# WHY WE HATE ANGULARJS AND WHY WE STILL KEEP USING IT

A dedication of love to AngularJS

Christian Ulbrich, Zalari UG

# AGENDA PROPAGANDA

- Don’t get me wrong
- *„I hate you so much, that I can’t stop loving you“*
- The dark vs. the light side
- The path of the light
- Bar time

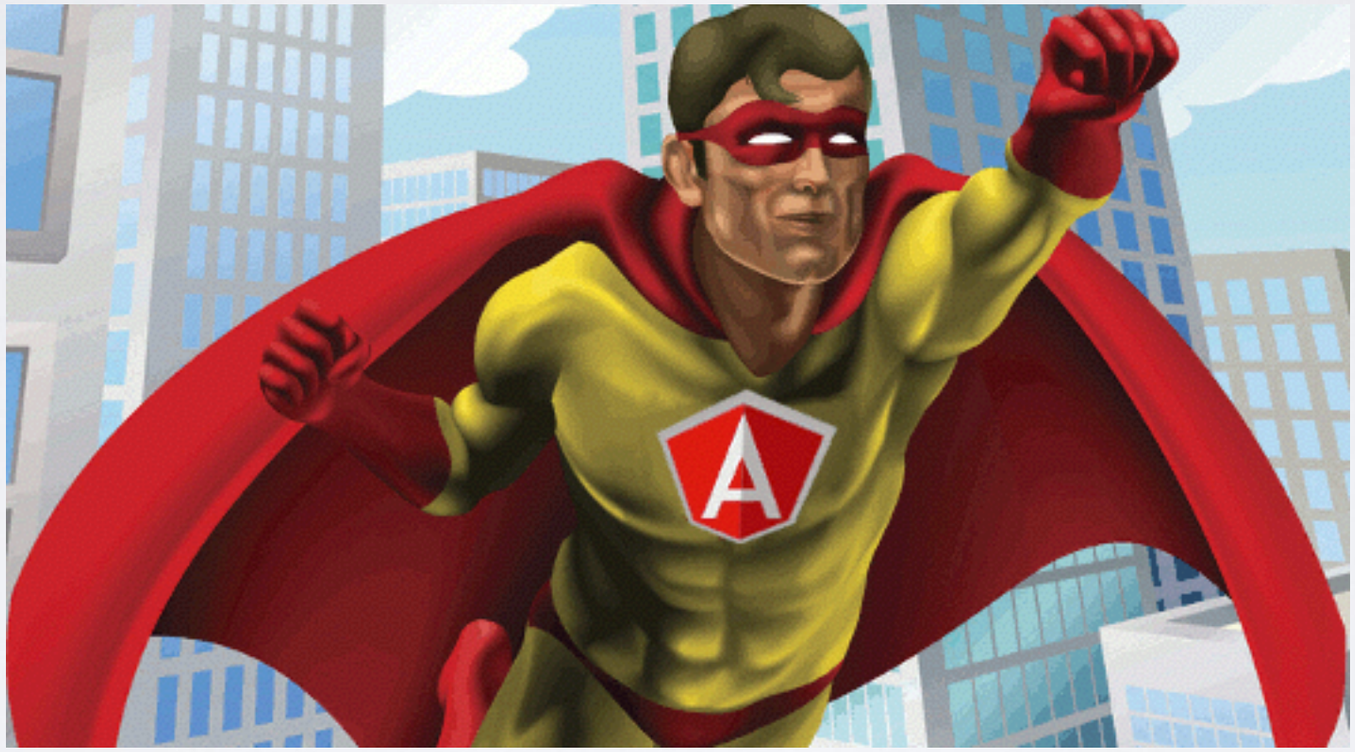
# INTRO

- 3 years of experience with AngularJS
- several projects, two huge AngularJS-centric projects for DAX companies
- I love you so hard!

