
CAMPARI
FRATELLI CAMPARI SUCCESSORI

AngularJS

For Umbraco developers

Ready?

- Download: umbra.co/ngworkshop
- Unzip and try to run the /site folder with web matrix
- Get a workbook
- We'll be here for 3.5 hours, go get a cup of coffee

Agenda

- package.manifest
- Views
- Controllers
- Property editors
- Services aka factories aka resources
- Trees and editors
- Parameter editors
- 3rd party libraries
- Watchers

Angular is a framework for building Single page applications

But what does that tell us?

Stop touching the DOM

~~$(“input”).val(“woo”);~~

~~$(“a”).on(“click”...)~~

DATA

package.manifest

- Is the connection between Umbraco and angular
- Registers editors
- Loads files
- Is in /app_plugins/folder/

```
{
    propertyEditors:[

        {
            name: “My editor”,
            alias: “per.my.editor”,
            editor:{
                view: “~/app_plugins/first/editor.html”
            }
        }


    ]
}
```

```
<input type="text" ng-model="model.value" />

<h1>{{model.value}}</h1>
```

Exercise 1

- package.manifest
- propertyEditors:[]
- view
- ng-model / {{ }} aka databinding

Controllers

- The codebehind of a view
- its .js file must be registered in package.manifest
- registers with ng-controller
- gives access to $scope
  - $scope is our context / view model
  - Contains values and functions which can be reached from the view

```
angular.module(“umbraco”)
    .controller(“My.EditorController”,

        function($scope){

            $scope.model.value = “Hey there”;

            $scope.sayHi = function(){
                $scope.model.value = “HI!”;
            };
        }
);
```

```
<div ng-controller="My.EditorController">

      <h1>{{model.value}}</h1>

      <button ng-click="sayHi()">Say hi</button>

      <input type="text" ng-model="model.value" />
</div>
```

Exercise 2

- add a controller file - register it in the manifest
- put values on $scope
- trigger methods with ng-click
- apply css with ng-class

Services

- Shared objects between controllers
- Is a "singleton" - there can be only one!
- Exposed to controllers through dependency injection
- **Naming**: in umbraco we call services that interact with the database “resource”

```
angular.module(“umb”).factory(“campariService”,
    function(){

        var myService = {};
        myService.archEnemy = “Bacardi Breezer”;

        myService.isCampariGreat = function(){
            return true;
        };

        return myService;

});
```

Dependency Injection?

- Easy way to automatically instantiate objects when we ask for them by name.

- Angular does this is a really really simple way

```
angular.module(“umb”).controller(“my.ctrl”,
    function($scope, campariService){
            var t = campariService.isCampariGreat();

});
```

Exercise 3

- Register a service
- use $http to call a serverside rest api
- Inject as a dependency in the editor controller

