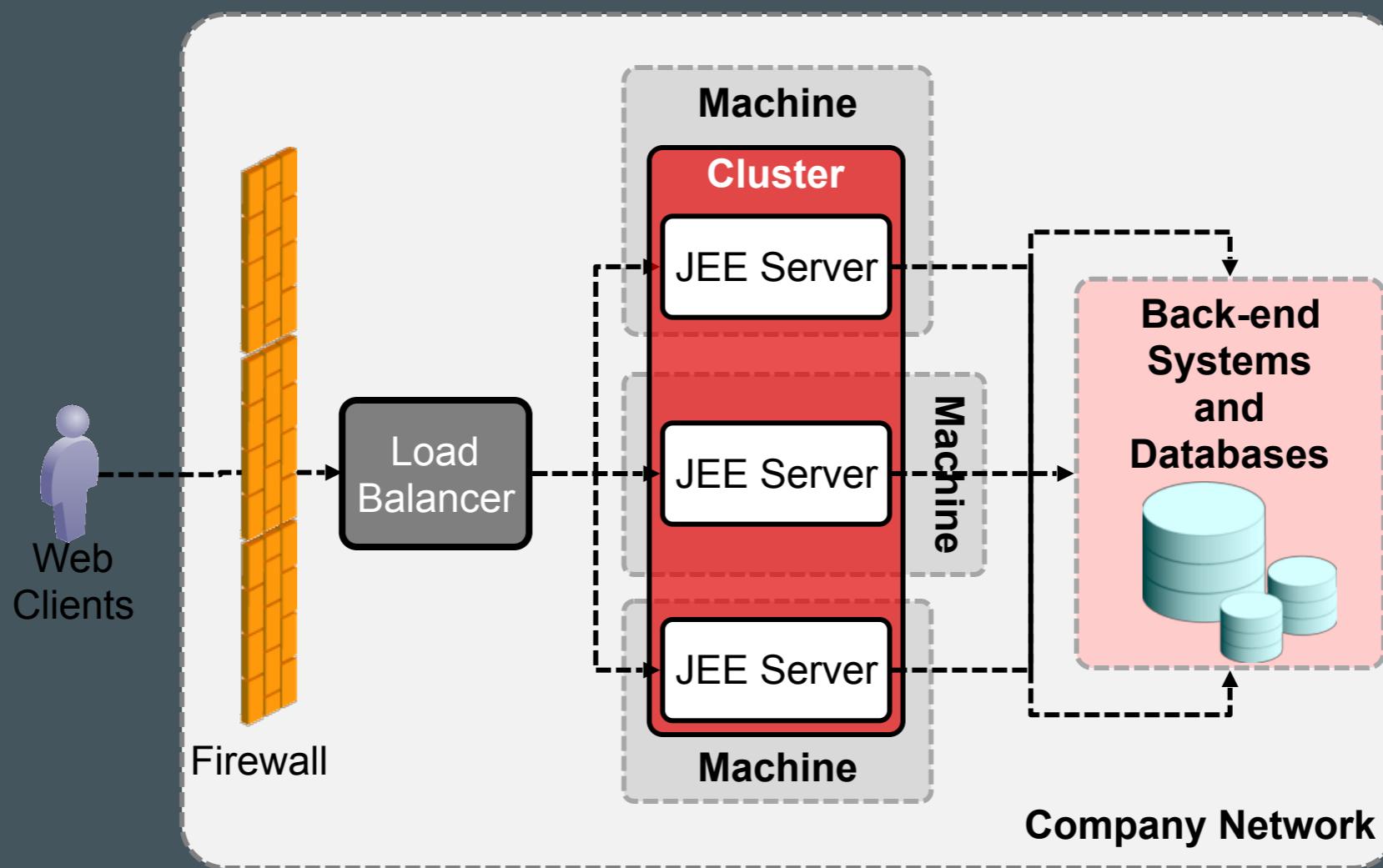
# Client-Server Pros & Cons

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>• Geringe Einstiegs-Hürde</li><li>• Überschaubarer Funktionsumfang</li><li>• Einfaches State-Management</li><li>• Benutzer und Berechtigungsprüfung in der Datenbank</li></ul> | <ul style="list-style-type: none"><li>• Geringer Funktionsumfang</li><li>• Wenig flexibel</li><li>• Client benötigt</li><li>• Schlechte Skalierbarkeit</li><li>• Mobile Devices nicht unterstützt</li><li>• Kaum neue Features</li></ul> |
|--|--|