# INTRO

- first version of AngularJS was released in 2009
- gained attention during the last 3 years
- Google is the main force behind, although completely open
- everyone wants to „*do Angular*“

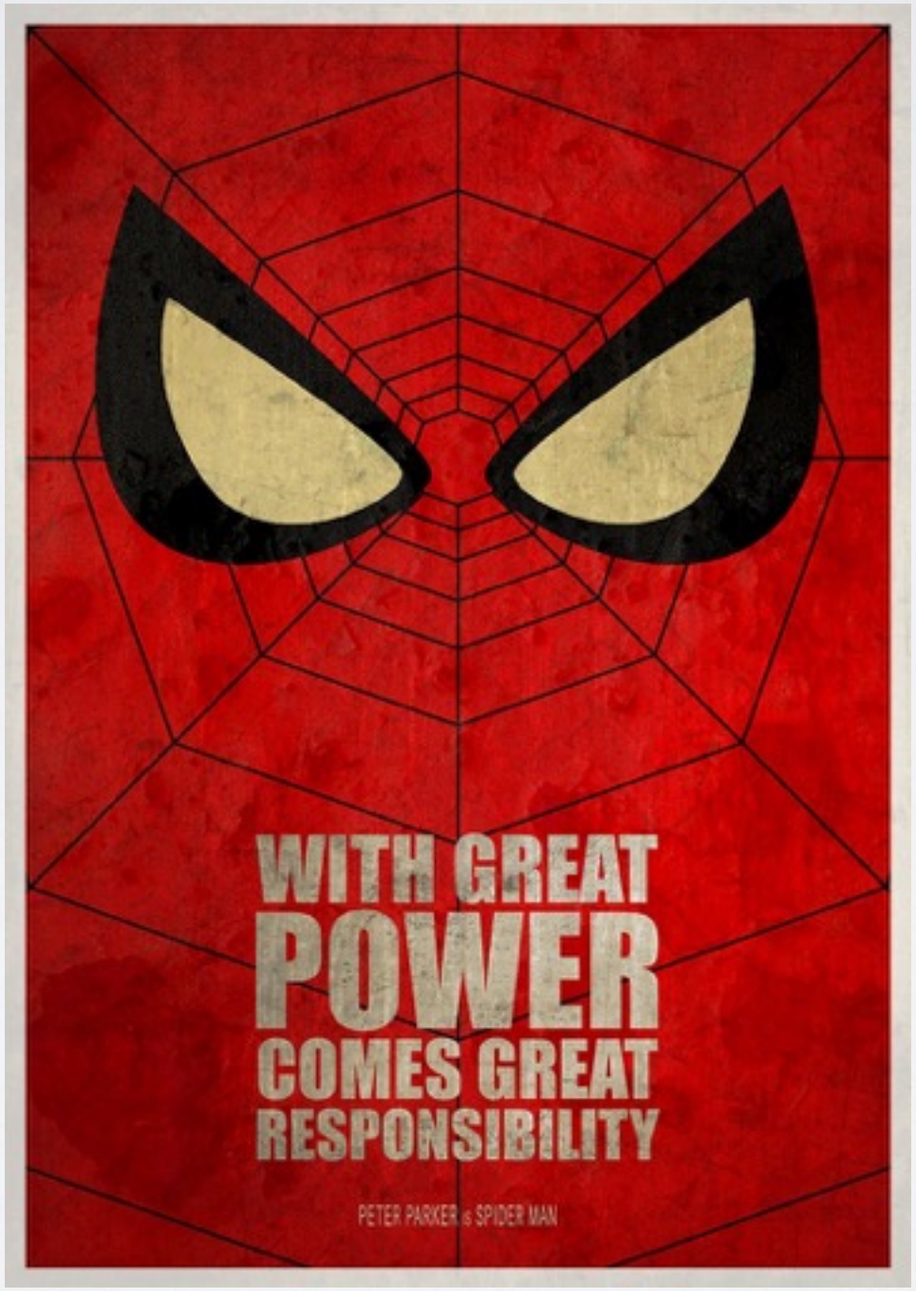
# ANGULARJS SUPERPOWERS



# ANGULARJS SUPERPOWERS

- first steps are easy
- 100+ tutorials for TODO - Apps
- Google can’t be wrong!

# ANGULARJS SUPERPOWERS



# LARIS[illegible]E DAR

- Overwhelming set of features
- Lots of micro APIs
- Prototypical Scope Inheritance
- Inconsistent documentation
- Lack of best practices, guidelines
- Moving target
- Strongly opinionated
- No async calls in config phase
- Hard to debug

# ANGULARJS THE DARK SIDE

Inconsistent documentation

# INCONSISTENT DOCUMENTATION

# INCONSISTENT DOCUMENTATION

- Guide https://docs.angularjs.org/guide/introduction
- API https://docs.angularjs.org/api
- Wiki https://github.com/angular/angular.js/wiki

# ANGULARJS THE DARK SIDE

Lack of best practices, guidelines

# LACK OF BEST PRACTICES

- best practices are scattered throughout the documentation
- no real guidelines
- best practices from wiki are a joke
- file layout from generator-angular does not scale
- ng-route is only for small apps

Notice that `{{vojta.name}}` and `{{vojta.address}}` are empty, meaning t
controller, it's not available within the directive.

