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

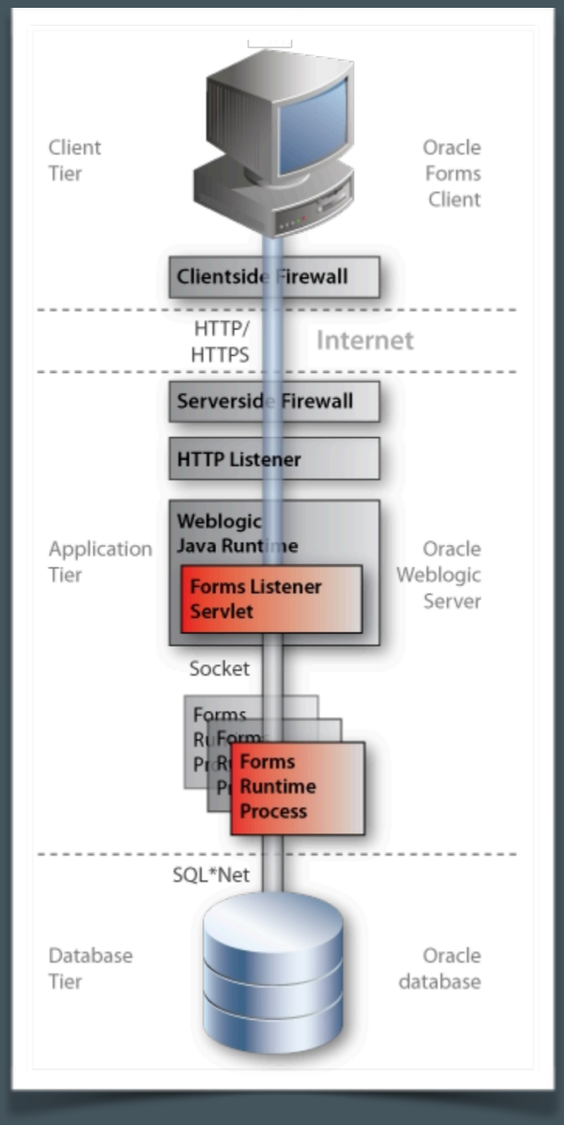
Anwendungs-Architekturen

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

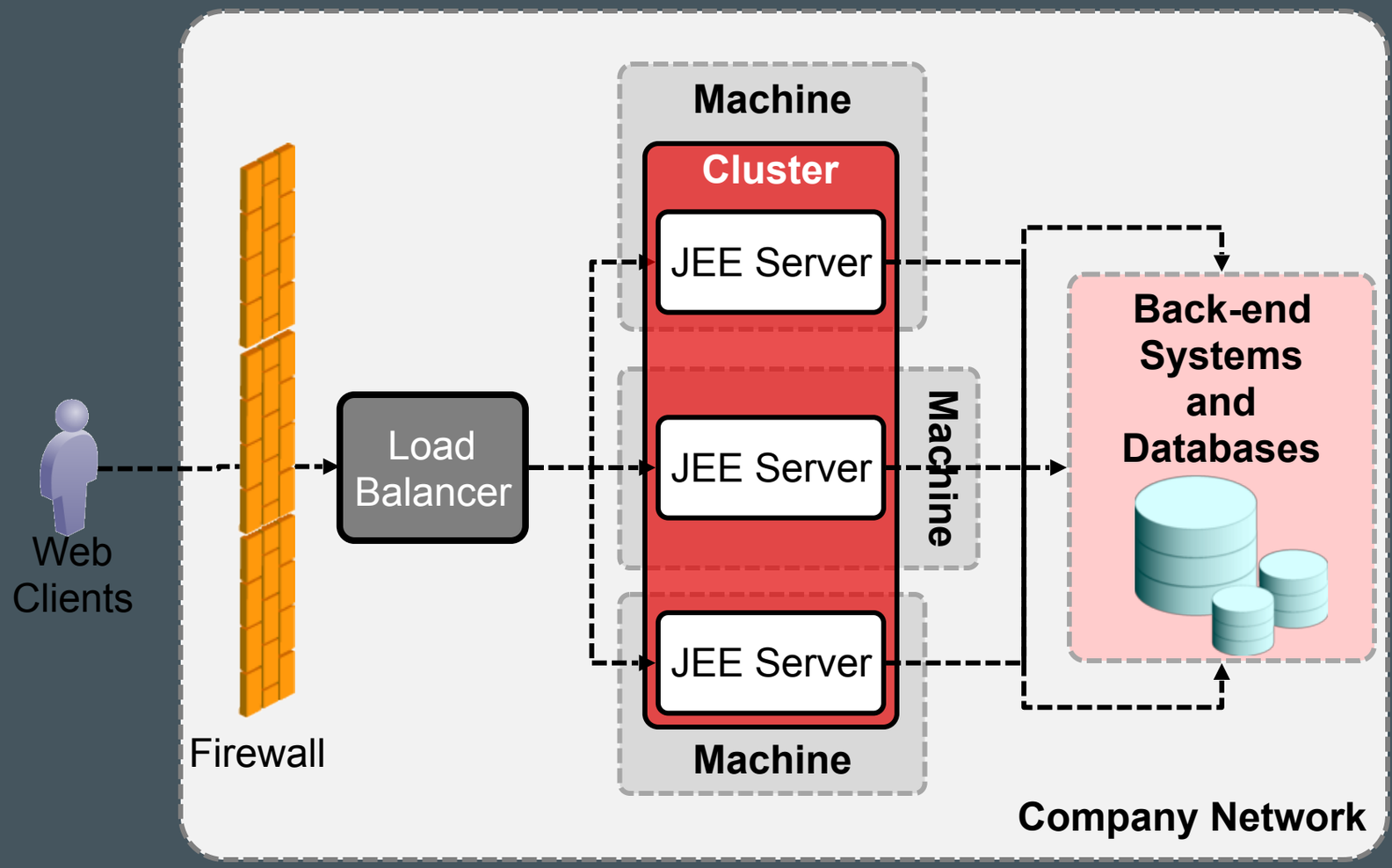
Client-Server Pros & Cons

- Geringe Einstiegs-Hürde
- Überschaubarer Funktionsumfang
- Einfaches State-Management
- Benutzer und Berechtigungsprüfung in der Datenbank