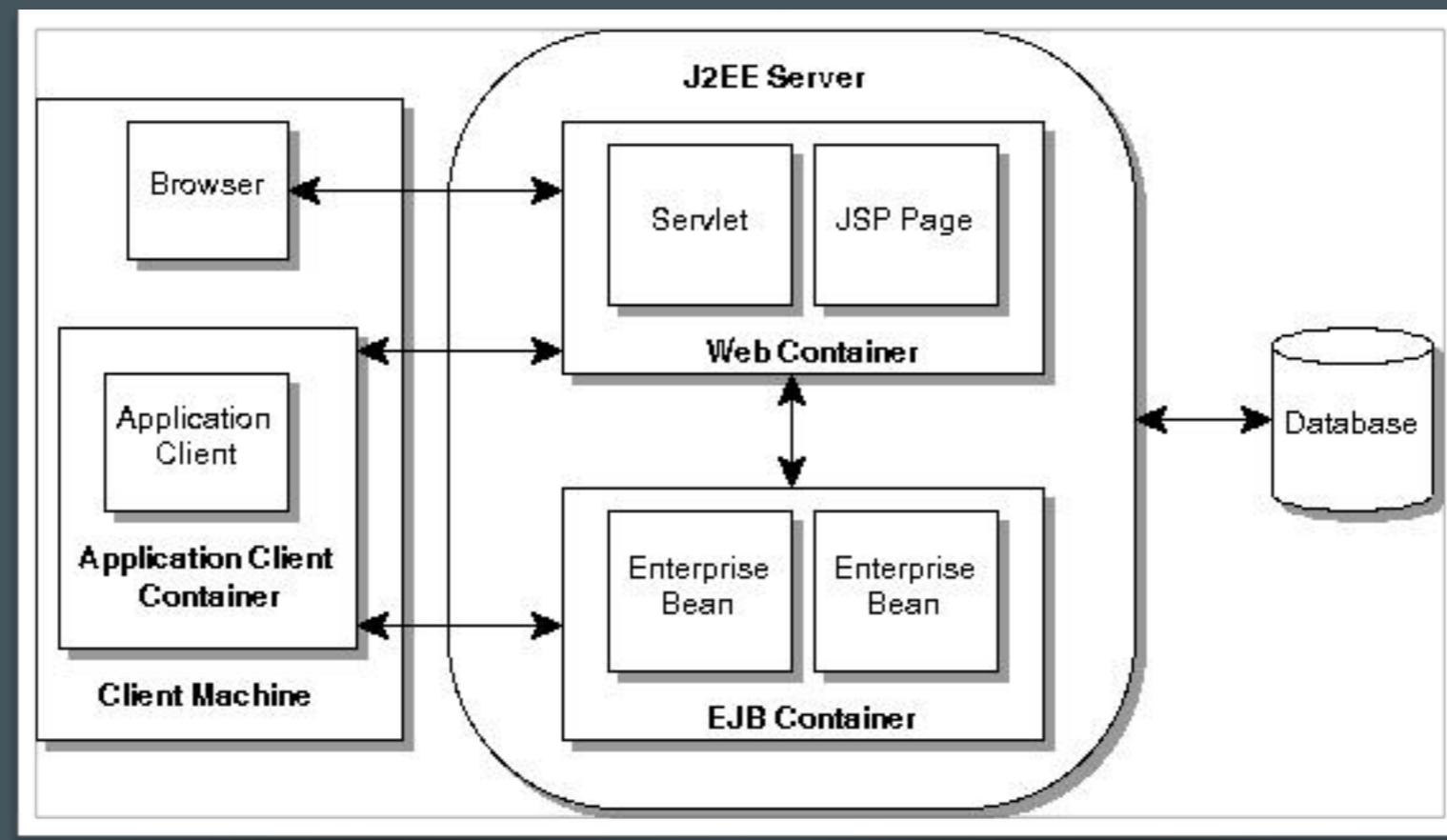


# Multi-Tier Architekturen

# Multi-Tier Überblick (JEE)



# JEE Architektur





# Flipchart Multi-Tier Pros/Cons

# Multi-Tier Pros & Cons

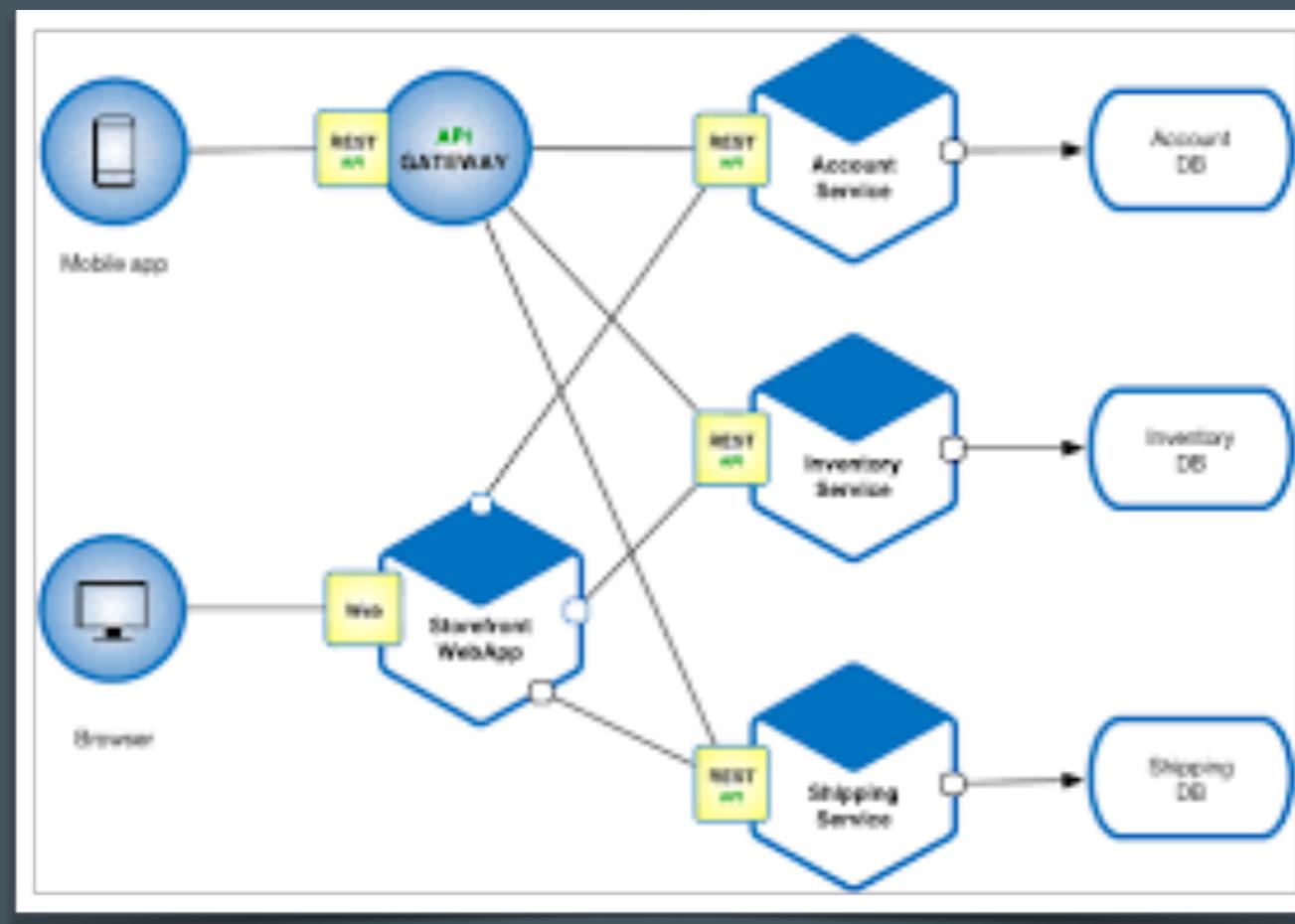
- Web Clients (Browser)
- Modular
- Transaktionssupport
- Flexibel
- Skalierbar (horizontal)

- Typischerweise monolithische Anwendungen
- Zustandsbehaftet
- Großer Einarbeitungsaufwand
- Komplexer Betrieb
- Schlechter Cloud Support
- Agilität