As the name suggests, the **isolate scope** of the directive isolates everything ex
hash object. This is helpful when building reusable components because it prev
for the models that you explicitly pass in.

**Note:** Normally, a scope prototypically inherits from its parent. An isolated sc
scope" section for more information about isolate scopes.

**Best Practice:** Use the `scope` option to create isolate scopes when making
app.

## Creating a Directive that Manipulates the DO

In this example we will build a directive that displays the current time. Once a s

Directives that want to modify the DOM typically use the `link` option to regist
after the template has been cloned and is where directive logic will be put.

`link` takes a function with the following signature,

```
function link(scope, element, attrs, controller, transcludeFn) {
```

- `scope` is an Angular scope object.
- `element` is the jqLite-wrapped element that this directive matches.
- `attrs` is a hash object with key-value pairs of normalized attribute nam
- `controller` is the directive's required controller instance(s) or its own c
  require property.
- `transcludeFn` is a transclude linking function pre-bound to the correct

For more details on the `link` option refer to the `$compile` API page.

In our `link` function, we want to update the displayed time once a second, or
our directive binds to. We will use the `$interval` service to call a handler on a
works better with end-to-end testing, where we want to ensure that all `$timeo`
want to remove the `$interval` if the directive is deleted so we don't introduc

# ANGULARJS THE DARK

Prototypical
Scope Inheritance

# PROTOTYPICAL SCOPE INHERITANCE

- the biggest „mistake“
- prototypical inheritance is unintuitive
- using $parent in your own directive templates is going to fail
- hard to tell which directives create isolated scope and which don’t
- some scopes are implicitly created (formScopes)

# ANGULARJS THE DARK SIDE

Lots of micro APIs

# LOT'S OF MICRO APIS

- $interpolateProvider anyone?
- $http transformRequest / transformResponse / interceptors
- ngModelController $parsers, $validators, $asyncValidators
- angular.module().filter() vs. $filter(,,filter‘‘) filterPredicate functions

# ANGULARJS THE DARK SIDE

Hard to debug

Moving target

# DEBUGGING + MOVING TARGET

- debugging scope (issues) is difficult
- Batarang is not reliably working or crashes your own app
- AngularAPI is constantly evolving
  - bind-once syntax in 1.3
  - ng-message introduced in 1.3, refactored in 1.4
  - silent feature updates because of PRs, fixes
- migration documentation is not always up to date

# LARJS THE DARK SIDE

Overwhelming set of features

Strongly opinionated

# DROWNING IN FEATURES

# DROWNING IN FEATURES

- Directives
- Services, Factories, Constants, Providers, Values, Controllers
- fault-tolerant expressions

# DROWNING IN FEATURES

- link function vs compile vs pre-link vs post-link
- complex opaque digest cycle,
- multiple tiny languages (filterPredicates, ngRepeat expressions, …)

