Advanced Developer Conference 2013

AngularJS by Google

Silverlight-Style HTML Apps

Rainer Stropek

software architects gmbh

Web http://www.timecockpit.com
Mail rainer@timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

Read/Download Sourcecode of Samples at
http://bit.ly/AngularTypeScript

Agenda

Introduction

What's it all about?

Image Source:
http://flic.kr/p/9bUJEX

Learn

Angular by example

Image Source:
http://flic.kr/p/3budHy

How far?

What didn't we cover?
How far can it go?

Image Source:
http://flic.kr/p/765iZj

Stop or go?

Critical evaluation

Image Source:
http://flic.kr/p/973C1u

TypeScript

- This presentation uses AngularJS with TypeScript
  - JavaScript is generated from TypeScript
  - However, you still have to understand the concepts of JavaScript
- TypeScript advantages
  - Type-safe AngularJS API (at least most part of it)
  - Native classes, interfaces, etc. instead of JavaScript patterns and conventions
  - Possibility to have strongly typed models
  - Possibility to have strongly typed REST services
- TypeScript disadvantages
  - Only a few AngularJS + TypeScript samples available
  - Additional compilation step necessary

Introduction

What's it all about?

Image Source:
http://flic.kr/p/9bUJEX

What's AngularJS

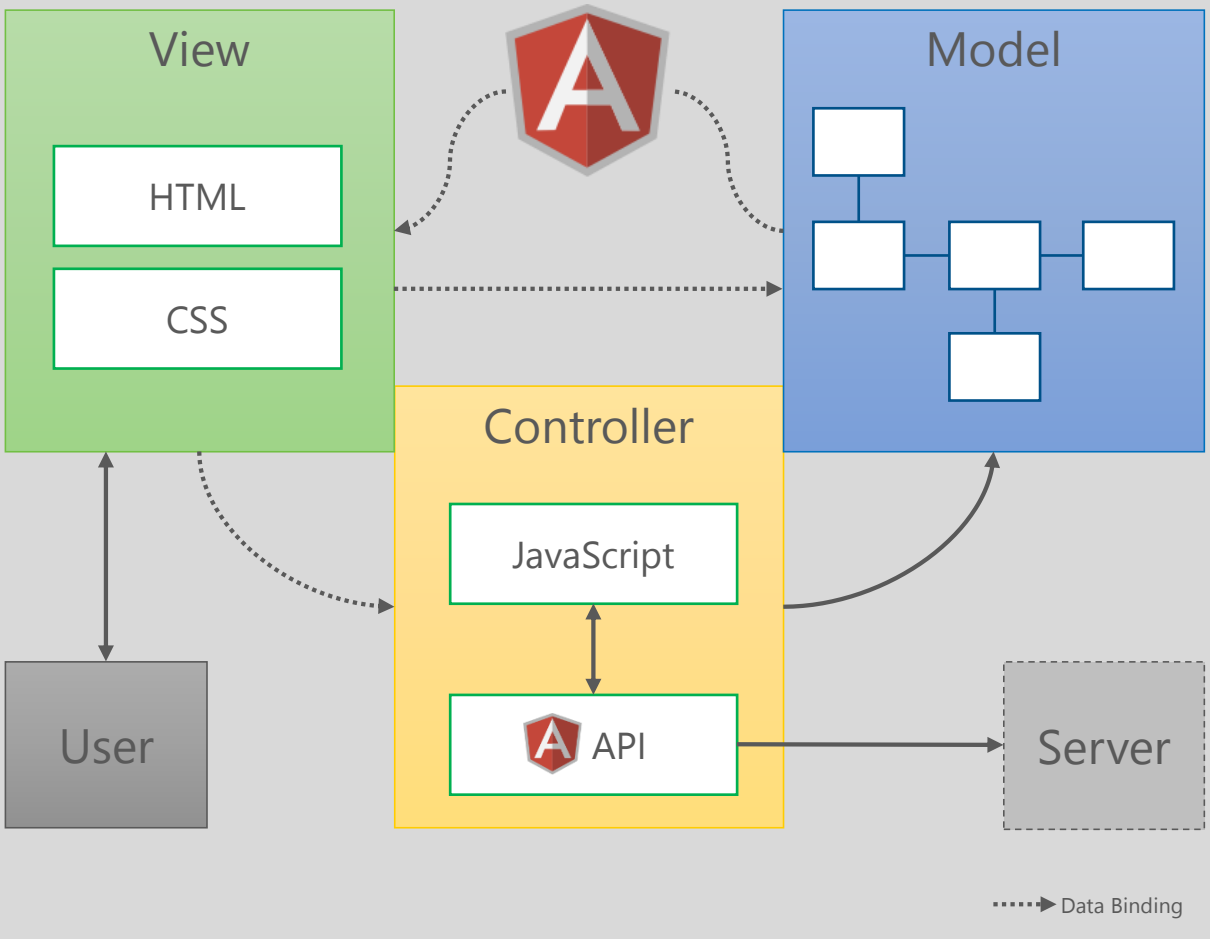
Developer's Perspective

- **MVC + data binding framework**
  - Fully based on HTML, JavaScript, and CSS → Plugin-free
  - Enables automatic unit testing
- **Dependency injection system**
  - Module concept with dependency management
- **Handles communication with server**
  - XHR, REST, and JSONP
  - Promise API for asynchronous programming

What's AngularJS

Developer's Perspective

- Navigation solution for SPAs
  Single Page Applications
- HTML extensibility mechanism
  Custom directives



MVC

Architectural Pattern

Layers

View: Visual appearance (declarative languages)

Model: Data model of the app (JavaScript objects)

Controller: Adds behavior (imperative languages)