# Micro Service Architekturen

# Micro Services Overview





# Eigenschaften von Micro Services

- Einzelne Prozesse die über Netzwerk kommunizieren
- Unabhängig deploybar
- Einzelne Services sind leicht zu ersetzen
- Services können in den unterschiedlichsten Technologien entwickelt werden
- Werden unabhängig von einander entwickelt
- Dezentralisiert
- Build/Deployment typischerweise automatisiert
- Können unterschiedlich gegliedert werden:
  - Funktionsblöcken
  - Abteilungen
  - Prozessen
  - etc



# Flipchart Micro Services Pros/Cons

# Micro Service Pros & Cons

- Beliebige Clients (Browser, CLI, mobile App, etc)
  - Extrem Modular
  - Flexibel
  - Skalierbar (horizontal, vertikal)
  - Zustandslos
  - Cloud
  - Agil
- 
- Transaktionshandling
  - Heterogen (?)
  - Komplexer Betrieb
  - Komplexes Deployment



3kraft™

# Beispiel Anwendung - Currency Converter

The diagram illustrates the flow of data from a currency rate API to a currency converter application.

**API Response (Top Left):**

	code	name	rate
1	CHF	Swiss Franc	1.15891
2	EUR	Euro	1
3	GBP	British Pound	0.87736
4	USD	US Dollar	1.17131

**Currency Calculator (Bottom Left):**

**Currency Calculator**  
A simple Currency Calculator Service  
[Contact the developer](#)

**internal**

Method	Endpoint	Action
POST	/currencies	Add currency
DELETE	/currencies/{code}	Delete currency
PATCH	/currencies/{code}	Update currency rate
PUT	/currencies/{code}	Update currency

**public**

Method	Endpoint	Action
GET	/convert	Convert amount
GET	/currencies	Get currencies
GET	/currencies/{code}	Get currency

[ BASE URL: /v1 , API VERSION: 1.0.0 ]

**Currency Converter (Right):**

3kraft™

## Currency Converter

From Currency: Euro

To Currency: US Dollar

Amount: EUR 100.00

Result: USD 117.131

About 3kraft  
Copyright © 2018 3kraft IT GmbH & Co KG.



# RESTful Web Services



3kraft™

# RESTful Web Service

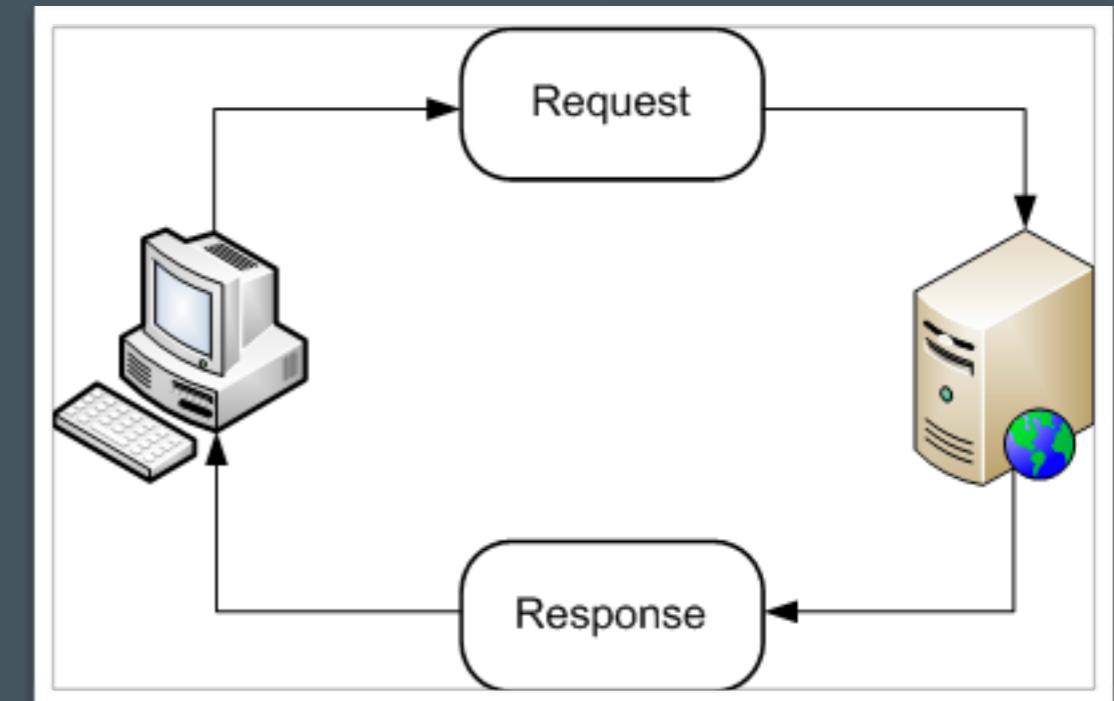
- REST ... Representational State Transfer
- “REST-compliant web services allow the requesting systems to access and manipulate textual representations of [web resources](#) by using a uniform and predefined set of [stateless](#) operations”

# Eigenschaften RESTful Web Services

- Zustandslos
- Transport Protokol:
  - HTTP
- Operationen:
  - GET, POST, PUT, ...
- Web Ressourcen:
  - URL's/URI's
- Datenprotokol:
  - JSON, XML, ...

# Eigenschaften HTTP

- Stateless
- Request-Response
- Metadata:
  - Request/Response Headers
- Implemented by:
  - Web Browser (User Agent)
  - Web Server
  - Kommandozeilen Tools
  - Development Libraries





# RESTful Resource URL's

- URL's identifizieren Objekte und Funktionen
  - Collections (Listen):
    - `http://api.service.com/resources?param=val`
    - zB: `http://api.service.com/v1/currencies?byRegion=EU`
  - Eindeutiges Objekt:
    - `http://service.com/.../resources/{id}`
    - zB: `http://service.com/v1/currencies/{currencyCode}`
  - Relationen:
    - `http://service.com/.../resources/{id}/relations`
    - `http://service.com/.../resources/{id}/relations/{relationsId}`
    - zB: `http://service.com/policies/privacy/translations/de`



