
CONSULTING
enpit

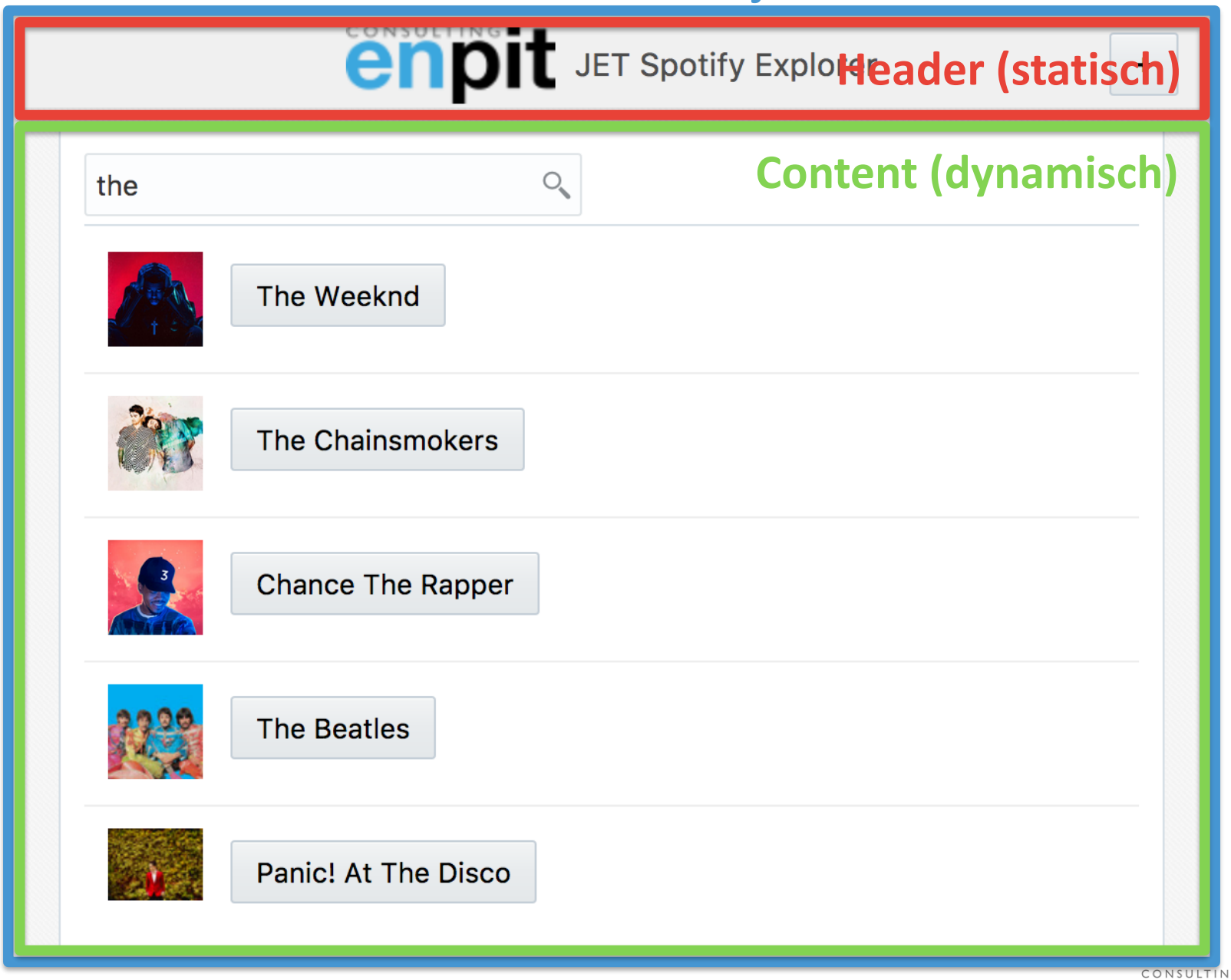
SESSION 2

MODULARISIERUNG
ROUTING
SERVICE-ANBINDUNG

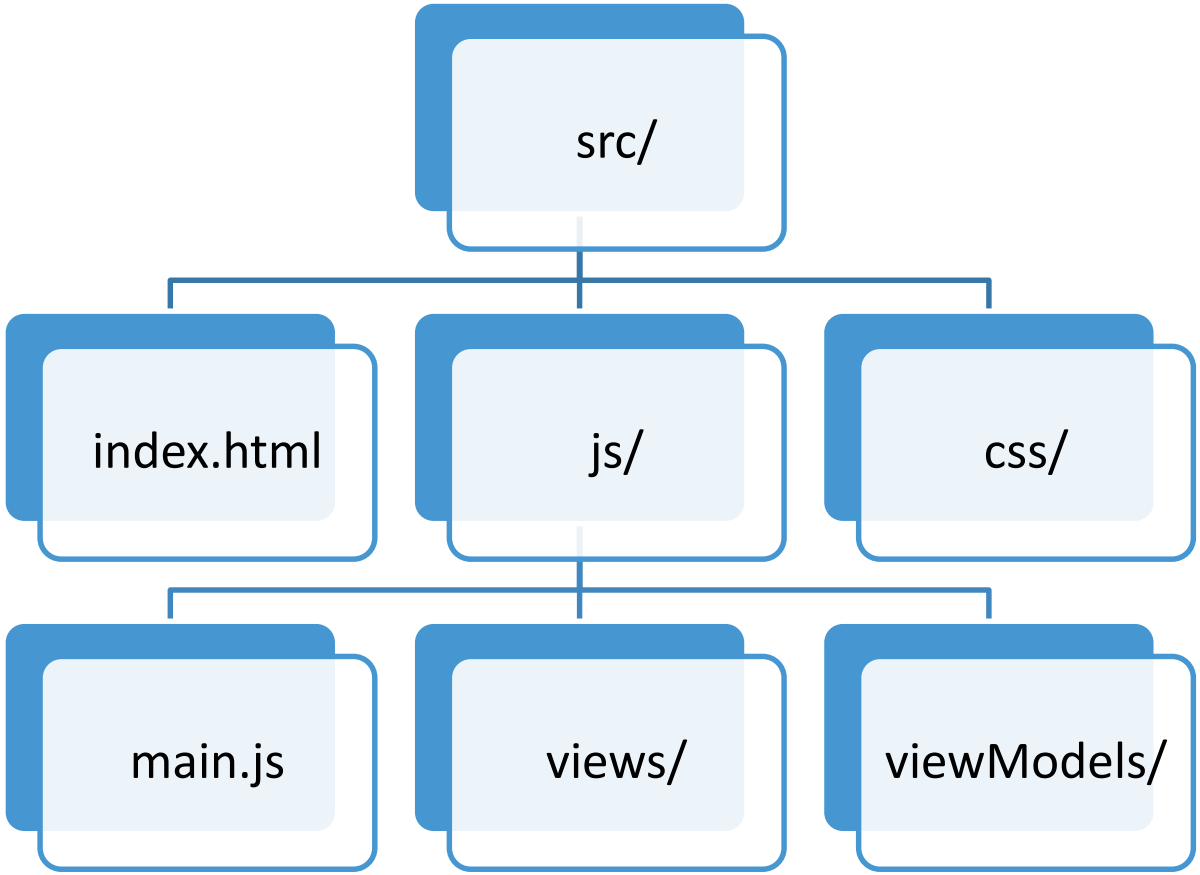
AGENDA

1. Modularisierung mit RequireJS & knockout-postbox
2. Routing
3. Service-Anbindung
4. Hands-On #2

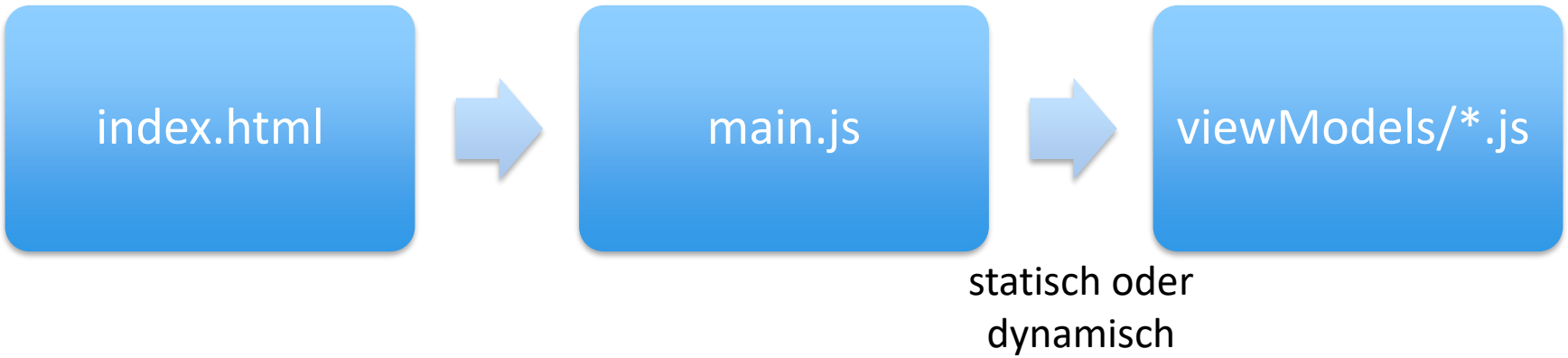
index.html & main.js



FILESYSTEM STRUKTUR



LOGISCHE STRUKTUR



MODULARISIERUNG IN ORACLE JET

- Warum Modularisierung?
  - Kapselung, Wiederverwendbarkeit, Wartbarkeit, …
- Oracle JET nutzt RequireJS
- RequireJS implementiert AMD
  - AMD: Asynchronous Module Definition

MODULE DEFINIEREN

- *define* mit Parameter:
  1. Dependencies
  2. Funktion
- Funktion gibt an, was das Modul tut
  - Parameter:
    Referenz auf geladene Dependency

```
1   define(
2     ['jquery'],
3     function ($) {
4       var setText = function (id, t) {
5           $('#'+id).text(t);
6       };
7       return setText;
8     }
9   );
```

VIEWMODEL DEFINIEREN

```
1   define(
2     ['ojs/ojcore', 'knockout', 'ojs/ojknockout'],
3     function (oj, ko) {
4       // Implementierung des Moduls:
5       var ViewModel = function () {
6         this.name = ko.observable('Janis');
7       };
8       // Rückgabewert
9       return new ViewModel();
10    }
11  );
```

REQUIRE-CALL IN MAIN.JS

1. Konfiguration von RequireJS per *requirejs.config*
   - Generiert bei Projekt-Setup
2. Laden des ersten Moduls per *require*
   - Lädt zunächst alle angegebenen Dependencies
   - Implementiert das Root-ViewModel

EINSPRUNGPUNKT
IN INDEX.HTML

```
<script
  type="text/javascript"
  data-main="js/main"
  src="js/libs/require/require.js">
</script>
```

RICHTIGE COOKBOOK VERWENDUNG

- `define` statt `require` verwenden (ojModule Komponenten)
- Keine applyBindings Calls (das übernimmt der *ojRouter*)
  - Stattdessen den *ViewModel* Konstruktor returnen
- Existierendes Modul: Nur function Body übernehmen

COOKBOOK VERWENDUNG – BEISPIEL

```
define(['ojs/ojcore', 'knockout', …],
function(oj, ko, …) {
    function ViewModel() {
        var self = this;
        self.answer = ko.observable(42);
    };
    return ViewModel;
});
```