Workflow

User interacts with the view

Changes the model, calls controller (data binding)

Controller manipulates model, interacts with server

AngularJS detects model changes and updates the view (two-way data binding)

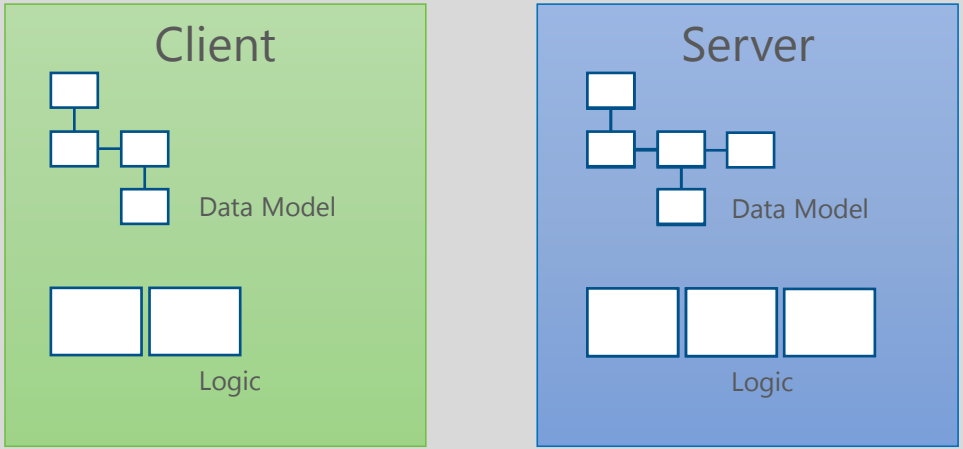
MVC Notes

- MVW = Model View Whatever
  MVC is not a precise pattern but an architectural pattern
- Clear separation of logic, view, and data model
  Data binding connects the layers
- Enables automated unit tests
  Test business logic and UI behavior (also kind of *logic*) without automated UI tests

Important Differences

- HTML+CSS for view
  Plugin-free
  Extensibility introduced by AngularJS
- Data binding introduced by AngularJS
  Change detection using model comparison
- JavaScript
- Many different development environments
  Open Source

- XAML for view
  Silverlight browser plugin
  Extensibility built in (e.g. user controls)
- Data binding built into XAML and .NET
  INotifyPropertyChanged, Dependency Properties
- CLR-based languages (e.g. C#)
- First-class support in Visual Studio
  Provided by Microsoft



Shared Code

JavaScript/TypeScript Everywhere

- Shared code between client and server
  - Server: nodejs
  - Single source for logic and data model
- Mix with other server-side platforms possible
  - E.g. ASP.NET

```
angular.module('helloWorldApp', [])
  .config(function ($routeProvider) {
    $routeProvider
      .when('/', {
        templateUrl: 'views/main.html',
        controller: 'MainCtrl'
      })
      .when('/about', {
        templateUrl: 'views/about.html',
        controller: 'AboutCtrl'
      })
      .otherwise({
        redirectTo: '/'
      });
  });
```

```
angular.module('helloWorldApp')
  .controller('MainCtrl', function ($scope) {
    …
  });
```

```
<div class="container"
  ng-view="">
</div>
```

```
<div class="hero-unit">
  <h1>'Allo, 'Allo!</h1>
  …
</div>
```

SPA

Single Page Apps

Define routes with *$routeProvider* service

- Placeholder with „:“ (e.g. */admin/users/:userid*)
- Access route paramter values with *$routeParams* service

Define where view should be included in index.html using *ng-view*

URL Modes

- Hashbang and HTML5 mode
- See *$location* service docs for details

Tools

- **Microsoft Visual Studio**
  - Not free
  - Only Windows
  - Very good support for TypeScript
  - Integrated debugging with IE
  - Build with MSBUILD
  - Package management with NuGet
- **JetBrains WebStorm**
  - Not free
  - Windows, Mac, Linux
  - Specialized on web apps
  - Good integration of external tools
- **Your favorite editor**
  - Some free, some not free
  - E.g. Sublime, Notepad++, Vim, etc.
  - Build and test with external tools

Open Source Tools

- **Project setup**
  - Yoeman
  - angular-seed
  - Bower for web package management
- **Build**
  - Grunt for automated build
  - Karma test runner
  - Jasmine for BDD unit tests
  - JSLint, JSHint for code quality
- **UI**
  - Bootstrap CSS for prettifying UI
  - AngularUI for UI utilities and controls
  - Batarang for analyzing data bindings and scopes
- **Server-side**
  - nodejs for server-side JavaScript with various npm modules (e.g. express)

Setup demo project

```
cd yeoman-demo
yo angular hello-world
```

Build and test your app (don't forget to set CHROME_BIN*)*

```
grunt
```

Add one item to awesomeThings *in* main.js
Run automated unit tests → will fail

```
grunt test
```

Correct unit test to expect 4 instead of 3 items
Run automated unit tests → will work

```
grunt test
```

Start development loop

```
grunt server
```

Change main.js*, save it, and watch the browser refreshing*

Add a new view + controller + route, look at changes in app.js

```
yo angular:route about
```

Start development loop, launch new route (maybe with Fiddler)

```
http://localhost:9000/#/about
```

Demo

Yeoman Angular Generator

Setup angular application

- Initial setup
- Add new artifacts (e.g. route)