3kraft™

# RESTful Operationen

URL	GET	PUT	PATCH	POST	DELETE
Collection: <a href="https://api.example.com/resources">https://api.example.com/resources</a>	Auflistung der Elemente	Ersetzen der kompletten Liste von Elementen	Typischerweise nicht benutzt	Anlegen eines neuen Elements in der Liste. Liefert üblicherweise die Id für das neue Element	Löscht die komplette Liste
Element: <a href="https://api.service.com/resources/item10">https://api.service.com/resources/item10</a>	Holen des Elements	Ersetzen des Elements oder neu Anlegen falls es nicht existiert	Ersetzen von speziischen Attributen eines Elements	Typischerweise nicht benutzt.	Löscht ein Element aus der Liste



# Demo - Simple REST Service mit NodeJS



# REST Contract

- Beschreibt die Schnittstelle
- mittels Beschreibungssprache:
  - RESTful API Description Langauge (analog zu WSDL)
    - [swagger.io](#) (Open API Spec)
    - RAML



# Contract Driven Development

- Interface-First Development
  - Die Schnittstelle existiert vor dem Code und nicht umgekehrt
  - Schnittstelle wird mit Beschreibungssprache umgesetzt und nicht in Code
  - Code Generieren für
    - Server
    - Client
  - Mocking
  - Dokumentation
- Clients (Konsumenten) und Server (Implementierungen) können getrennt von einander umgesetzt werden, sobald die Schnittstelle existiert
- Typischerweise von oder gemeinsam mit der Fachabteilung umgesetzt

# REST API mit Open API Spec

```
1  swagger: '2.0'
2
3  info:
4    title: Redbull Policy Center Internal API
5    description: The Redbull Policy Center Internal API provides restful
       operations for SPA.
6    version: 1.3.0
7
8  basePath: /internal/api/policies
9
10  produces:
11    - application/json
12
13  consumes:
14    - application/json
15
16  tags:
17    - name: policies_internal
18    | description: Policy Center Internal API
19
20  paths:
21    /projects:
22      get:
23        summary: List projects
24        tags:
25          - policies_internal
26        parameters:
27          - in: query
28            name: unpublished
29            description: if true, returns only projects with unpublished
               policy translations.
30        type: boolean
31        required: false
32        default: false
33        x-timeout: 60000
34        responses:
35          200:
36            description: a list of projects
37            schema:
38              type: array
39              description: A list of project objects.
40              items:
41                $ref: '#/definitions/Project'
```

## Redbull Policy Center Internal API

1.3.0

[ Base URL: /internal/api/policies ]

The Redbull Policy Center Internal API provides restful operations for SPA.

### policies\_internal Policy Center Internal API

- |               |   |  |
|---------------|---|--|
| <b>GET</b>    | /projects   | List projects                                    |
| <b>POST</b>   | /projects   | Create project.                                  |
| <b>GET</b>    | /projects/{projectId}                                   | Retrieve project by id.                          |
| <b>PUT</b>    | /projects/{projectId}                                   | Create or update project.                        |
| <b>DELETE</b> | /projects/{projectId}                                   | Deletes a project.                               |
| <b>GET</b>    | /projects/{projectId}/policies                          | Retrieve all policies assigned to a project      |
| <b>GET</b>    | /projects/{projectId}/policies/{policyId}/publishedUrls | Retrieve published urls for a policy and project |



# Demo Swagger Editor - Currency Converter API



3kraft™

# Code Generierung Server/Client

Generate Server ▾	Generate Client ▾	Generate Client ▾			
ada-server	jaxrs-resteasy	rust-server	ada	elixir	jmeter
aspnetcore	jaxrs-resteasy-eap	scala-lagom-server	akka-scala	elm	kotlin
erlang-server	jaxrs-spec	scalatra	android	erlang-client	lua
finch	lumen	sinatra	apex	flash	objc
go-server	msf4j	slim	bash	go	perl
haskell	nancyfx	spring	closure	groovy	php
inflector	nodejs-server	undertow	cpprest	haskell-http-client	powershell
java-pkmst	php-silex	ze-ph	csharp	html	python
java-play-framework	php-symfony		csharp-dotnet2	html2	qt5cpp
java-vertx	pistache-server		cwiki	java	r
jaxrs	python-flask		dart	javascript	ruby
jaxrs-cxf	rails5		dynamic-html	javascript-closure-angular	rust
jaxrs-cxf-cdi	restbed		eiffel	jaxrs-cxf-client	scala
					typescript-node



# Demo - Create Currency Converter API Server for Spring Boot



3kraft™

# API Documentation/Mocking/Testing

GET /currencies/{code} Get currency

**Implementation Notes**  
Gets a currency by code

**Response Class (Status 200)**  
Successfully retrieved currency

**Model** | Example Value

```
Currency {  
    code (string),  
    name (string),  
    rate (number)  
}
```

**Response Content Type** application/json

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
code	GBP	A currency code	path	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Currency Not Found		

[Try it out!](#) [Hide Response](#)

**Curl**

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/v1/currencies/GBP'
```

**Request URL**

```
http://localhost:8080/v1/currencies/GBP
```



# Demo - Test Mock



# RESTful Web Services Implementieren

- Authentication/Authorization
- Request Verarbeitung
  - Convertierung Eingabe Parameter
  - Validierung Eingabe Parameter
- Senden der Response
  - Metadaten (Response Header)
  - Status (HTTP Status Codes)
  - Caching
- Transaktionen

# API Authentication/Authorization

- Wenn möglich auf Applikations-/Service-Ebene
  - nicht in jeder Resource implementieren
  - typischerweise in einem vorgeschaltetem API-Gateway implementiert
- Arten der Authentifizierung:
  - API Key
    - JSON Web-Token
  - Basic Authentication
  - OAuth
    - OpenID

```
securityDefinitions:  
  JWT:  
    description: "JWT authorization header"  
    type: apiKey  
    name: Authorization  
    in: header  
    scopes:  
      admin: Admin scope  
      user: User scope
```

# Validierung/Konvertierung

- Validierung/Konvertierung wenn möglich auf Service Ebene
- Implementierung arbeitet mit Domain Objekten und nicht mit Protokol-Daten