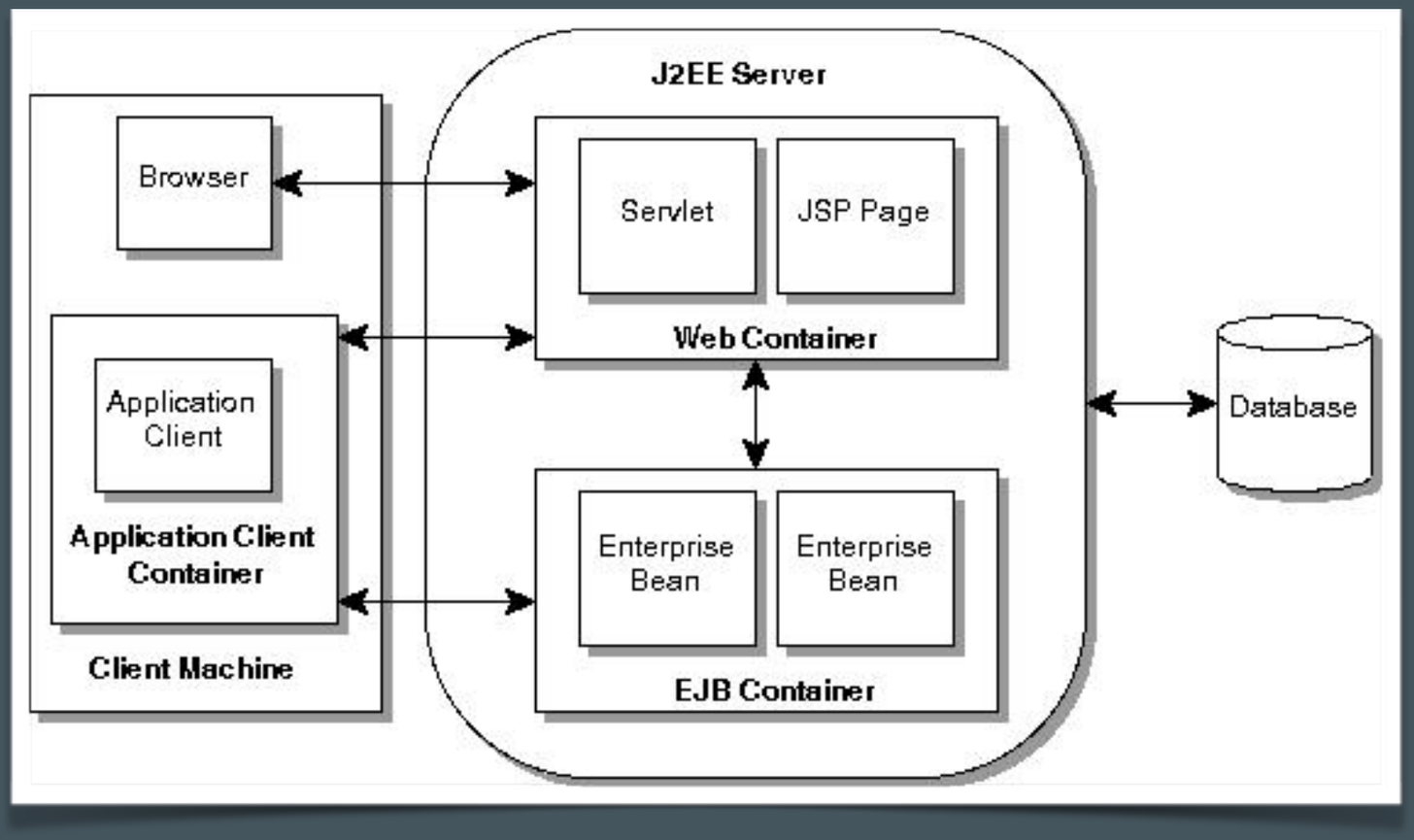
- Geringer Funktionsumfang
- Wenig flexibel
- Client benötigt
- Schlechte Skalierbarkeit
- Mobile Devices nicht unterstützt
- Kaum neue Features

Multi-Tier Architekturen

Multi-Tier Überklick (JEE)