Run unit tests

- Karma and Jasmine

Code editing with editor

- Sublime text

Learn

Angular by example

Image Source:
http://flic.kr/p/3budHy

Demo

Project Setup

In Visual Studio

Create HTML app with TypeScript

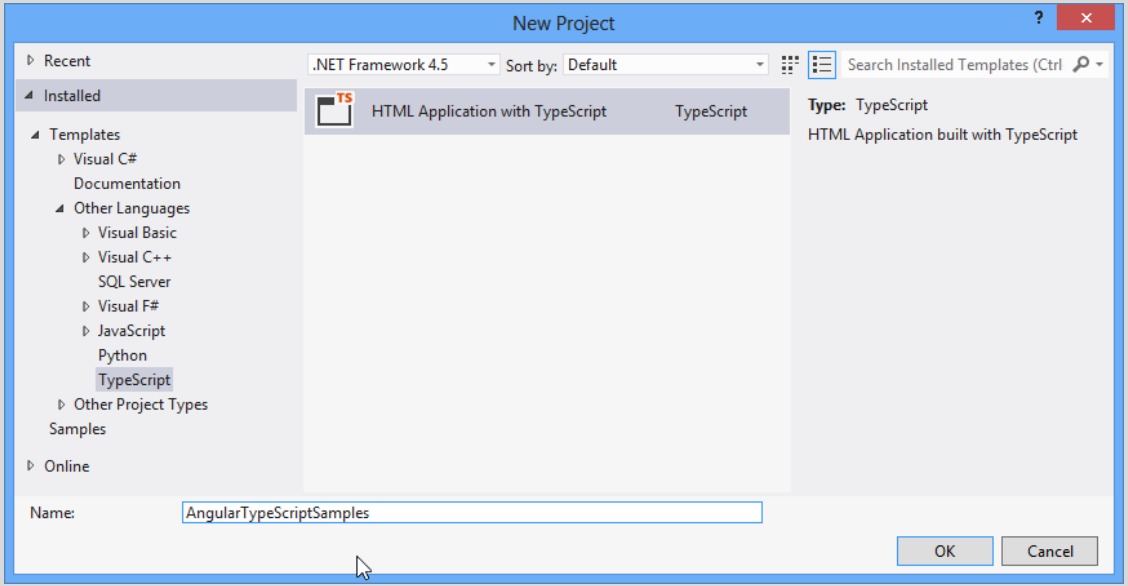
Use NuGet to add angular and bootstrap

Get TypeScript declaration from GitHub

Basic controller with two-way data binding

TypeScript

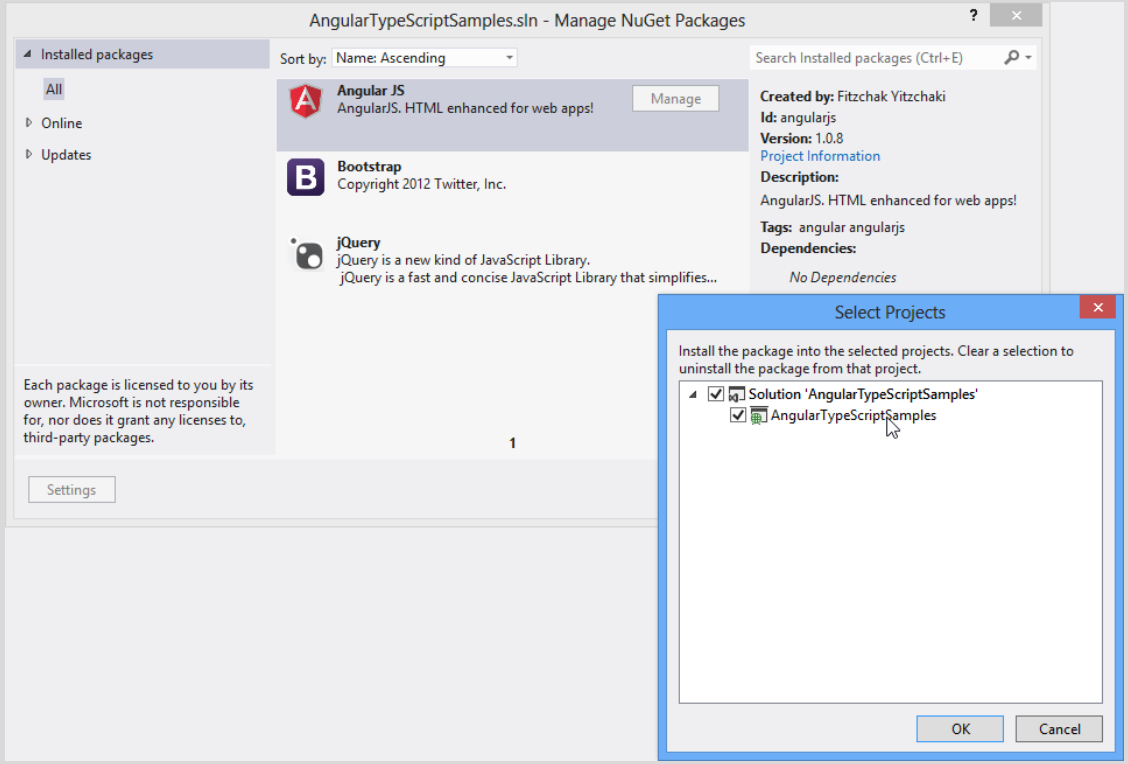
Setup TypeScript Project



Screenshot: Microsoft Visual Studio 2012, Oct. 2013

NuGet

Add JavaScript Libraries to VS Projects



Screenshots: Microsoft Visual Studio 2012, Oct. 2013

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Angular.js Samples Using TypeScript</title>
    <link href="../../../Content/bootstrap/bootstrap.css" rel="stylesheet">
    <link href="helloWorldWithController.css" rel="stylesheet">
    <script src="../../../Scripts/angular.js"></script>
    <script src="helloWorldWithController.js"></script>
</head>
<body ng-app>
    <div ng-controller="HelloCtrl">
        <form>
            <h2>Two-Way Binding</h2>
            <label for="messageInput">Say 'Hello' to:</label>
            <input type="text" id="messageInput" ng-model="name">