```
@ApiOperation(value = "Add currency", nickname = "addCurrency", notes = "Creates a new currency", response = Currency.class, authorizations = {  
    @Authorization(value = "basicAuth")  
}, tags={ "internal", })  
@ApiResponses(value = {  
    @ApiResponse(code = 200, message = "The created currency", response = Currency.class),  
    @ApiResponse(code = 400, message = "Invalid Currency/Already exists") })  
@RequestMapping(value = "/currencies",  
    produces = { "application/json" },  
    consumes = { "application/json" },  
    method = RequestMethod.POST)  
ResponseEntity<Currency> addCurrency(@ApiParam(value = "A currency object" ,required=true ) @Valid @RequestBody Currency body);
```

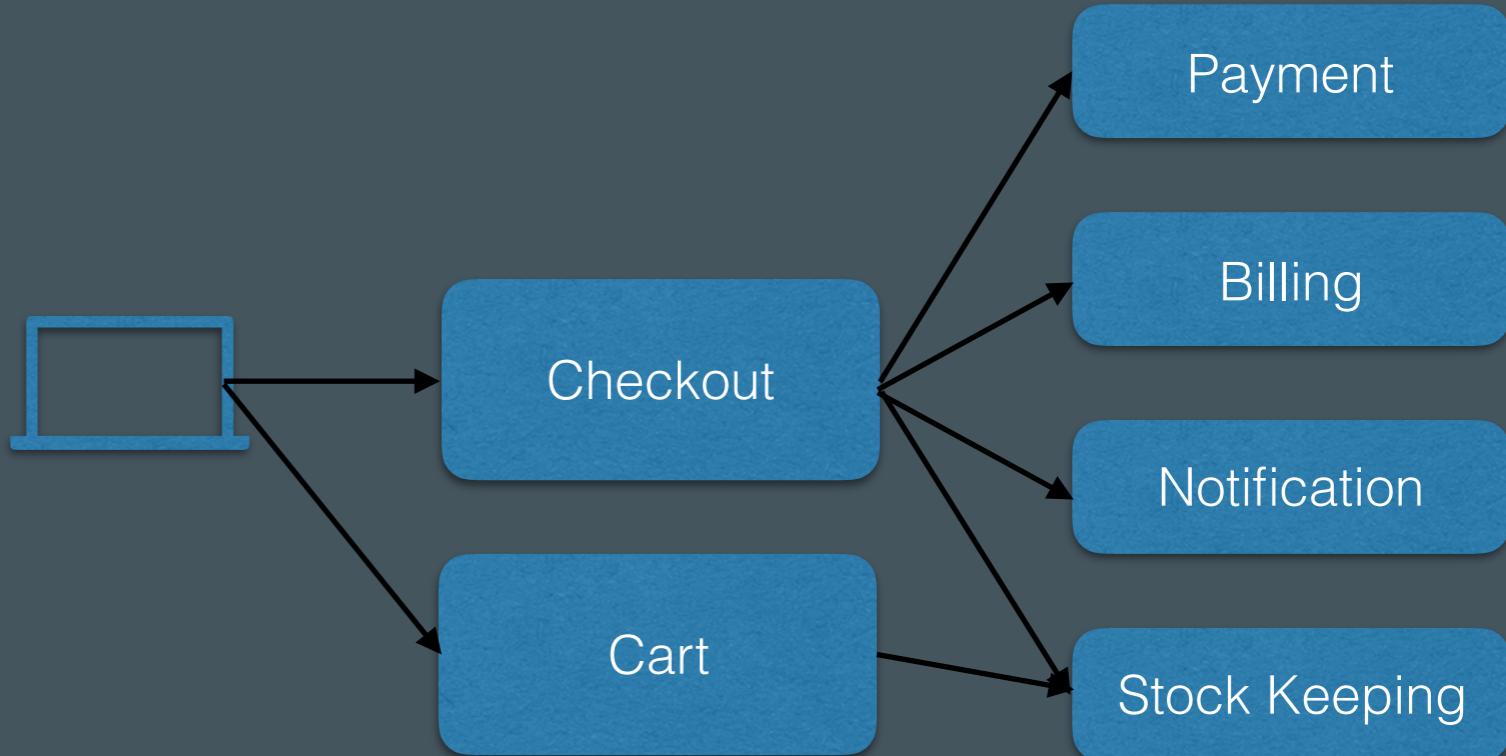
# Response Senden

- Verarbeitungserfolg/-fehler werden mittels HTTP Status Code an den Client gesendet
- Metadaten für die weitere Verarbeitung der gesendeten Daten als HTTP-Header
- Caching Header für Performance Optimierung