JEE Architektur



Flipchart Multi-Tier Pros/Cons

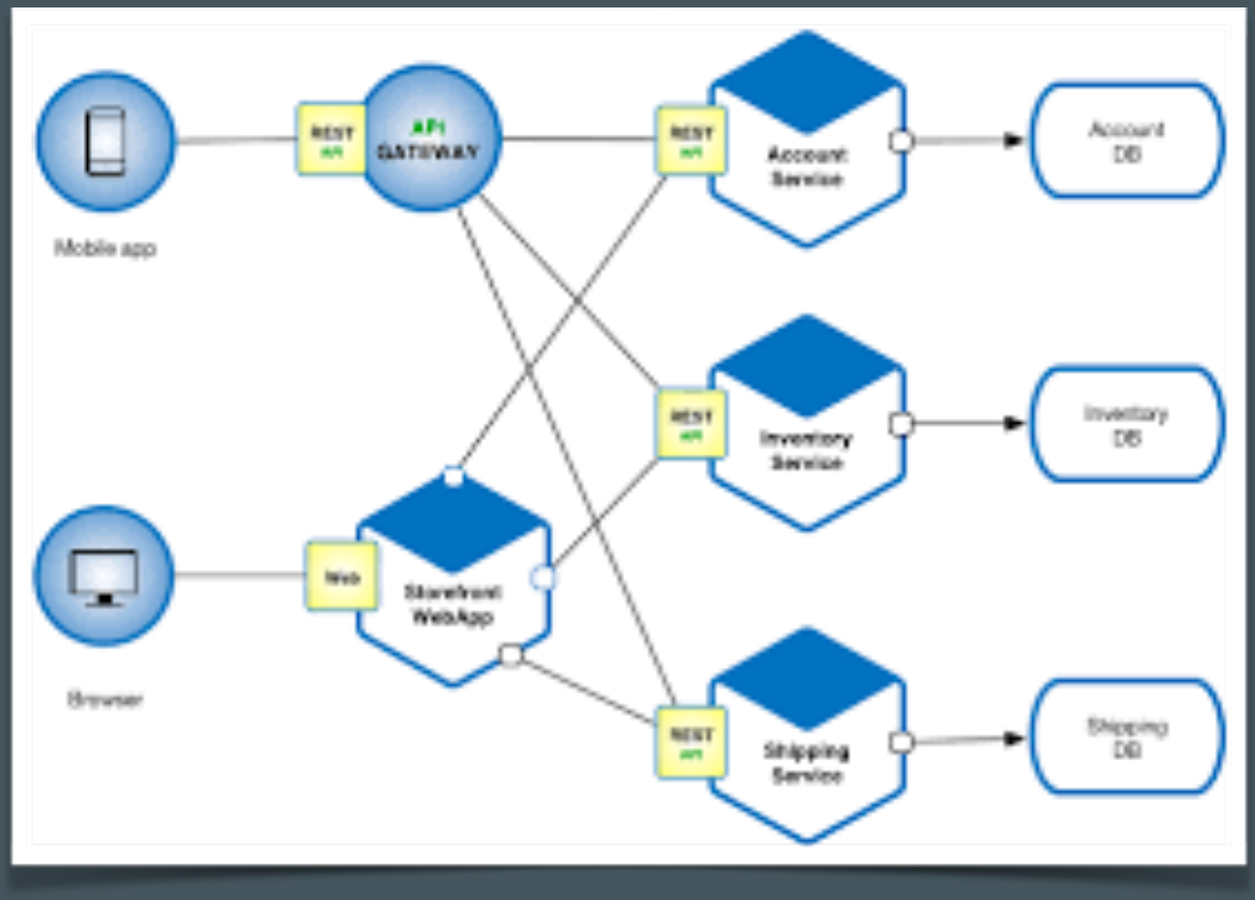
Multi-Tier Pros & Cons

- 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
- Versionsmanagement

Beispiel Anwendung - Currency Converter

	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

A simple Currency Calculator Service

Contact the developer

internal — Show/Hide | List Operations | Expand Operations

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

public — Show/Hide | List Operations | Expand Operations

GET	/convert	Convert amount
GET	/currencies	Get currencies
GET	/currencies/{code}	Get currency

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

Currency Converter

From Currency

Euro

To Currency

US Dollar

Amount

EUR 100.00

Result

USD 117.131

About 3kraft

RESTful Web Services

RESTful Web Service

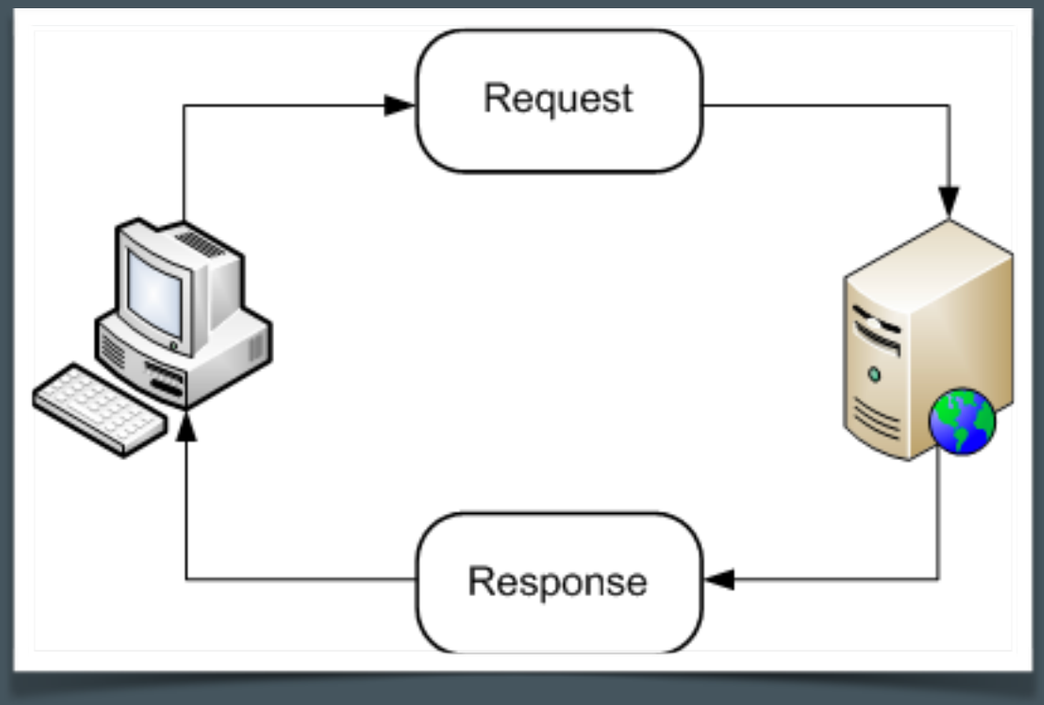
- REST … Representational State Transfer
- “REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations”

Eigenschaften RESTful Web Services

- Zustandslos
- Transport Protokol:
  - HTTP
- Operationen:
  - GET, POST, PUT, …
- Web Resourcen:
  - 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`