            <h2>Simple Bindings</h2>
            <table class="table table-hover table-condensed">
                <tr>
                    <th>Syntax</th><th>Result</th>
                </tr>
                <tr>
                    <td>Interpolation</td><td>Hello, {{name}}!</td>
                </tr>
                <tr>
                    <td>ng-bind</td><td>Hello, <span ng-bind="name" />!</td>
                </tr>
                <tr>
                    <td>Interpolation with controller function</td>
                    <td>Hello, {{getName()}}!</td>
                </tr>
                <tr>
                    <td>ng-bind with getEnclosedName</td>
                    <td>Hello, <span ng-bind="getEnclosedName('b')" />!</td>
                </tr>
                <tr>
                    <td>ng-bind-html-unsafe with getEnclosedName</td>
                    <td>Hello, <span ng-bind-html-unsafe="getEnclosedName('b')" />!</td>
                </tr>
            </table>
        </form>
    </div>
</body>
</html>
```

Controller

Basic Sample with Controller

See AngularJS docs for *ng* module

```
/// <reference
       path="../../../tsDeclarations/angularjs/angular.d.ts"/>

// Create a custom scope based on angular's scope and define
// type-safe members which we will add in the controller function.
interface IHelloWorldScope extends ng.IScope {
    name: string;
    getName: () => string;
    getEnclosedName: (tag: string) => string;
}
```

Referred to from ng-controller

```
var HelloCtrl = function ($scope: IHelloWorldScope) {
    $scope.name = "World";
    $scope.getName = () => $scope.name;
    $scope.getEnclosedName = (tag) => "<" + tag + ">"
       + $scope.name
       + "<" + tag + "/>";
};
```

Controller

Basic Sample with Controller

Get TypeScript definitions for AngularJS, Jasmine, etc. from Definitely Typed project

Collections

Binding to Collections

Create collection in controller

Binding the view to collections

Demo

```
<!DOCTYPE html>
<html lang="en">
<head>
  …
</head>
<body ng-app>
  <div ng-controller="HelloCtrl">

    <form>
      …
      <h2>Collection Binding</h2>
      <table class="table table-hover table-condensed">
        <tr>
          <th>Pos.</th>
          <th>ISO Code</th>
          <th>Country Name</th>
        </tr>
        <tr ng-repeat="country in countries">
          <td>{{$index}}</td>
          <td>{{country.isoCode}}</td>
          <td>{{country.name}}</td>
        </tr>
      </table>
    </form>

  </div>
</body>
</html>
```

Controller

Basic Sample with Controller

See AngularJS docs for *ngRepeat*

```
/// <reference
       path="../../../tsDeclarations/angularjs/angular.d.ts"/>

// Create a custom scope based on angular's scope and define
// type-safe members which we will add in the controller function.
interface IHelloWorldScope extends ng.IScope {
    name: string;
    countries: ICountryInfo[];
    getName: () => string;
    getEnclosedName: (tag: string) => string;
}

interface ICountryInfo {
    isoCode: string;
    name: string;
}

var HelloCtrl = function ($scope: IHelloWorldScope) {
    …
    $scope.countries = [
        { isoCode: 'AT', name: 'Austria' },
        { isoCode: 'DE', name: 'Germany' },
        { isoCode: 'CH', name: 'Switzerland' }];
};
```

Controller

Basic Sample with Controller

Demo

Scopes

Hierarchy of Scopes

Sample with hierarchy of scopes

Analyze scope hierarchy with Batarang

Sample inspired by Kozlowski, Pawel; Darwin, Peter Bacon: Mastering Web Application Development with AngularJS, Chapter *Hierarchy of scopes*

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Angular.js Samples Using TypeScript</title>
   <link href="../../../Content/bootstrap/bootstrap.css" rel="stylesheet">
   <script src="../../../Scripts/angular.js"></script>
   <script src="hierarchyOfScopes.js"></script>
</head>

<body ng-app>
   <div ng-controller="WorldCtrl" class="container">
      <hr>
      <ul>
         <li ng-repeat="country in countries">
            {{country.name}} has population
               of {{country.population | number:1}} millions,
            {{worldsPercentage(country.population) | number:1}} %
               of the World's population
         </li>
      </ul>
      <hr>
      World's population: {{population | number:1}} millions
   </div>
</body>
</html>
```

Controller

Basic Sample with Controller

See AngularJS docs about scopes

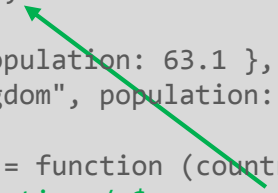
See AngularJS docs about filters

```
/// <reference
       path="../../../tsDeclarations/angularjs/angular.d.ts"/>

interface ICountry {
    name: string;
    population: number;
}

interface IHierarchyScope extends ng.IScope {
    population: number;
    countries: ICountry[];
    worldsPercentage: (countryPopulation: number) => number;
}

var WorldCtrl = function ($scope: IHierarchyScope) {
    $scope.population = 7000;
    $scope.countries = [
        { name: "France", population: 63.1 },
        { name: "United Kingdom", population: 61.8 }
    ];
    $scope.worldsPercentage = function (countryPopulation) {
        return (countryPopulation / $scope.population) * 100;
    };
};
```