```
if (currency != null) {  
    return new ResponseEntity<Currency>(currency, HttpStatus.OK);  
}  
else {  
    return new ResponseEntity<Currency>(HttpStatus.NOT_FOUND);  
}
```

# Transaktionen

- Ausschließlich innerhalb einer API Resource (API Endpoint)
- Wenn möglich deklarativ
- REST API Ressourcen sind nicht transaktional (kein Commit, kein Rollback)
  - Möglicherweise alternative Konzepte notwendig
    - Message Queueing
    - Eventual Consistency





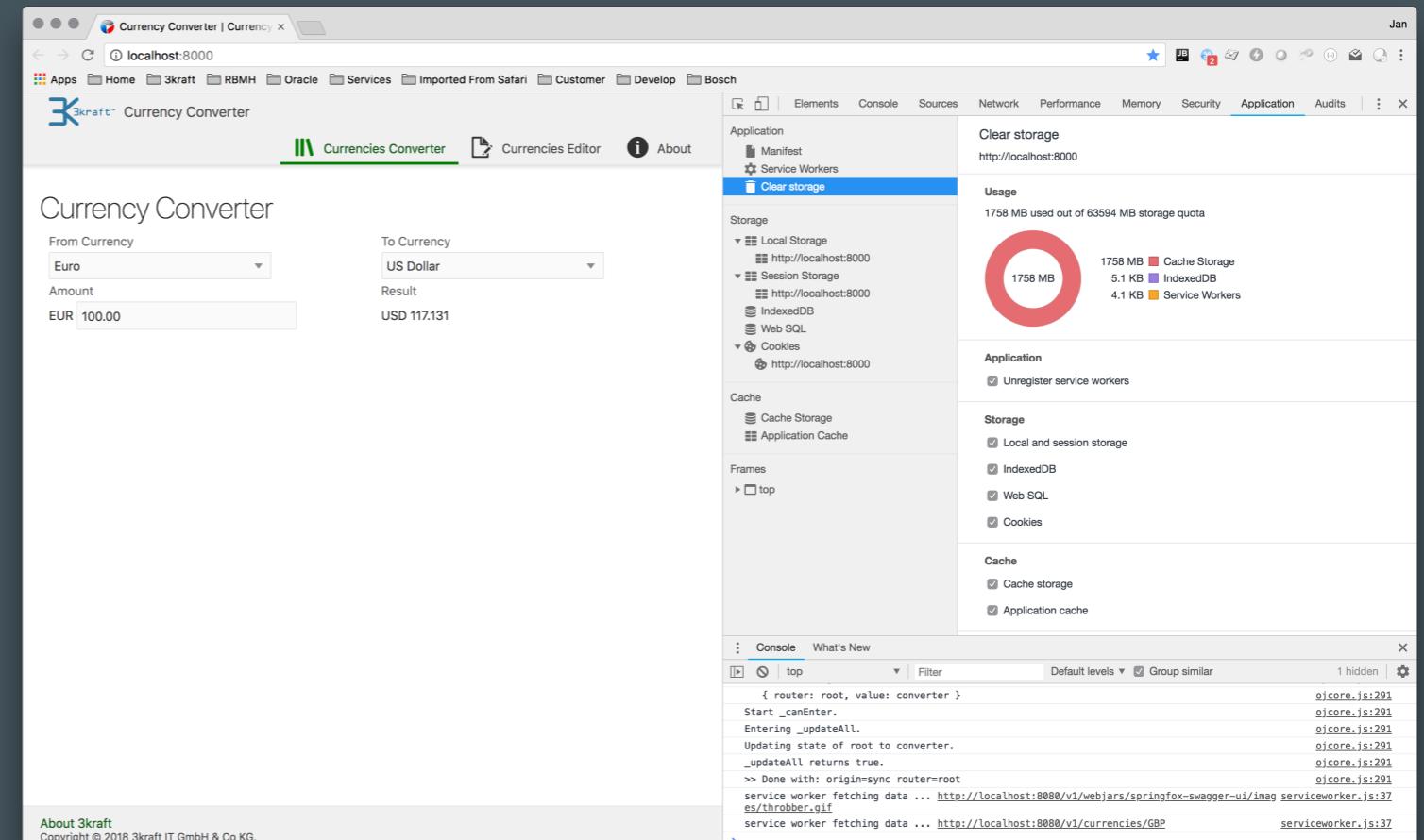
# Demo - Implementierung Currency Converter API in Spring Boot



# Browser Frontends

# Web Browser

- Laufzeit Umgebung für Webanwendung
  - Template Rendering Engine
    - Styling Engine
  - Formular Framework
  - Javascript Runtime
  - Netzwerk Implementierung
    - TCP, HTTP, HTTP/2, SSL/TLS
  - Database Engines
  - Development Tools (Editing, Testing, Debugging)



# Elemente einer Web Anwendung

- Struktur
  - HTML Hyper Text Markup Language
- Styling
  - CSS Cascading Stylesheets
- UI/DOM Manipulation
  - Javascript
- Client Logik
  - Javascript

# HTML

- HTML . . . Auszeichnungssprache zum Beschreiben der Struktur eines Dokumentes oder User Interfaces

```
<!doctype html>
<html lang="en">

  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

    <title>Simple Webapp</title>
  </head>

  <body>
    <div class="container-fluid">
      <h1>Greeter</h1>
      <div class="row">
        <div class="col-12 col-sm-6 text-align-left">
          <input data-bind="value: name">
          <button data-bind="click: greet">greet</button>
        </div>
        <div class="col-12 col-sm-6">
          <p>Hello <strong data-bind="text: bigName"></strong></p>
        </div>
      </div>
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.4.2/knockout-min.js"></script>

    <script src="js/main.js"></script>
  </body>

</html>
```

# CSS

- Definiert Styling Anweisungen/Regeln für die Elemente im Markup
  - Regeln werden kaskadierend aufgelöst
    - alle passenden Regeln für ein Element werden aufgelöst
    - genauer passende Regeln überschreiben, wenig genaue Regeln
  - Styling des Markups
    - auf Element/Tag Typ Ebene (zb: für <H1>)
    - auf Element Ebene (by Id)
    - als Styling Klassen
      - Gruppierung von Style-Anweisungen die unterschiedlichen Elementen zugewiesen werden können
      - Mehrere Klassen können auf ein Element angewendet werden