RESTful Operationen

URL	GET	PUT	PATCH	POST	DELETE
Collection: `https://api.example.com/resources`	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: `https://api.service.com/resources/item10`	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 wir 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

```yaml
swagger: '2.0'

info:
  title: Redbull Policy Center Internal API
  description: The Redbull Policy Center Internal API provides restful
    operations for SPA.
  version: 1.3.0

basePath: /internal/api/policies

produces:
  - application/json

consumes:
  - application/json

tags:
  - name: policies_internal
    description: Policy Center Internal API

paths:
  /projects:
    get:
      summary: List projects
      tags:
        - policies_internal
      parameters:
        - in: query
          name: unpublished
          description: if true, returns only projects with unpublished
            policy translations.
          type: boolean
          required: false
          default: false
      x-timeout: 60000
      responses:
        200:
          description: a list of projects
          schema:
            type: array
            description: A list of project objects.
            items:
              $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

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

Demo Swagger Editor - Currency Converter API

Code Generierung Server/Client

Generate Server ▾ Generate Client ▾

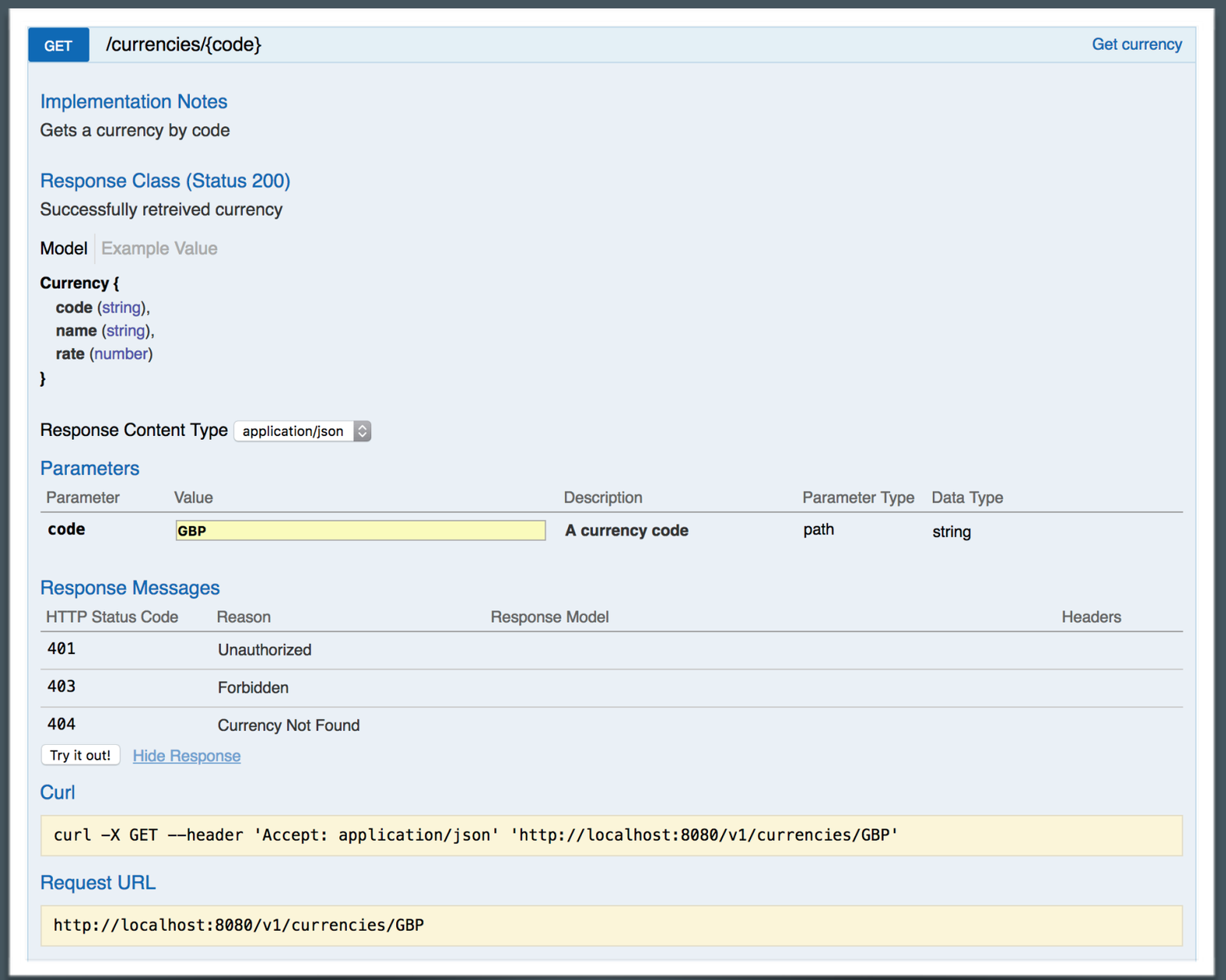
ada-server	jaxrs-resteasy	rust-server
aspnetcore	jaxrs-resteasy-eap	scala-lagom-server
erlang-server	jaxrs-spec	scalatra
finch	lumen	sinatra
go-server	msf4j	slim
haskell	nancyfx	spring
inflector	nodejs-server	undertow
java-pkmst	php-silex	ze-ph
java-play-framework	php-symfony	
java-vertx	pistache-server	
jaxrs	python-flask	
jaxrs-cxf	rails5	
jaxrs-cxf-cdi	restbed	

Generate Client ▾

ada	elixir	jmeter	scalaz
akka-scala	elm	kotlin	swagger
android	erlang-client	lua	swagger-yaml
apex	flash	objc	swift
bash	go	perl	swift3
clojure	groovy	php	swift4
cpprest	haskell-http-client	powershell	tizen
csharp	html	python	typescript-angular
csharp-dotnet2	html2	qt5cpp	typescript-angularjs
cwiki	java	r	typescript-aurelia
dart	javascript	ruby	typescript-fetch
dynamic-html	javascript-closure-angular	rust	typescript-jquery
eiffel	jaxrs-cxf-client	scala	typescript-node

Demo - Create Currency Converter API Server for Spring Boot

API Documentation/Mocking/Testing



GET /currencies/{code} — Get currency

Implementation Notes

Gets a currency by code

Response Class (Status 200)

Successfully retreived 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