# OPINIONATED ARCHITECTURE

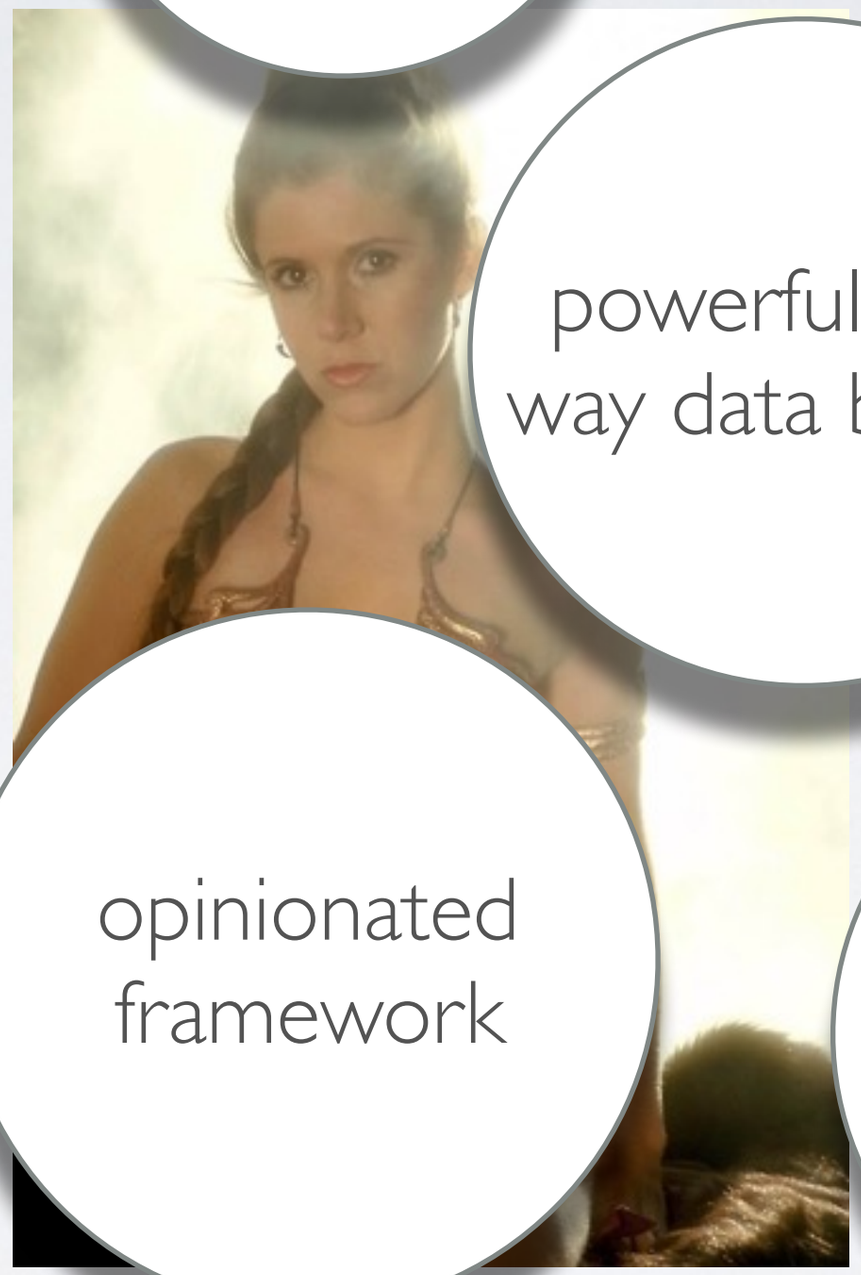
- lock-in, re-usability of AngularJS code in other frameworks is next-to impossible
- only really suitable for single page apps
- components can only be used with the AngularCore; no public API

# A LIGH

global exception handling

powerful filter concept

strict use of promises for async operations



powerful two-way data binding

decorators

working separation of concerns

opinionated framework

global $http pipeline

# ANGULARJS THE LIGHT SIDE

opinionated
framework

# OPINIONATED FRAMEWORK

- AngularJS is the de-facto solution for SPAs
- useful Dependency Injection, working modularization
- enforces modularization and discourages too tight coupling of components
- freedom of choice can be a bad thing…
- not too strongly opinionated, good choice

# ANGULARJS THE LIGHT SIDE

working separation
of concerns

# SEPARATION OF CONCERNS

- clear separation of templates and controllers
  - CSS artist + frontend dev can work together
  - via two-data binding, the template is rather easy
  - declarative directives allow for even cleaner templates

# SEPARATION OF CONCERNS

```
<div ng-message="deletionWarning">
  <otdb-show-warning-message
    name="delete-car"
    description-text="Sind Sie sicher dass Sie das Fahrzeug löschen wollen?"
    confirm-function="deleteCar()"
    confirm-label="Löschen"
    reset-function="hideWarningMessages()">
  </otdb-show-warning-message>
</div>
<div ng-message="unavailableWarning">
  <otdb-show-warning-message
    name="undriveable-car"
    only-confirm
    description-text="Das Fahrzeug muss verfügbar bleiben, da es noch in offenen Buchungen ist!"
    confirm-function="hideWarningMessages()"
  </otdb-show-warning-message>
</div>
</div>
```

# ARJS THE LIGHT SIDE

global exception handling

# GLOBAL EXCEPTION HANDLING

- AngularJS wraps the whole app in try/catch handler and allows for a global exception handler
- *every* Exception can thus be caught
- in combination with a http interceptor one can throw own exceptions
- -> all http errors can be caught no matter on which action they occur

# GLOBAL EXCEPTION HANDLING

```
//hook up $exceptionHandler with our custom exceptionHandler
$provide.decorator('$exceptionHandler', function($delegate, appExceptionHandler) {
  return function(exception, cause) {
    //for the time being, use the standard behaviour
    $delegate(exception, cause);
    //call our own implementation
    appExceptionHandler.handleException(exception, cause);
  };
});
```

