> CONFESS_2015

Conference for
Enterprise Software
Solutions_

Modern Web-Frontend Development 101: Angular, Less, Bower and more ...

Stefan Schuster

LESS

Less

- CSS pre-processor
  - Has nothing to do with JS
  - Introduces additional concepts to CSS
    - Variables
    - Mixins
    - Nested declarations
    - Functions

Less

```
.button {
    .font(@size: 20px);
    background: red;
    .border-radius(5px);

    > span {
        color: black;

        > a {
            text-decoration: underline;

            &:hover {
                font-weight: bold;
            }
        }
    }

    &.disabled {
        background: gray;
    }
}
```

Node.js

Node.js

- JavaScript runtime environment
  - Based on Google (Chrome) V8 Engine
  - No browser APIs
    - No DOM
    - No UI
    - No Window, ...
  - Everything form the JS base language works

Node.js

- Alternate APIs available
  - Buffer (Binary Data)
  - Crypto (SHA, MD5, ...)
  - File System
  - Network
    - HTTP/HTTPS
    - UDP
    - DNS
  - HTTP Server & Client libraries
  - OS/Process (for command line utilities, ...)

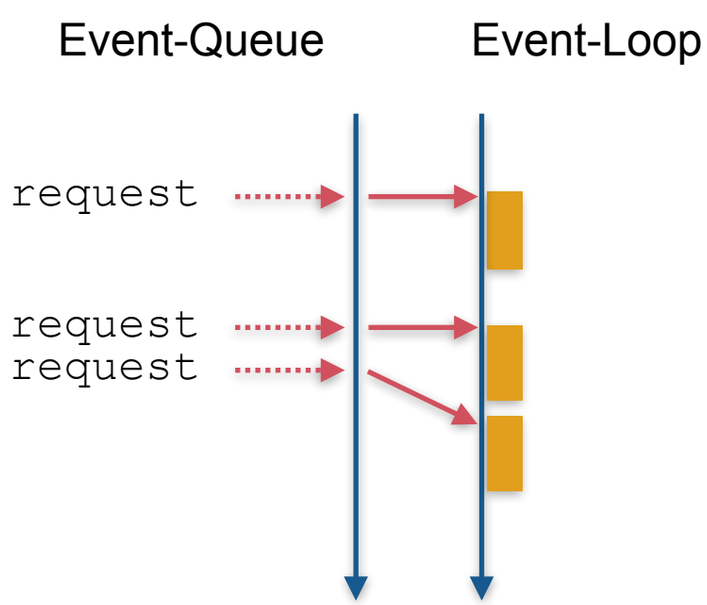
Node.js

- Used for a lot of development tools
- But also for real server side programming
  - Best known "Hello World" example:

```
var http = require("http");

http.createServer(function(req, res) {
   res.end("Hello, World!");
}).listen(8080);

console.log("Server running at http://localhost:8080/");
```