```yaml
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);
```

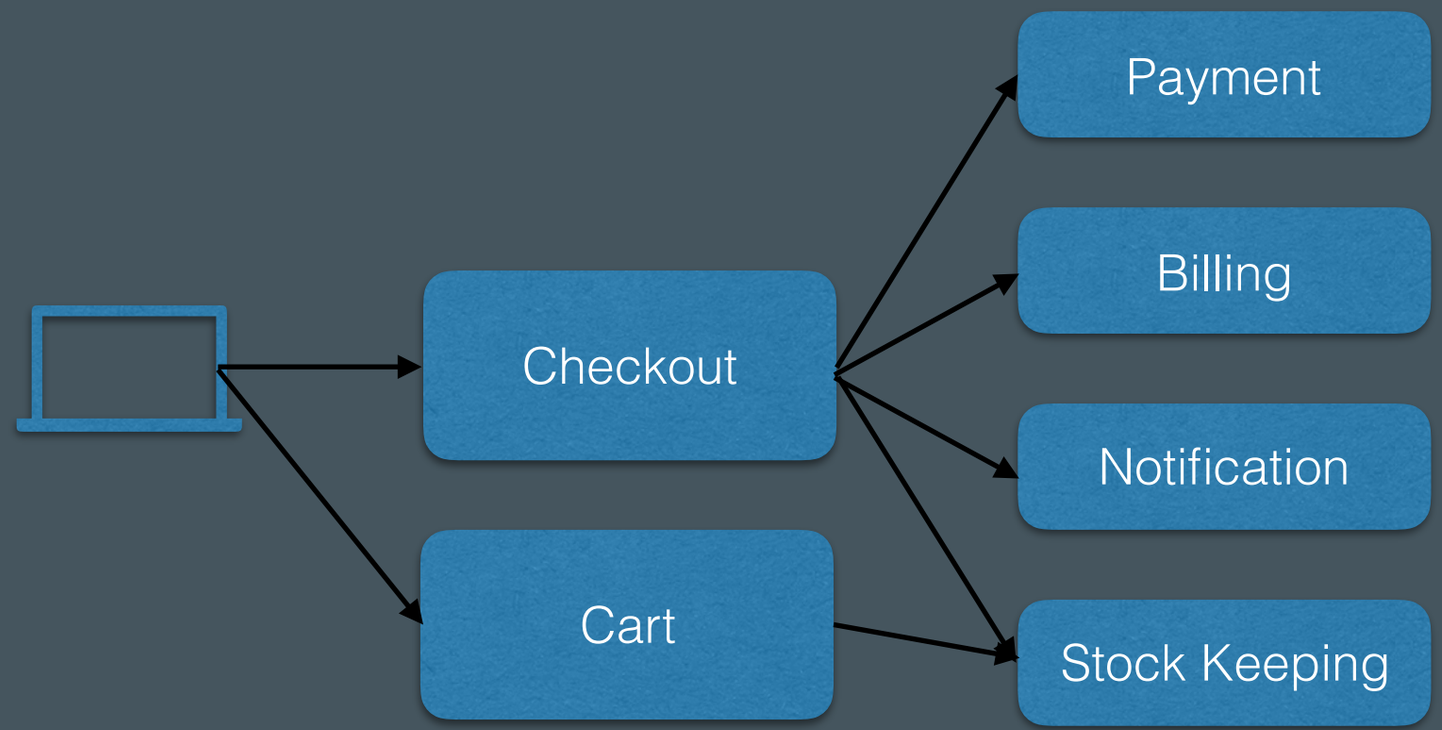
Response Senden

- 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 Resourcen sind nicht transaktional (kein Commit, kein Rollback)
  - Möglicherweise alternative Konzepte notwendig
    - Message Queueing
    - Eventual Consistency

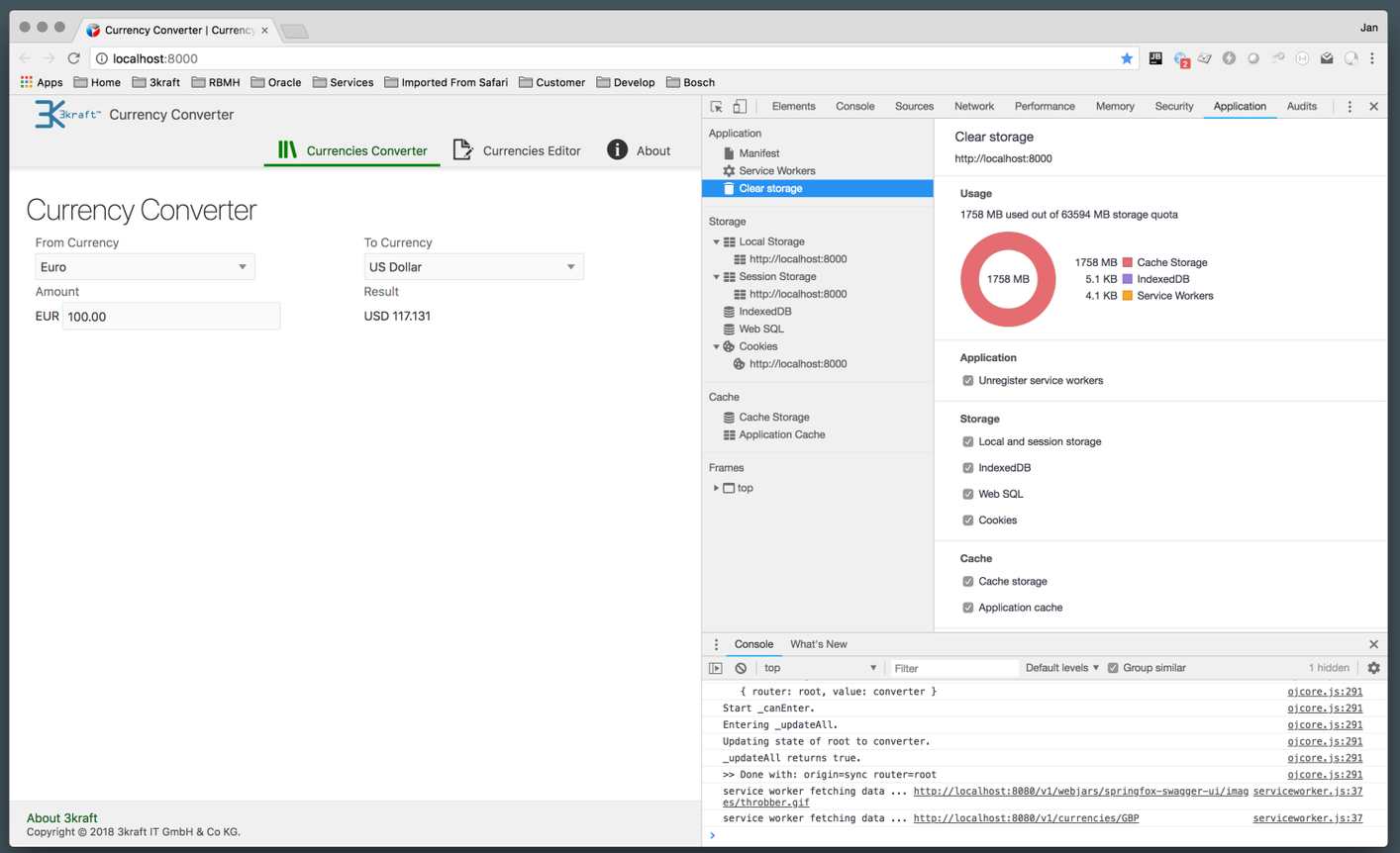


Demo - Implementierung Currency Converter API in Spring Boot

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