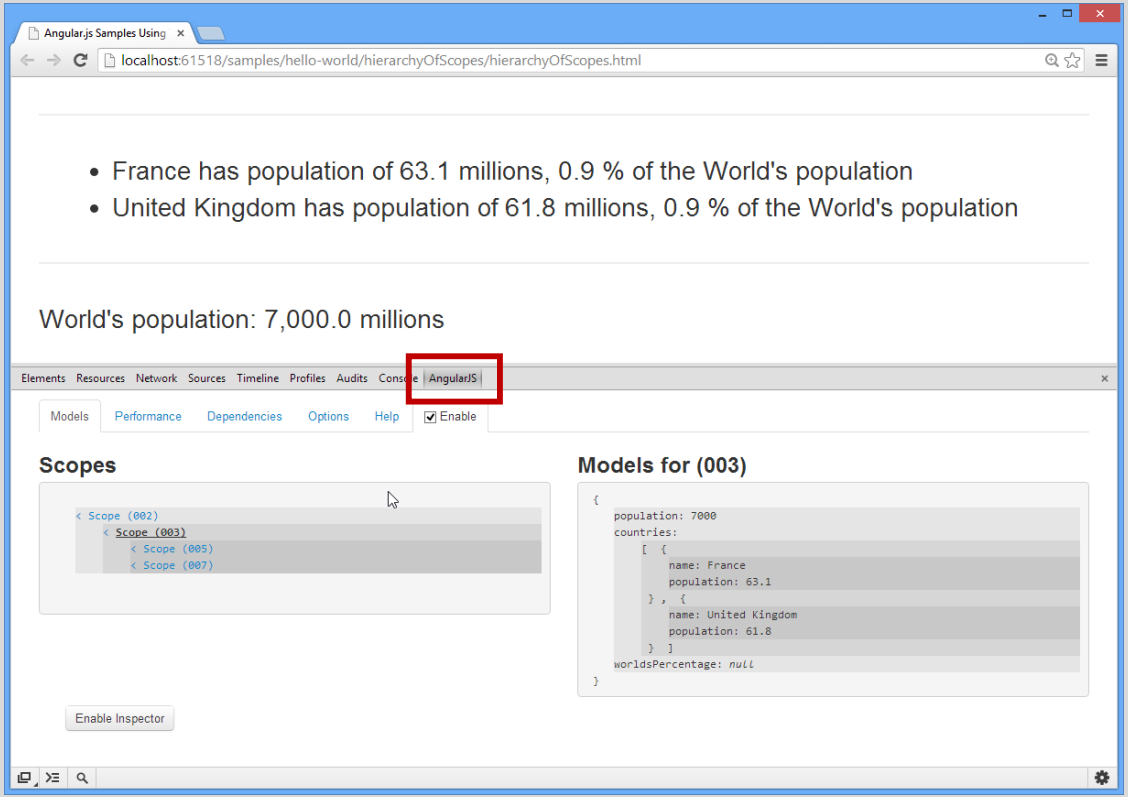


Controller

Basic Sample with Controller

Batarang

Chrome Addin



```
…
<body ng-app="notificationsApp" ng-controller="NotificationsCtrl">
  …
</body>

module NotificationsModule { …
  export class NotificationsCtrl {
    constructor(
      private $scope: INotificationsCtrlScope,
      private notificationService: NotificationsService) { … }
    …
  }

  export class NotificationsService {
    …
    public static Factory(
      …,
      MAX_LEN: number, greeting: string) { … }
  }
}

angular.module("notificationsApp", …)
  .constant("MAX_LEN", 10)
  .value("greeting", "Hello World!")
  .controller("NotificationsCtrl",
      NotificationsModule.NotificationsCtrl)
  .factory("notificationService",
      NotificationsModule.NotificationsService.Factory);
```

Modules, Services

Dependency Injection

AngularJS module system

Typically one module per application or reusable, shared component

Predefined services

E.g. $rootElement, $location, $compile, …

Dependency Injection

Based on parameter names

Tip: Use *$inject* instead of param names to be minification-safe

Demo

Modules, Services

Dependency Injection

TypeScript modules vs. AngularJS modules

AngularJS modules and factories

Sample inspired by Kozlowski, Pawel; Darwin, Peter Bacon: Mastering Web Application Development with AngularJS, *Collaborating Objects*

```
module NotificationsModule {
    export interface INotificationsArchive {
        archive(notification: string);
        getArchived(): string[];
    }
}
```

Notification Service

Contract

Contract for notifications archive

- Common for all notifications archive implementations

```
/// <reference path="INotificationsArchive.ts"/>

module NotificationsModule {
    export class NotificationsArchive
      implements INotificationsArchive {
        private archivedNotifications: string[];

        constructor() {
            this.archivedNotifications = [];
        }

        archive(notification: string) {
            this.archivedNotifications.push(notification);
        }
        public getArchived(): string[]{
            return this.archivedNotifications;
        }
    }
}
```

Notification Service

Archive Implementation

Factory function for service creation

Other options

value, *service*, *provider*

See Angular docs about *angular.Module* for details

```
/// <reference path="INotificationsArchive.ts"/>
module NotificationsModule {
    export class NotificationsService {
        private notifications: string[];
        public maxLen: number = 10;

        public static Factory(notificationsArchive: INotificationsArchive,
          MAX_LEN: number, greeting: string) {
            return new NotificationsService(
                notificationsArchive, MAX_LEN, greeting);
        }

        constructor(private notificationsArchive: INotificationsArchive,
          MAX_LEN: number, greeting: string) {
            this.notifications = [];
            this.maxLen = MAX_LEN;
        }

        public push(notification: string): void {
            var notificationToArchive: string;
            var newLen = this.notifications.unshift(notification);
            if (newLen > this.maxLen) {
                notificationToArchive = this.notifications.pop();
                this.notificationsArchive.archive(notificationToArchive);
            }
        }

        public getCurrent(): string[] {
            return this.notifications;
        }
    }
}
```