# ANGULAR LIGHT SIDE

powerful
filter concept

# POWERFUL FILTER CONCEPT

- pipes and filters is a proven software architecture
- in combination with ng-repeat a lot of functionality can be realized
  - search across all field
  - faceted search, …

# POWERFUL FILTER CONCEPT

- filters can be used for translation
  - AngularJS date filter + currency are locale-aware
  - angular-translate to enable on-demand, lazy-loading, dynamic, client-side translation
  - pluralization, salutations are also use cases
- easy syntax for CSS artist

# POWERFUL FILTER CONCEPT

```
<tr ng-repeat="Customer in Customers  | filter:filterByAlphabet | filter:filterByArchiveState |
orderBy:settings.customers.list.sort.getActive():!settings.customers.list.sortOrder.get()"  ng-class="{'active':
activeId == Customer.model.id}" >
```

```
<tr ng-repeat="Booking in Bookings | bookingInTimeRange:filter.leadBegin:filter.leadEnd |
                                     bookingWaitingOrEscalated:filter.waitingOrEscalated |
                                     bookingArchive:filter.archived | bookingState:filter.bookingState |
                                     bookingType:filter.bookingType | bookingByEmployee:filter.employee.id |
                                     filter:filter.searchText |
                                     orderBy:settings.bookings.list.sort.getActive():!settings.bookings.list.sortOrder.get()"
    ng-class="{'active': activeId == Booking.model.id}">
```

# POWERFUL FILTER CONCEPT

- AngularJS' built-in *filter* can easily be customized with filterPredicate functions

```
this.personBySearchText = function (searchText) {
  searchText = searchText.toLowerCase();
  return function (person) {
    if (searchText.length >= MIN_COUNT_PERSONS) {
      return (person.name.toLowerCase().indexOf(searchText)>=0) ||
        (person.firstname.toLowerCase().indexOf(searchText)>=0);
    } else {
      return true;
    }
  }
};
```

# ANGULARJS THE LIGHT SIDE

powerful two-way data binding

# TWO-WAY DATA BINDING

- algorithm are about data, not about the representation
- thus algorithms itself are not coupled to a representation and are re-usable
- d3.js has ultimately the same concept

# TWO-WAY DATA BINDING

- powerful expressions, that *mostly* work out of the box
- since Angular 1.3+ there is the new bind-once syntax {{:model}} to allow for one-way-binding if performance is a concern

# ANGULARJS THE LIGHT SIDE

decorators

# DECORATORS

- AngularJS allows *decorating* arbitrary services in the config phase via $provide
- useful for decorating $log , mocking back-ends, function spies, …

# DECORATORS

```
//decorate $log.debug for production
$provide.decorator('$log', function($delegate, appConfig) {

  //for runtime configuration of debugging, we need to make $log.debug
  //aware of the app config

  var originalDebugFn = $delegate.debug;

  $delegate.debug = function(args) {
    //if debugging is enabled call original implementation with provided args
    if (angular.isDefined(appConfig.debug) && appConfig.debug.enabled) {
      originalDebugFn.apply(null, arguments);
    }
    //otherwise do nothing...
  };


  return $delegate;

});
```

# ANGULARJS THE LIGH

strict use of promises for async operations

# ANGULARJS THE LIGHT SIDE

- I love promises!
- callbacks are imperative, promises are functional!
- promises are going to be in ES6 anyway
- they clearly abstract asynchronous problems in a better way

# ANGULARJS THE LIGHT SIDE

- $http uses promises
- angular-ui $modals are promises
- and your code should use promises with $q as well
- using $q automagically triggers digest cycle

# THE PATH OF THE LIGHT

- Angular Style Guide: http://bit.ly/19hgCvm
- Scalable code organization in AngularJS: http://bit.ly/1nJ8Ktk
- Angular-UI Router: http://bit.ly/1idkZH7
- Angular Migration: http://bit.ly/1GYItBe
- An Unconventional Review of AngularJS: http://bit.ly/1ukuLfZ

# THE PATH OF THE LIGHT

- Callbacks are imperative, promises are functional: http://bit.ly/1hwxDEf
- When and how to use compile, controller, pre-link and post-link: http://bit.ly/1EEV1vG
- Understanding Scopes: http://bit.ly/1hCpOhg

# BAR TIME

- Discussion!
- Questions!
- Be a part of us!