PROBLEM: KOMMUNIKATION ZWISCHEN VIEWMODELS

- Knockout bietet *dataFor* Funktion
- `ko.dataFor($('#artist')[0]).name;`
- **Problem: Abhängig von Darstellung (ID)**
- **Viel Boilerplate nötig um Fehler abzufangen**
- Vgl. Geertjan´s Blog “Intermodular Communication in Oracle JET”

LÖSUNG: KNOCKOUT-POSTBOX

- Library für entkoppelte ViewModel Kommunikation
- Setzt auf Knockouts nativer pub/sub Mechanik auf
- Allgemein: *subscribe / publish*
- Observables: *subscribeTo / publishOn*

KNOCKOUT-POSTBOX
BEISPIEL

```
// search.js

self.selectedArtist =
ko.observable().publishOn('selectedArtist');

// artist.js

self.artist =
ko.observable().subscribeTo('selectedArtist'
, true);
```

AGENDA

1. Modularisierung mit RequireJS & knockout-postbox
2. Routing
3. Service-Anbindung
4. Hands-On #2

OJROUTER

- Import per `'ojs/ojrouter'`
- Routing zwischen verschiedenen Modulen
- Statische Router-Instanz per `oj.Router.rootInstance`
  - **Reicht für gewöhnlich aus**
  - **Ansonsten: Child-Router**

ROUTER GRUNDLAGEN

- Konfiguration per `router.configure`
  - Parameter: Config-Objekt, das mögliche Router States definiert (Key = Name der View / ViewModel)
- Navigation per `router.go`
  - Parameter: ID des Router State der aktiviert werden soll
- `router.sync` Call nötig bevor die Applikation gestartet werden kann
- Referenz auf aktuellen Router-State per ojModule Binding

ROUTER BEISPIEL

```
var router = oj.Router.rootInstance;
router.configure({
  'search': {label: 'Suche', isDefault: true},
  'artist': {label: 'Interpret'},
  'album': {label: 'Album'},
  'add-artist': {label: 'Add Artist'}
});

var viewModel = {
  router: router
};

$(document).ready(function () {
  oj.Router.sync();
  ko.applyBindings(viewModel, document.getElementById('page'));
});
```

AGENDA

1. Modularisierung mit RequireJS & knockout-postbox
2. Routing
3. Service-Anbindung
4. Hands-On #2

EIGENE AMD-MODULE VERWENDEN

- Modul in requirejs.config Aufruf bekannt machen
- Beispieleintrag:
  - `'jquery': 'libs/jquery/jquery-3.1.0'`
- Dann Referenz per ‘jquery’ in define calls

REST-CALLS IN JET

- Typischerweise mit $.ajax (also jQuery)
- Parameter:
  - Webservice-URL
  - Options (dataType, method)
- Return: Promise

PROMISES

- “Versprechen”, dass ein Wert zurückgegeben wird
  - sofort oder später
- Wert konsumieren per `then(function () {…})`
- Fehlerfall abfangen per `catch(function () {…})`

SERVICE-ANBINDUNG BEISPIEL

```
var fetchAlbumsByArtist = function fetchAlbumsByArtist (artistId) {
  return $.ajax(
    webserviceUrl + 'artists/' + artistId + '/albums', {
      dataType: 'json',
      method: 'GET'
    }
  );
};
```

AGENDA

1. Modularisierung mit RequireJS & knockout-postbox
2. Routing
3. Service-Anbindung
4. Hands-On #2

Hands On #2

MODULARISIERUNG, ROUTING SERVICE-ANBINDUNG

HANDS-ON #2

- exercises/2_modularisierung
- README öffnen (Browser)

```
$ cd exercises/2_modularisierung
```

```
$ grunt serve
```

- **Optionale Bonusaufgabe**

HABEN SIE NOCH FRAGEN?

CONSULTING

enpit