```html
<!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-nowrap">
                <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

```css
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;
}
```

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”

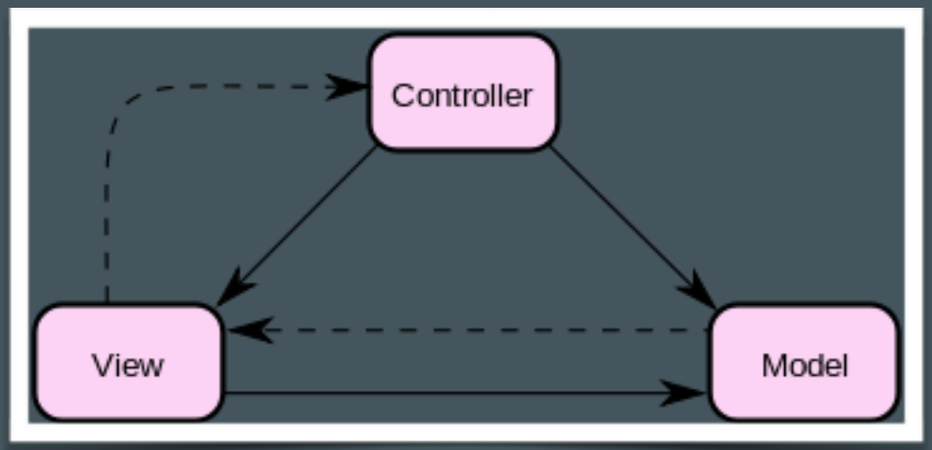
Javascript

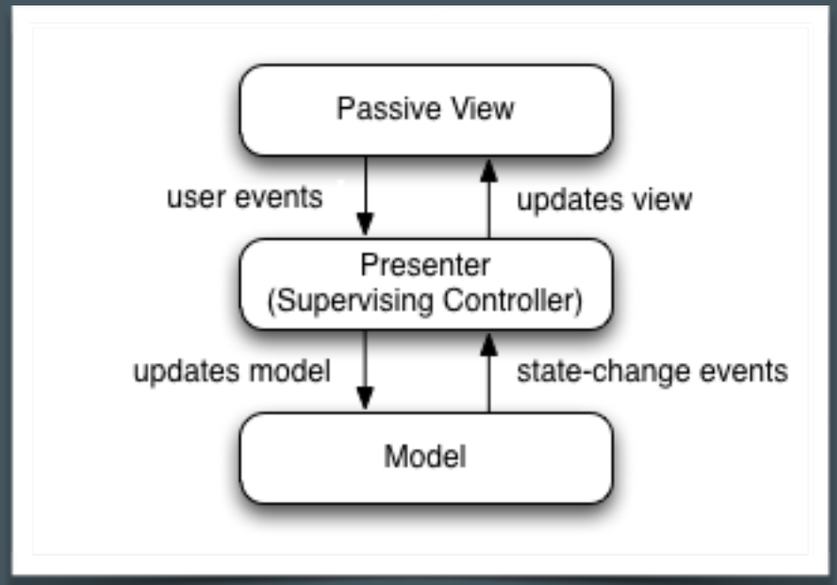
- Script Sprache zur Interaktion mit dem Browser bzw. geladene Webseite
- Interpretiert
- Dynamisch
- Typeless
- Funktional
  - Funktionen sind “1-Class Citizens”
- “Easy to Learn, Hard to Master”