Notification Service

Service Implementation

```
/// <reference path="../../../tsDeclarations/angularjs/angular.d.ts"/>
/// <reference path="NotificationsArchive.ts"/>

module NotificationsModule {
    export interface INotificationsCtrlScope extends ng.IScope {
        notification: string;
        vm: NotificationsCtrl;
    }

    export class NotificationsCtrl {
        constructor(private $scope: INotificationsCtrlScope,
          private notificationService: NotificationsService) {
            $scope.vm = this;
        }

        private addNotification(): void {
            this.notificationService.push(this.$scope.notification);
            this.$scope.notification = "";
        }

        private getNotifications(): string[] {
            return this.notificationService.getCurrent();
        }
    }
}
```

Notification Service

Controller

```
/// <reference
  path="../../../tsDeclarations/angularjs/angular.d.ts"/>
/// <reference path="NotificationsArchive.ts"/>
/// <reference path="NotificationsService.ts"/>
/// <reference path="NotificationsCtrl.ts"/>

angular.module("notificationsApp", ["notificationsArchive"])
    .value("greeting", "Hello World")
    .constant("MAX_LEN", 10)
    .controller(
      "NotificationsCtrl",
      NotificationsModule.NotificationsCtrl)
    .factory(
      "notificationService",
      NotificationsModule.NotificationsService.Factory);
```

Notification Service

Dependency Injection

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Angular.js Samples Using TypeScript</title>
   <link href="../../../Content/bootstrap/bootstrap.css" rel="stylesheet">
   <script src="../../../Scripts/angular.js"></script>
   <script src="NotificationsArchive.js"></script>
   <script src="NotificationsService.js"></script>
   <script src="NotificationsCtrl.js"></script>
</head>

<body ng-app="notificationsApp" ng-controller="NotificationsCtrl">
<div style="margin: 10px">
    <form role="form">
        <textarea ng-model="notification" cols="40"
          rows="3" class="span6"></textarea><br>
        <button class="btn btn-primary"
          ng-click="vm.addNotification()">Add</button>
    </form>
</div>
<table class="table table-striped table-bordered">
    <tr>
        <th>Notifications</th>
    </tr>
    <tr ng-repeat="notification in vm.getNotifications()">
        <td>{{notification}}</td>
    </tr>
</table>
</body>
</html>
```

Notification Service

View

Server Communication

- *$http* service (*ng.IHttpService*)
  Support for XHR and JSONP
- *$resource* service for very simple REST services
  Not covered in this talk; see AngularJS docs for details
- *$q* service for lightweight promise API
  Note: *$http* methods return *IHttpPromise<T>*
- *$httpBackend* service (*ng.IHttpBackendService*)
  Used for unit testing of $http calls

Demo

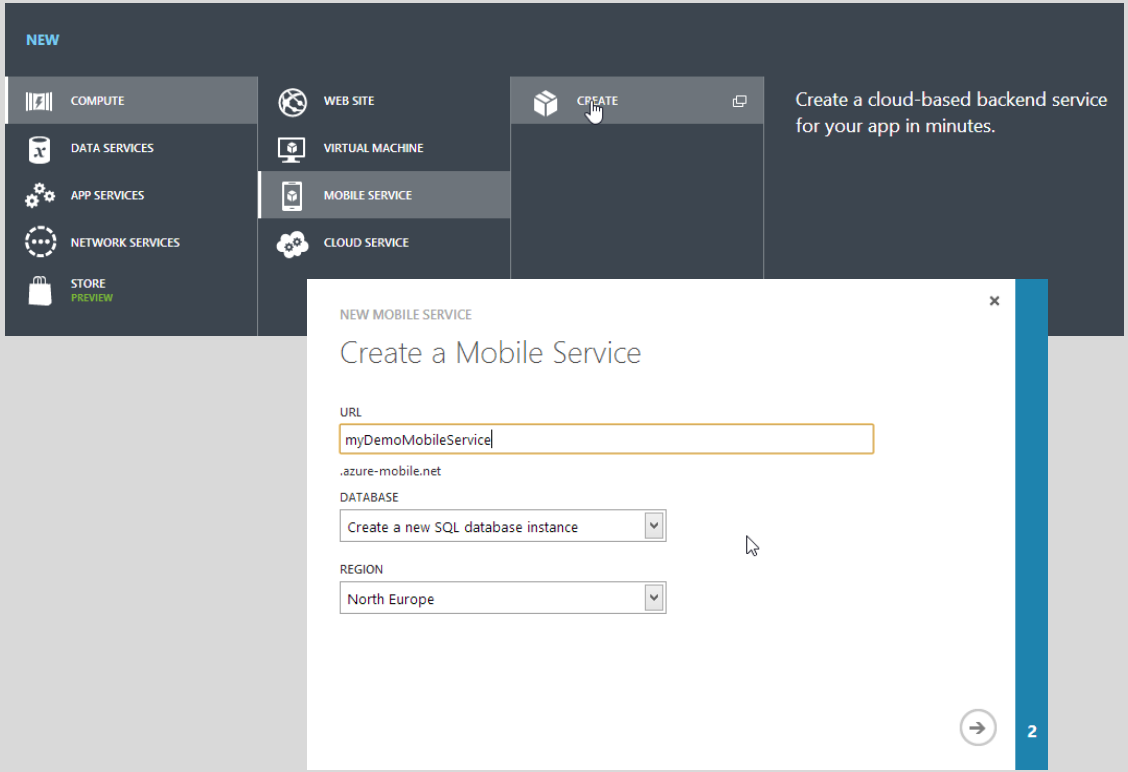
$http
Server Communication

Create Cloud Backend
- Azure Mobile Service

Access REST service using
- *$http* service

Unit testing with
- *$httpBackend*

Build UI with Bootstrap



Cloud Backend

Azure Mobile Services

Create a REST services backed by SQL Azure

https://manage.windowsazure.com

Create a table

Step 1: No protection

Step 2: Protection with API key