Node.js

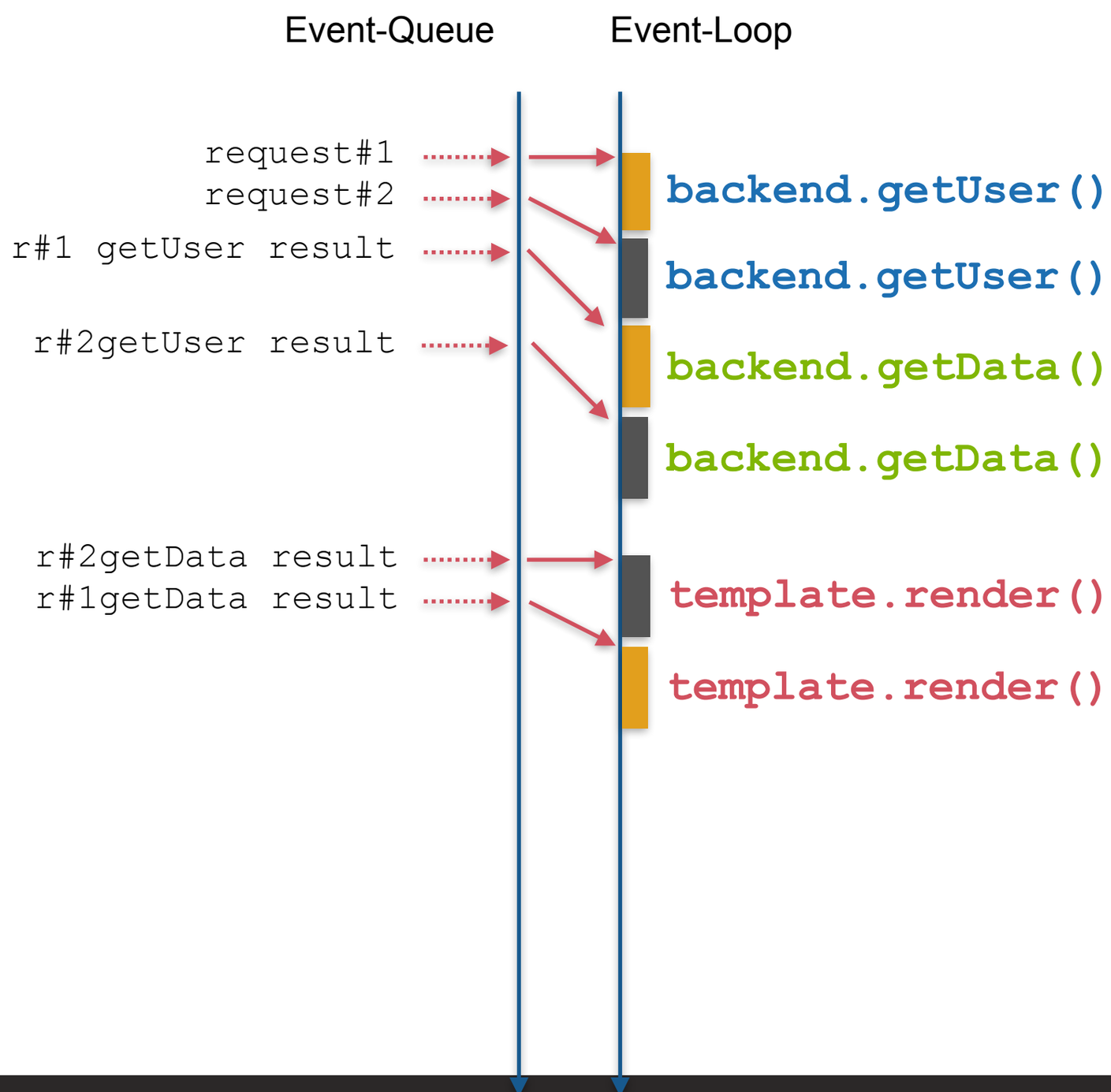
- Like all JavaScript: Async
  - "Reactor" pattern
  - Surprisingly performant



Node.js

```
var http = require("http");
var backend = require("./backend");
var template = require("./template");

http.createServer(function(req, res) {
    backend.getUser(req.headers, function(err, user) {
        //Todo: Error checking
        backend.getData(user, function(err, data) {
            //Todo: Error checking
            var html = template.render({ user:user, data:data });
            res.end(html);
        });
    });
}).listen(8080);
```

Node.js


Event-Queue
Event-Loop
request#1
request#2
r#1 getUser result
r#2getUser result
r#2getData result
r#1getData result
backend.getUser()
backend.getUser()
backend.getData()
backend.getData()
template.render()
template.render()