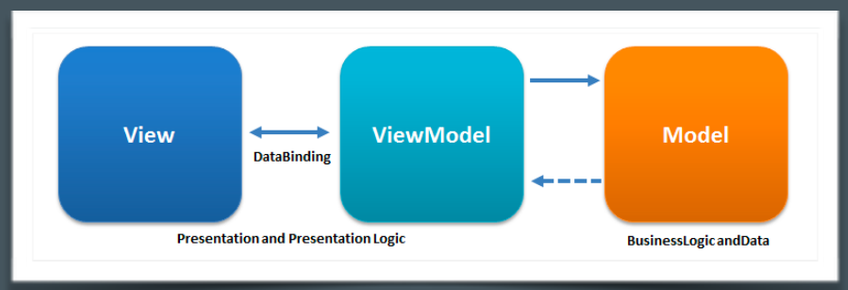
```
var ViewModel = function() {
    var self = this;
    self.name = ko.observable();
    self.bigName = ko.observable();

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

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

MVC/MVP/MVVM

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



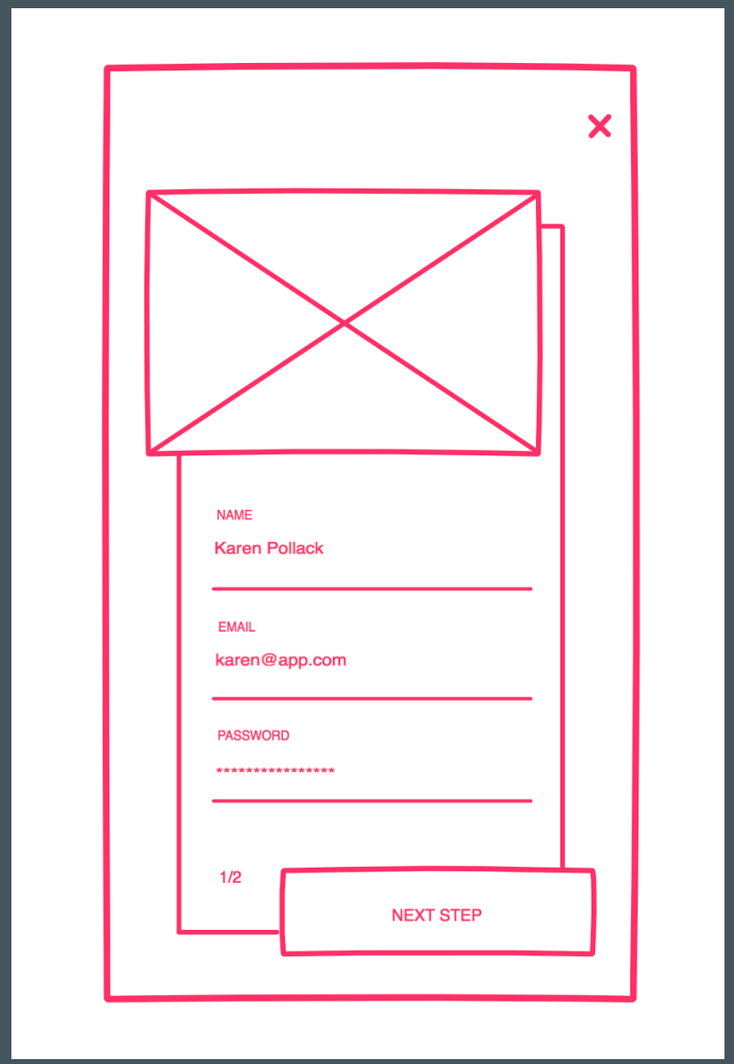




Demo - Simple Web App

User Interface Design

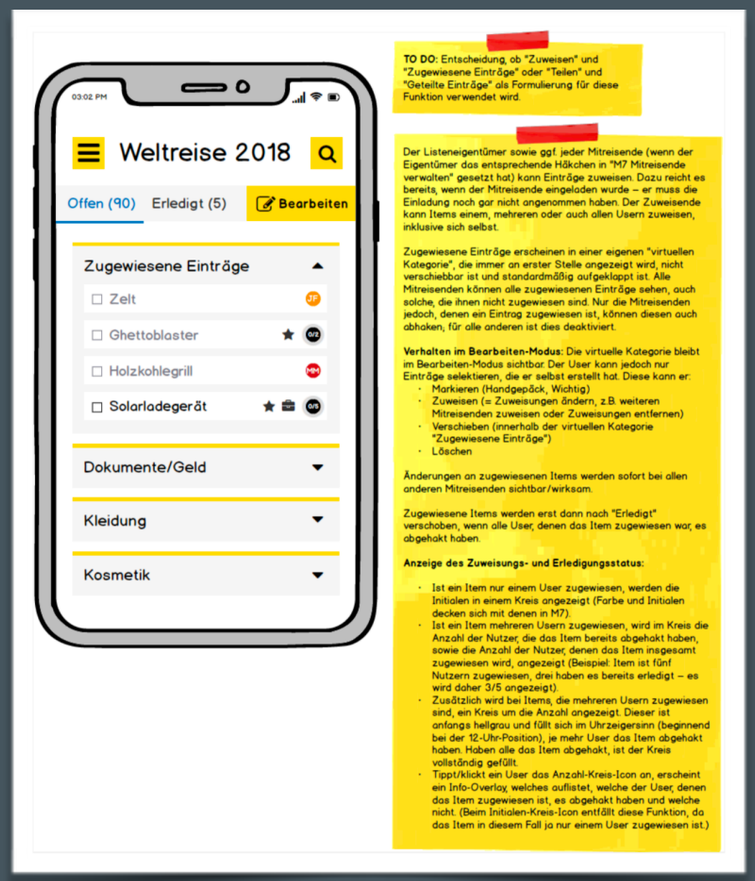
Wireframes

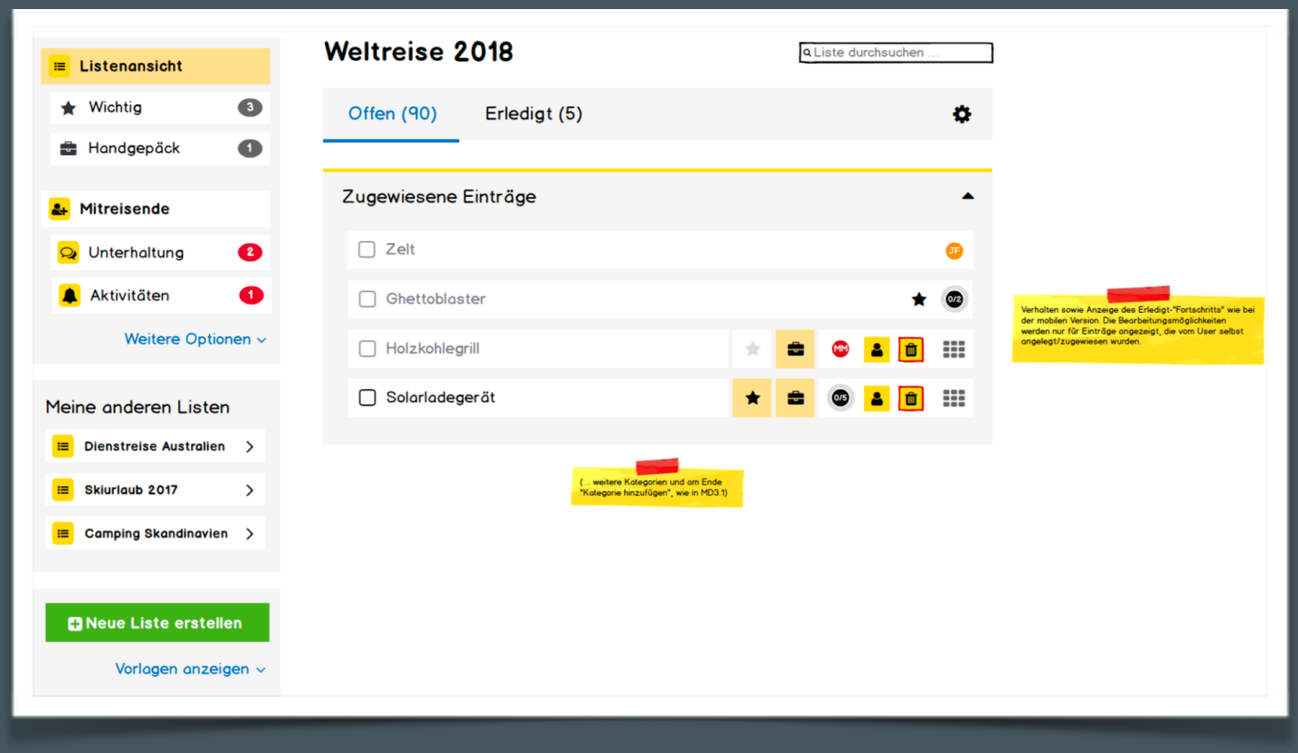

NAME
Karen Pollack
EMAIL
karen@app.com
PASSWORD
1/2
NEXT STEP


Currency Converter
EUR