```
body {  
    background-color: #FFF;  
}  
  
#title {  
    font-family: monospace;  
}  
  
.marginX {  
    margin-left: 1rem;  
    margin-right: 1rem;  
}  
  
strong.marginX {  
    margin-left: 3rem;  
    margin-right: 3rem;  
}  
  
h1 strong {  
    margin-left: 0rem;  
    margin-right: 0rem;  
}
```



Ekraft™

# CSS Anweisungen

- Farben
- Schriften
- Größen
- Abstände
- Textfluss
- Scroll-Verhalten
- Animationen
- Hintergründe
- Bilder
- Positionen
- Platzhalter
- Rahmen
- etc ...



# CSS Frameworks

- Styles für all Basis Tags/Elemente
  - Basis für eigene Stylesheets
- Theme-Support
  - steuerbar über Variablen
- Umfangreiche Style-Klassen Bibliothek
- Support Responsive Design
  - Grid Systeme (960 Grid)
  - Responsive Utilities
  - Breakpoints für unterschiedliche Display Größen
- Typischerweise "Mobile First"

# Javascript

- Script Sprache zur Interaktion mit dem Browser bzw. geladene Webseite
- Interpretiert
- Dynamisch
- Typeless
- Funktional
  - Funktionen sind “1-Class Citizens”
- “Easy to Learn, Hard to Master”

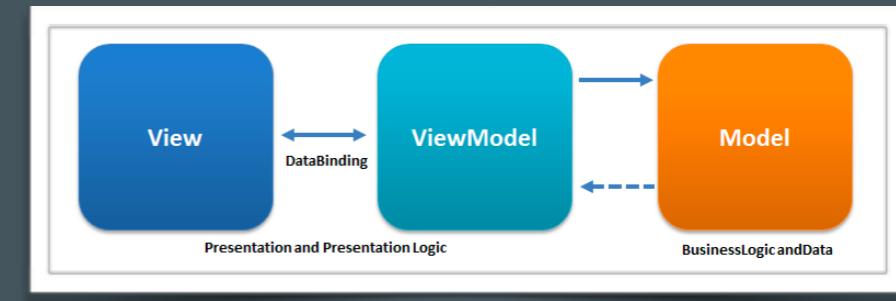
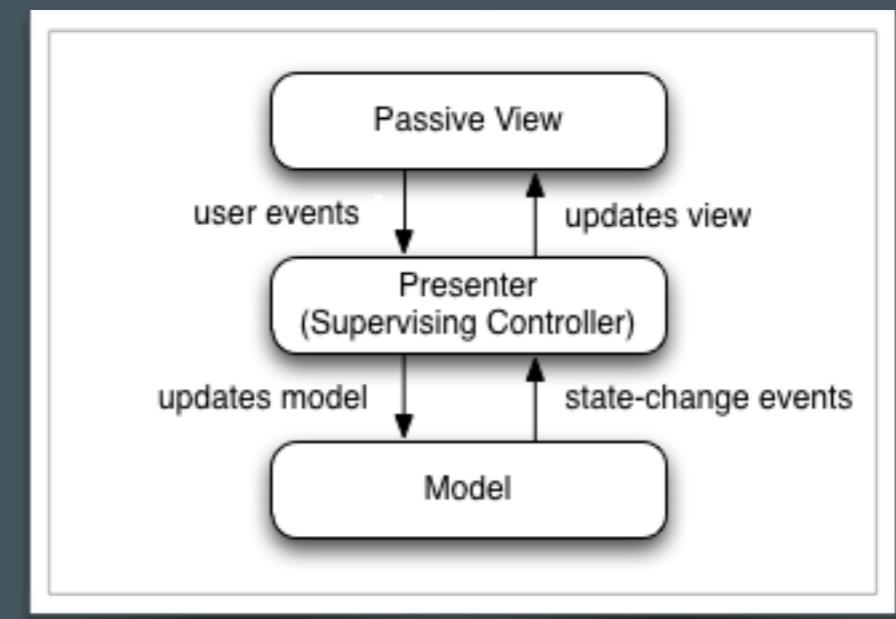
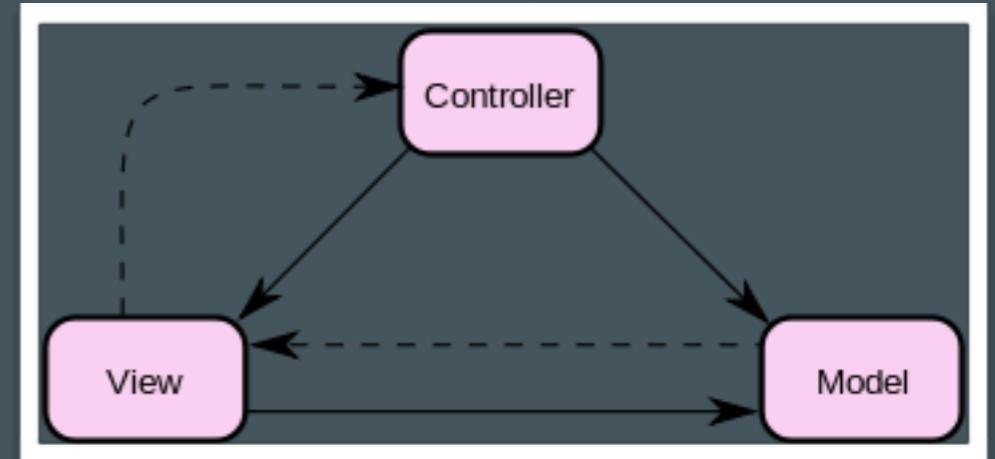
```
var ViewModel = function() {
    var self = this;
    self.name = ko.observable();
    self.bigName = ko.observable();

    self.greet = function() {
        self.bigName(self.name().toUpperCase());
    }
};