NPM

- Node Package Management System
  - Comparable to Maven
  - Part of Node.js
  - > 60.000 modules
  - Good resolution of dependencies

```
test — bash — 113×43
Marble:test sschuster$ npm install coffee-script
npm http GET https://registry.npmjs.org/coffee-script
npm http 200 https://registry.npmjs.org/coffee-script
npm http GET https://registry.npmjs.org/coffee-script/-/coffee-script-1.7.1.tgz
npm http 200 https://registry.npmjs.org/coffee-script/-/coffee-script-1.7.1.tgz
npm http GET https://registry.npmjs.org/mkdirp
npm http 200 https://registry.npmjs.org/mkdirp
coffee-script@1.7.1 node_modules/coffee-script
└── mkdirp@0.3.5
Marble:test sschuster$
```

package.json

- Config file for dependencies
  - package.json

```
{
    "name": "my-project",
    "version": "1.0.0",
    "private": true,
    "dependencies": {
        "coffee-script": "1.9.1",
        "grunt-less": "0.1.7",
        "jasmine": "2.2.1"
    }
}
```

Gulp

Grunt / Gulp

- JavaScript Task Runners
  - Like Gradle (and a little mit like MVN)
  - The bigger the client side project, the more of these JS ecosystem "technologies" get used
  - They often require pre-processing steps
    - LESS compilation
    - JS Transpile (CoffeeScript, TypeScript)
    - JS Build (Concatenation, Minification)
    - JS TestRunner

Grunt / Gulp

```
npm install
npm install -g grunt-cli

grunt less
```

Grunt / Gulp

- Task Runners take over this tasks within such development workflows
  - Example: Gruntfile.js (Grunt Config)

```
module.exports = function(grunt) {
  grunt.initConfig({
    less: {
      styles: {
        options: {
          compress: true
        },
        files: {
          "app/lesscss/main.css": "app/less/main.less"
        }
      }
    }
  });

  grunt.loadNpmTasks("grunt-contrib-less");
};
```

Bower

Bower

- How to deal with client-side dependencies?
  - Create ReadMe with download instructions?
  - Check into VCS?

- Bower
  - Dependency management for client side libraries

```
npm install -g bower
```

Bower

- bower.json

```
{
  "name": "my-project",
  "version": "1.0.0",
  "dependencies": {
    "jquery": "2.1.3",
    "angular": "1.3.14",
    "angular-cookies": "1.3.14",
    "angular-ui-router": "0.2.13"
  }
}
```

- Plus: .bowerrc
  - Config in which directory dependencies should be put into

Angular

AngularJS

- AngularJS (by Google)
  - Very modern and popular framework
  - Radically different concepts than e.g. Dojo
  - Not a "full-stack" framework
    - No out of the box widgets

- Core concept based on "enhanced" HTML and two way binding

AngularJS

- Example HTML:

```
<div ng-controller="UserListController as userList">
  <h1>User List</h1>
  <div ng-show="userList.users.length == 0">No users yet</div>
  <ul ng-show="userList.users.length > 0">
    <li ng-repeat="user in userList.users">
      {{user.name}}
    </li>
  </ul>
  <input type="text" ng-model="userList.newUserName">
  <button ng-click="userList.add()">Add User</button>
</div>
```

AngularJS

- Example JS:

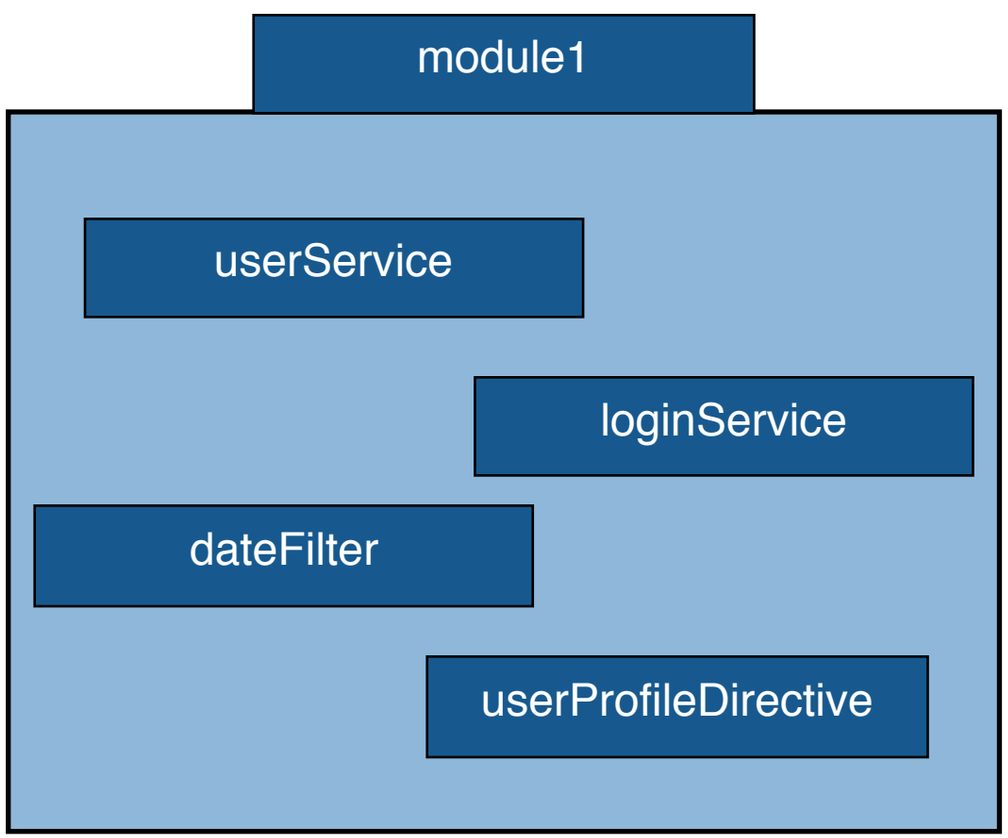
```
angular.module("app", [])
  .controller("UserListController", function() {
    this.users = [];
    this.newUserName = "";

    this.add = function() {
      this.users.push({
        name: this.newUserName
      });
    }
  });
```