EUR
0.00
Result
0.00

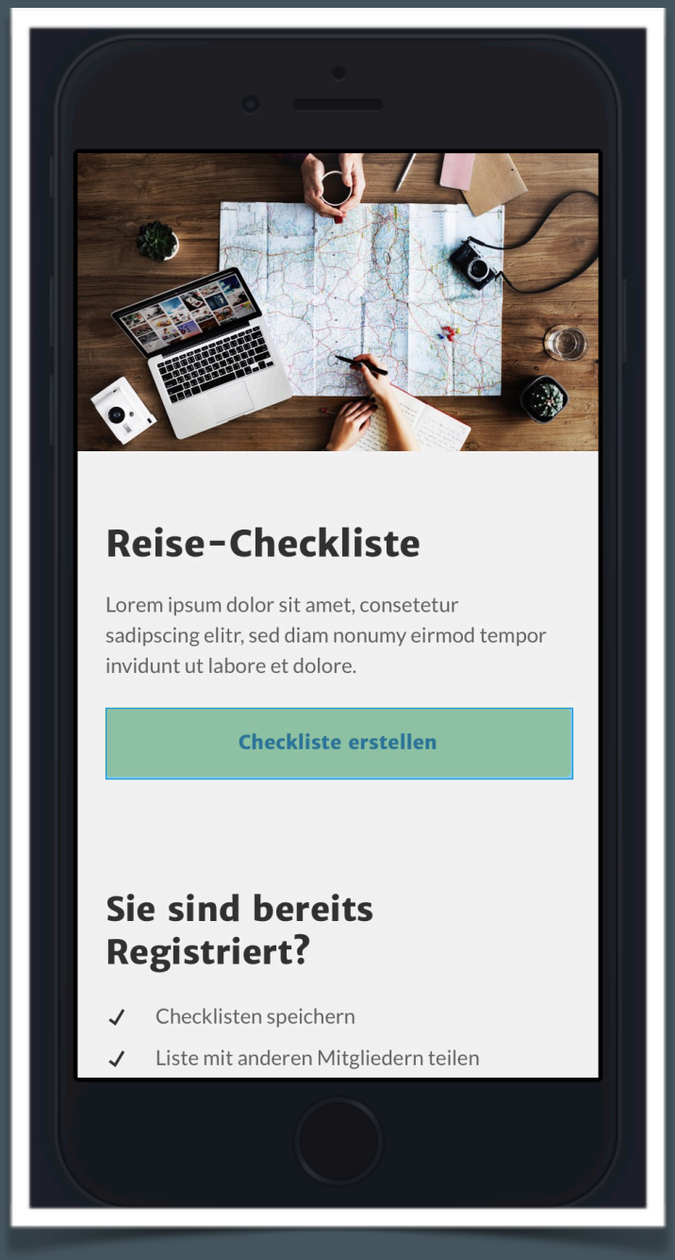
Demo - Freehand

User Interface Mockups





Click Prototype

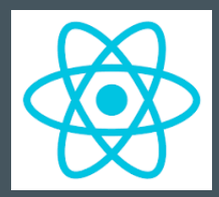


Demo - Click Prototype Adobe XD

Browser App Frameworks

Browser Application Frameworks

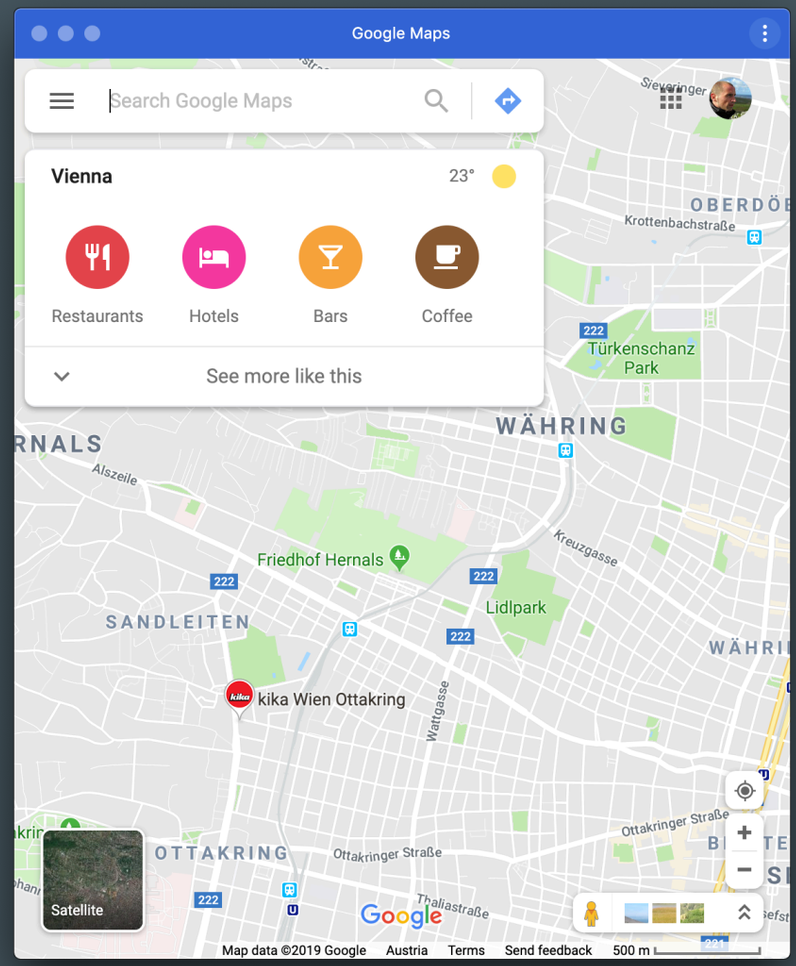
- Component Based
  - MVC/MVP/MVVM
    - Angular
    - KnockoutJS
      - Oracle JET
    - Vue.js
- UI State Pattern
  - React



Demo - Currency Converter mit Oracle JET

Progressive Web Apps (PWA)

- Offline
- Installierbar
- Native Device Funktionen
- Benachrichtigungen
- Background Tasks



Demo - Google Maps PWA