ko.applyBindings(new ViewModel());
```

# MVC/MVP/MVM

- Model-View-Controller
  - Model-View-Presenter
  - Model-View-Viewmodel
- Schichtenmodel
- Wiederverwendbarkeit





# Demo - Simple Web App



# User Interface Design



# Wireframes

A wireframe of a user registration form. It features a large red 'X' at the top right. Below it is a large input field with a red 'X'. The form contains fields for NAME (Karen Pollack), EMAIL (karen@app.com), and PASSWORD (\*\*\*\*\*). At the bottom, there is a progress indicator '1/2' and a 'NEXT STEP' button.

A wireframe of a currency converter interface. The header includes the Ekraft logo and a menu icon. The main section is titled 'Currency Converter' and shows two input fields, each with 'EUR' and a dropdown arrow, and a result field showing '0.00'. Below the result is a 'Result' label and another '0.00' placeholder.



# Demo - Freehand



Ekraft™

# User Interface Mockups

The image displays two side-by-side user interface mockups for the Ekraft application, showing the mobile version on the left and the web version on the right.

**Mobile Version (Left):**

- Header:** Weltreise 2018, Offen (90), Erledigt (5), Bearbeiten.
- Section: Zugewiesene Einträge**
  - Zelt (checkbox)
  - Ghettoblaster (checkbox)
  - Holzkohlegrill (checkbox)
  - Solarladegerät (checkbox)
- Section: Dokumente/Geld**
- Section: Kleidung**
- Section: Kosmetik**
- Info Callout:** TO DO: Entscheidung, ob "Zuweisen" und "Zugewiesene Einträge" oder "Teilen" und "Geteilte Einträge" als Formulierung für diese Funktion verwendet wird.
- Text Block:** Der Listeninhaber sowie ggf. jeder Mitreisende (wenn der Eigentümer das entsprechende Häkchen in "Mitreisende verwalten" gesetzt hat) kann Einträge zuweisen. Dazu reicht es bereits, wenn der Mitreisende eingeladen wurde – er muss die Einladung noch gar nicht angenommen haben. Der Zuweisende kann Items einem, mehreren oder auch allen Usern zuweisen, inklusive sich selbst.  
Zugewiesene Einträge erscheinen in einer eigenen "virtuellen Kategorie", die immer an erster Stelle angezeigt wird, nicht verschiebar und ist standardmäßig aufgeklappt. Alle Mitreisenden können alle zugewiesenen Einträge sehen, auch solche, die ihnen nicht zugewiesen sind. Nur die Mitreisenden jedoch, denen ein Eintrag zugewiesen ist, können diesen auch abhaken, für alle anderen ist dies deaktiviert.
- Text Block:** Verhalten im Bearbeiten-Modus: Die virtuelle Kategorie bleibt im Bearbeiten-Modus sichtbar. Der User kann jedoch nur Einträge selektieren, die er selbst erstellt hat. Diese kann er:
  - Zuweisen (= Zuweisungen ändern, z.B. weiteren Mitreisenden zuweisen oder Zuweisungen entfernen)
  - Verschieben (innerhalb der virtuellen Kategorie "Zugewiesene Einträge")
  - Löschen
- Text Block:** Änderungen an zugewiesenen Items werden sofort bei allen anderen Mitreisenden sichtbar/wirksam.
- Text Block:** Zugewiesene Items werden erst dann nach "Erledigt" verschoben, wenn alle User, denen das Item zugewiesen war, es abgehakt haben.
- Text Block:** Anzeige des Zuweisungs- und Erledigungsstatus:
  - Ist ein Item nur einem User zugewiesen, werden die Initialen in einem Kreis angezeigt (Farbe und Initialen decken sich mit denen in M7).
  - Ist ein Item mehreren Usern zugewiesen, wird im Kreis die Anzahl der Nutzer, die das Item bereits abgehakt haben, sowie die Anzahl der Nutzer, denen das Item insgesamt zugewiesen wird, angezeigt (Beispiel: Item ist fünf Nutzern zugewiesen, drei haben es bereits erledigt – es wird daher 3/5 angezeigt).
  - Zusätzlich wird bei Items, die mehreren Usern zugewiesen sind, ein Kreis um die Anzahl angezeigt. Dieser ist anfangs hellgrün und füllt sich im Uhrzeigersinn (beginnend bei der 12-Uhr-Position), je mehr User das Item abgehakt haben. Haben alle das Item abgehakt, ist der Kreis vollständig gefüllt.
  - Trifft es einen User das Anzahl-Kreis-Icon an, erscheint ein Info-Dropdown, welches auffliest, welche der User, denen das Item zugewiesen ist, es abgehakt haben und welche nicht. (Beim Initialen-Kreis-Icon entfällt diese Funktion, da das Item in diesem Fall ja nur einem User zugewiesen ist.)

The image shows the web-based interface for the Ekraft application, titled Weltreise 2018.

- Header:** Weltreise 2018, Liste durchsuchen ...
- Section: Listenansicht**
  - Wichtig (3)
  - Handgepäck (1)
- Section: Mitreisende**
  - Unterhaltung (2)
  - Aktivitäten (1)
- Section: Weitere Optionen**
- Section: Zugewiesene Einträge**
  - Zelt (checkbox)
  - Ghettoblaster (checkbox)
  - Holzkohlegrill (checkbox)
  - Solarladegerät (checkbox)
- Section: Meine anderen Listen**
  - Dienstreise Australien
  - Skiurlaub 2017
  - Camping Skandinavien
- Buttons:** Neue Liste erstellen, Vorlagen anzeigen
- Text Callout:** Verhalten sowie Anzeige des Erledigt-/Fortschritts wie bei der mobilen Version. Die Bearbeitungsmöglichkeiten werden nur für Einträge angezeigt, die vom User selbst angelegt/zugewiesen wurden.
- Text Callout:** ( weitere Kategorien und am Ende "Kategorie hinzufügen", wie in MD3.1)

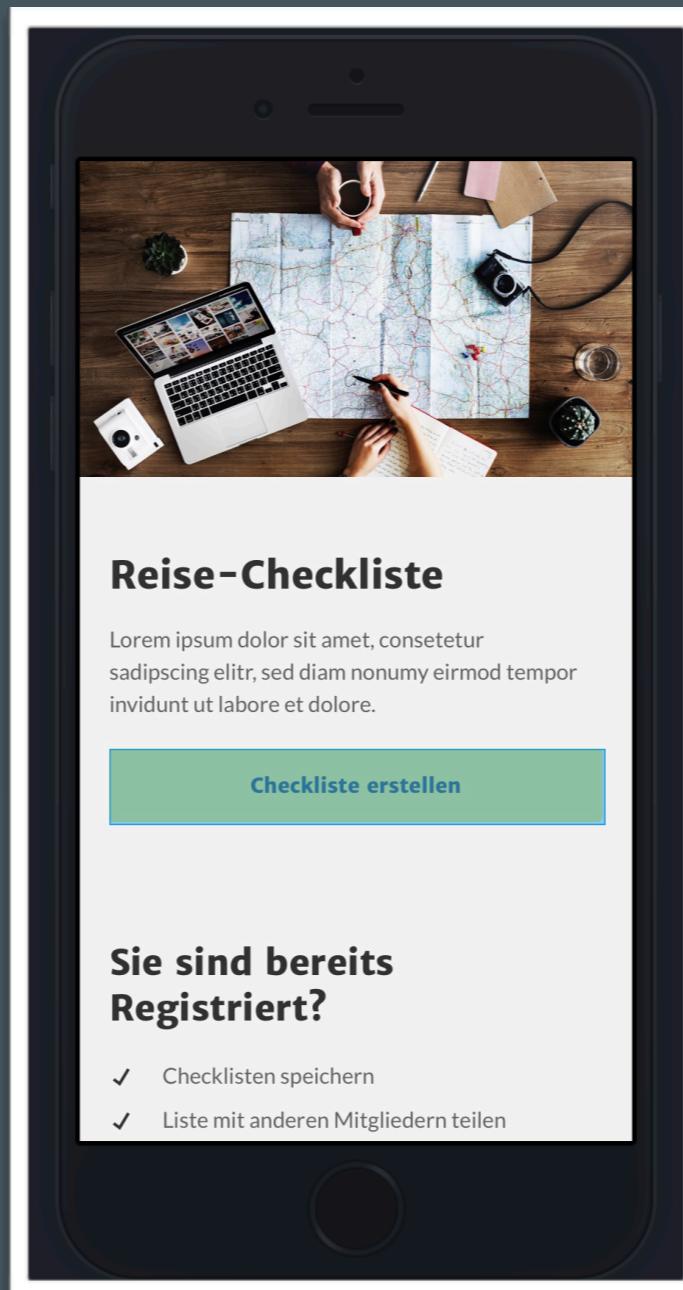


# Demo - Click Prototype



Ekraft™

# Click Prototype





# Browser App Frameworks



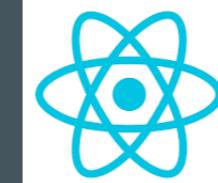
# Browser Application Frameworks

- Component Based
  - MVC/MVP/MVVM
    - Angular
    - KnockoutJS
  - Oracle JET
- UI State Pattern
  - React



*Knockout.*

ORACLE® JET





3kraft™

# Demo - Currency Converter mit Oracle JET