```
/// <reference
  path="../../../tsDeclarations/angularjs/angular.d.ts"/>

module MobileServicesDataAccess {
    export interface ITableRow {
        id?: number;
    }

    export interface ITable<T extends ITableRow> {
        query: (page?: number) => ng.IHttpPromise<IQueryResult<T>>;
        insert: (item: T) => ng.IHttpPromise<any>;
        update: (item: T) => ng.IHttpPromise<any>;
        deleteItem: (item: T) => ng.IHttpPromise<any>;
        deleteItemById: (id: number) => ng.IHttpPromise<any>;
    }

    export interface IQueryResult<T extends ITableRow> {
        results: T[];
        count: number;
    }
```

Access Class

REST Access Layer

Interface representing a single data row

id property needed for Azure Mobile Services

Interface for data access class for Azure Mobile Services

Note usage of TypeScript generics
Note promise API types

Helper interface for query result

Result (eventually filtered) and total server row count

```
export class Table<T extends ITableRow> implements ITable<T> {
    constructor(private $http: ng.IHttpService,
      private serviceName: string, private tableName: string,
      private pageSize: number, private apiKey: string) {
        // Set public methods using lambdas for proper "this" handling
        this.query = (page?) => this.queryInternal(page);
        this.insert = (item) => this.insertInternal(item);
        this.update = (item) => this.updateInternal(item);
        this.deleteItem = (id) => this.deleteItemInternal(id);
        this.deleteItemById = (id) => this.deleteItemByIdInternal(id);

        // Build http header with mobile service application key
        this.header = {
            headers: {
                "X-ZUMO-APPLICATION": apiKey
            }
        };
    }

    public query: (page?: number) => ng.IHttpPromise<IQueryResult<T>>;
    public insert: (item: T) => ng.IHttpPromise<any>;
    public update: (item: T) => ng.IHttpPromise<any>;
    public deleteItem: (item: T) => ng.IHttpPromise<any>;
    public deleteItemById: (id: number) => ng.IHttpPromise<any>;

    private header: any;
```

Access Class

REST Access Layer

Setting up the access class

```
        private queryInternal(page?: number):
           ng.IHttpPromise<IQueryResult<T>> {
            var uri = this.buildBaseUri()
                 + "?$inlinecount=allpages&$orderby=id";
            if (page !== undefined) {
                // Add "skip" and "top" clause for paging
                uri += "&$top=" + this.pageSize.toString();
                if (page > 1) {
                    var skip = (page - 1) * this.pageSize;
                    uri += "&$skip=" + skip.toString();
                }
            }
            return this.$http.get(uri, this.header);
        }
        private insertInternal(item: T): ng.IHttpPromise<any> {
            return this.$http.post(this.buildBaseUri(), item, this.header);
        }
        private updateInternal(item: T): ng.IHttpPromise<any> {
            var uri = this.buildBaseUri() + "/" + item.id.toString();
            return this.$http({ method: "PATCH", url: uri,
              headers: this.header, data: item });
        }
        private deleteItemInternal(item: T): ng.IHttpPromise<any> {
            return this.deleteItemByIdInternal(item.id);
        }
        private deleteItemByIdInternal(id: number): ng.IHttpPromise<any> {
            var uri = this.buildBaseUri() + "/" + id.toString();
            return this.$http.delete(uri, this.header);
        }
        private buildBaseUri(): string {
            return "https://" + this.serviceName + ".azure-mobile.net/tables/"
              + this.tableName;
        }
    }
}
```

Access Class

REST Access Layer

Accessing Azure Mobile Services

```
/// <reference path="../../../tsDeclarations/jasmine/jasmine.d.ts"/>
/// <reference path="../../../tsDeclarations/angularjs/angular.d.ts"/>
/// <reference path="../../../tsDeclarations/angularjs/angular-mocks.d.ts"/>
/// <reference
 path="../../../samples/communication/httpService/MobileServicesTable.ts"/>

interface IDummyRow extends MobileServicesDataAccess.ITableRow {
}

describe("Mobile Services Table Test", function () {
    var $http: ng.IHttpService;
    var $httpBackend: ng.IHttpBackendService;
    var table: MobileServicesDataAccess.ITable<IDummyRow>;
    beforeEach(inject((_$http_, _$httpBackend_) => {
        $http = _$http_;
        $httpBackend = _$httpBackend_;
        table = new MobileServicesDataAccess.Table<IDummyRow>(
           $http, "dummyService", "dummyTable", 10, "dummyKey");
    }));
    var dummyResult: MobileServicesDataAccess.IQueryResult<IDummyRow> =
        { results: [{ id: 1 }, { id: 2 }], count: 2 };

    it(' should query Azure Mobile Service without paging', () => {
        $httpBackend.whenGET("https://dummyService.azure-mobile.net
/tables/dummyTable?$inlinecount=allpages&$orderby=id")
            .respond(dummyResult);

        var result: IDummyRow[];
        table.query().success(r => {
            result = r.results;
        });
        $httpBackend.flush();
        expect(result.length).toEqual(2);
    });
```

Unit Tests

REST Access Layer

```
    ...

    it(' should issue a POST to Azure Mobile Service for insert', () => {
        $httpBackend.expectPOST("https://dummyService.azure-mobile.net
/tables/dummyTable")
            .respond(201 /* Created */);

        var data: IDummyRow = {};
        table.insert(data);
        $httpBackend.flush();
    });