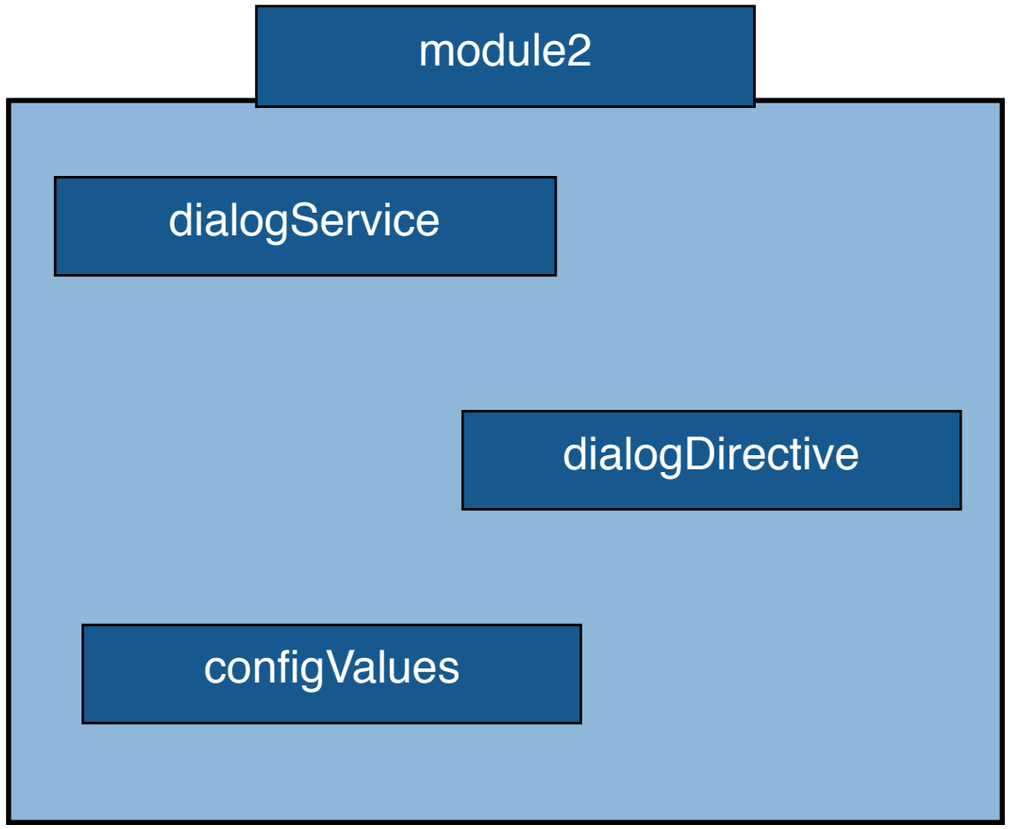
AngularJS

- So basically you have
  - Sections of HTML with some bound functionality and two way binding
- But Angular provides more tools to structure big projects
  - Based on "Dependency Injection"
  - Directives
  - Factories
  - Services

DI in AngularJS

- Angular apps have two "layers" of code organisation, both with their own DI
  - modules
  - services / specialized objects





DI in AngularJS

- Modules can depend on other modules
- "objects" inside modules can depend on other "objects" in same or other module

- Angular takes care of resolving and ordering all this dependencies

DI in AngularJS

- Example

```
angular.module("app", ["backendModule"])
  .controller("UserListController", ["backendService",
function(backendService) {
    var that = this;
    this.users = [];
    this.newUserName = "";

    this.add = function() {
      backendService.createUser(this.newUserName).then(function() {
        that.users.push({
          name: this.newUserName
        });
      });
    }
  }]);
```

Directives

- Directives allow to create own tags and attributes
  - Encapsulate functionality

Element

```
<div ng-controller="someController">
  <span>Some UI</span>
  <user-profile user="user"></user-profile>
</div>
```

Attribute

```
<div tooltip-help>
  Some UI code
</div>
```

Directives

- Element Directive HTML

```
<div class="userProfile">
  <strong>Name:</strong> {{userProfile.user.name}}
  <button ng-click="userProfile.sendMessage()">Send message</button>
</div>
```

Directives

- Element Directive JS

```
angular.module("app", [])
  .directive("userProfile", function() {
    return {
      restrict: "E",
      scope: {
        user: "="
      },
      templateUrl: "./userProfile.html",
      controllerAs: "userProfile",
      bindToController: true,
      controller: function() {
        this.sendMessage = function() {
          //Do something...
        }
      }
    };
  });
```

Services

- Different providers available
  - Share functionality and communication within the app
  - All singletons

```
angular.module("app", [])
  .service("userService", function() {
    this.loggedInUser = null;

    this.loginUser = function() {
      //Do some Ajax Call or something
    }
    this.logoutUser = function() {
      //Do something ...
      this.loggedInUser = null;
    }
  });
```

Require.js

RequireJS

- Most widespread implementation of AMD
  - Mentioned before > Dojo
  - Stand alone implementation
  - AMD is generally pretty common nowadays

```
define([
  "dependency1"
  "dependency2"
], function(resolvedDependency1, resolvedDependency2) {
  return "someModule";
});

require([
  "dependency1"
  "dependency2"
], function(resolvedDependency1, resolvedDependency2) {
  //Run some code
});
```

RequireJS

- Many libraries already support AMD
  - jQuery
- Can basically be used with everything
  - jQuery
  - Angular
  - Any JS code

- Doesn't pollute global namespace
- Better separation of code

RequireJS

- Plugins available
  - Load text (like dojo/text)
  - Live transpile of alternative languages (coffeescript)

- Build available
  - r.js
  - Like dojo build > concatenate resources into one production file

irian
THE SOFTWARE EXPERTS

> CONFESS_2015

Conference for
Enterprise Software
Solutions

con-fess.com

@con_fess

irian.at