    ...

    afterEach(() => {
        $httpBackend.verifyNoOutstandingExpectation();
        $httpBackend.verifyNoOutstandingRequest();
    });
});
```

Unit Tests

REST Access Layer

How Far?

What didn't we cover?

How far can it go?

Image Source:
http://flic.kr/p/765iZj

```
angular.module('MyReverseModule', [])
  .filter('reverse', function() {
    return function(input, uppercase) {
      var out = "";
      for (var i = 0; i < input.length; i++) {
        out = input.charAt(i) + out;
      }
      // conditional based on optional argument
      if (uppercase) {
        out = out.toUpperCase();
      }
      return out;
    }
  });
function Ctrl($scope) {
  $scope.greeting = 'hello';
}

<body>
    <div ng-controller="Ctrl">
      <input ng-model="greeting" type="greeting"><br>
      No filter: {{greeting}}<br>
      Reverse: {{greeting|reverse}}<br>
      Reverse + uppercase: {{greeting|reverse:true}}<br>
    </div>
  </body>
```

Filters

Standard and Custom Filters

Formatting filters

- currency
- date
- json
- lowercase
- number
- uppercase

Array-transforming filters

- filter
- limitTo
- orderBy

Custom filters (see left)

Source of custom filter sample: AngularJS docs

Advanced $http

- Interceptors
  Used e.g. for retry logic, authentication, etc.
- Support for JSONP
- For details see AngularJS docs

```
myModule.directive('button', function() {
  return {
    restrict: 'E',
    compile: function(element, attributes) {
      element.addClass('btn');
      if (attributes.type === 'submit') {
        element.addClass('btn-primary');
      }
      if (attributes.size) {
        element.addClass('btn-' + attributes.size);
      }
    }
  }
}
```

Directives

Custom Directives and Widgets

Not covered in details here

For details see AngularJS docs

Localization

- Internationalization (i18n)
  Abstracting strings and other locale-specific bits (such as date or currency formats) out of the application
- Localization (L10n)
  Providing translations and localized formats
- For details see AngularJS docs

Further Readings, Resources

- AngularJS Intellisense in Visual Studio 2012
  See Mads Kristensen's blog
- Recommended Book
  Kozlowski, Pawel; Darwin, Peter Bacon: Mastering Web Application Development with AngularJS
- Sample code from this presentation
  http://bit.ly/AngularTypeScript

Stop or Go?

Critical Evaluation

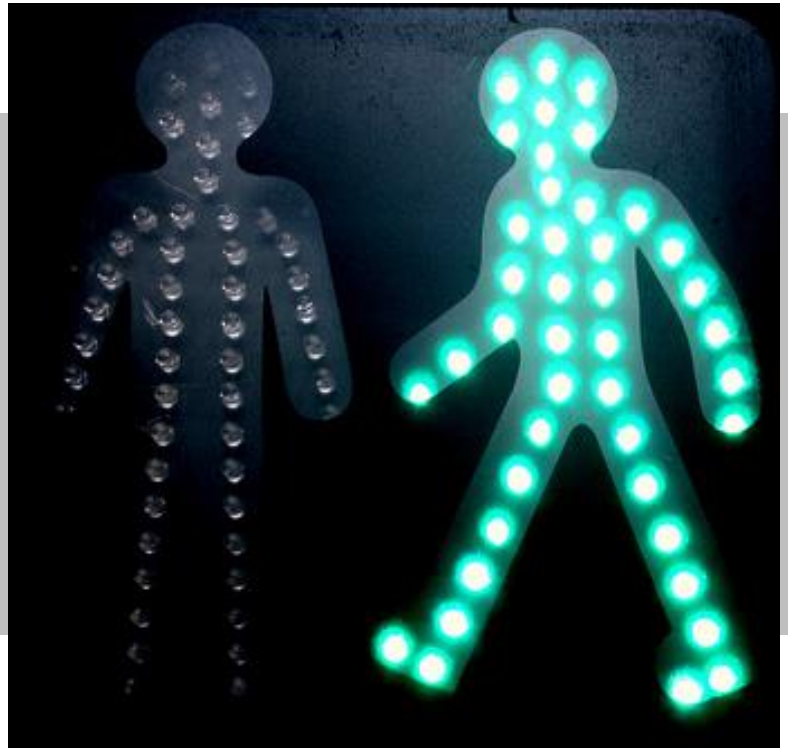


Image Source:
http://flic.kr/p/973C1u

Stop or Go?

- Many moving parts sometimes lead to problems
  - You have to combine many projects
  - Development tools
  - Services, UI components (directives, widgets), IDE/build components
- You still have to test on all target platforms
  - Operating systems
  - Browsers
- Learning curve for C#/.NET developers
  - Programming language, framework, runtime, IDE

Stop or Go?

- TypeScript for productivity
  - Type information helps detecting error at development time
- Clear separation between view and logic
  - Testability
  - Possible code reuse between server and client
- One framework covering many aspects
  - Less puzzle pieces
- Relatively large developer team
  - AngularJS by Google

Q&A

Thank your for coming!

Rainer Stropek

software architects gmbh

Mail rainer@timecockpit.com
Web http://www.timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

time cockpit is the leading time tracking solution for knowledge workers. Graphical time tracking calendar, automatic tracking of your work using signal trackers, high level of extensibility and customizability, full support to work offline, and SaaS deployment model make it the optimal choice especially in the IT consulting business.

Try time cockpit for free and without any risk. You can get your trial account at http://www.timecockpit.com. After the trial period you can use time cockpit for only 0,20€ per user and month without a minimal subscription time and without a minimal number of users.