Exercise 4

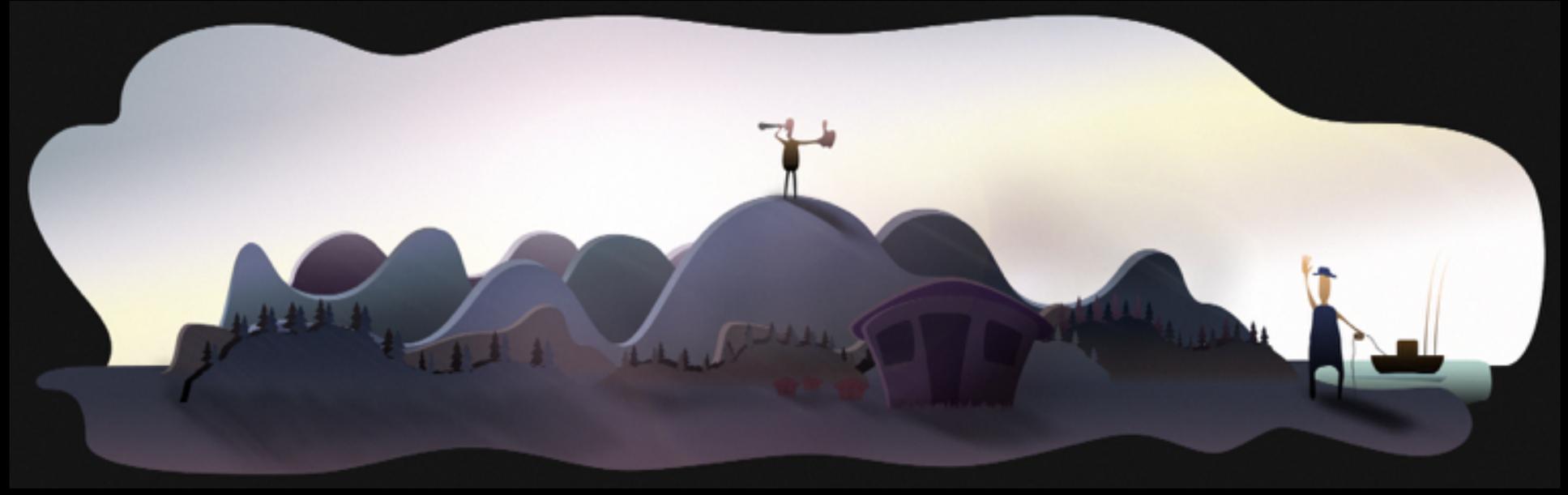
- Select ingredient to put it on model.value
- Render model.value as a table
- Remove an item at a given index with $index

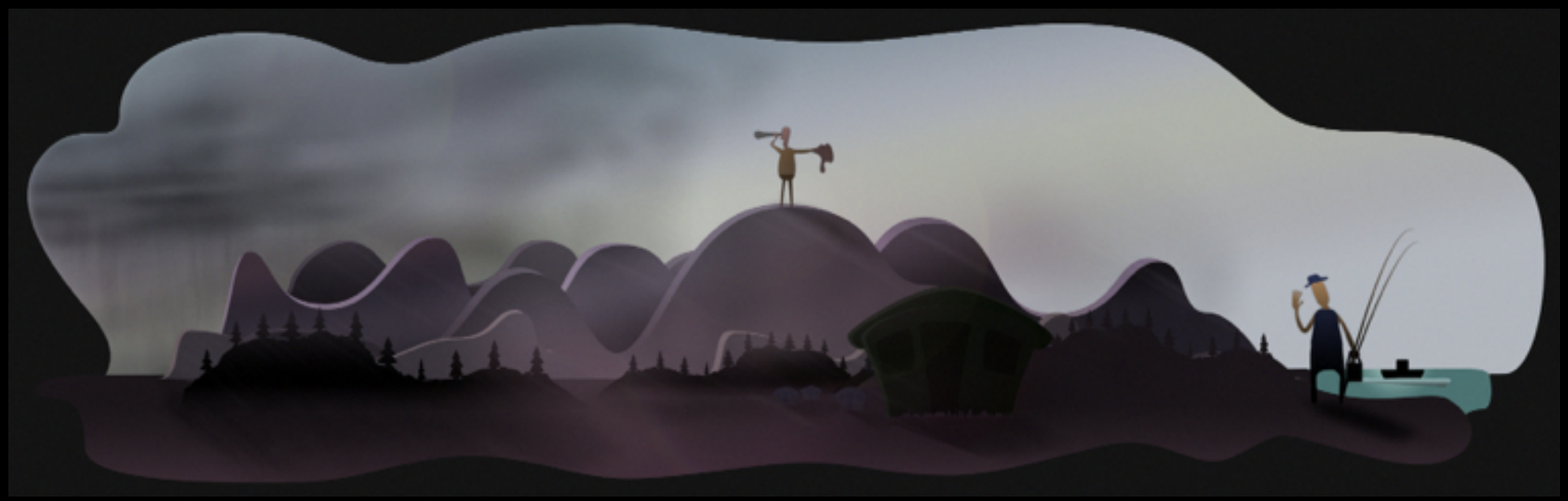
Exercise 5

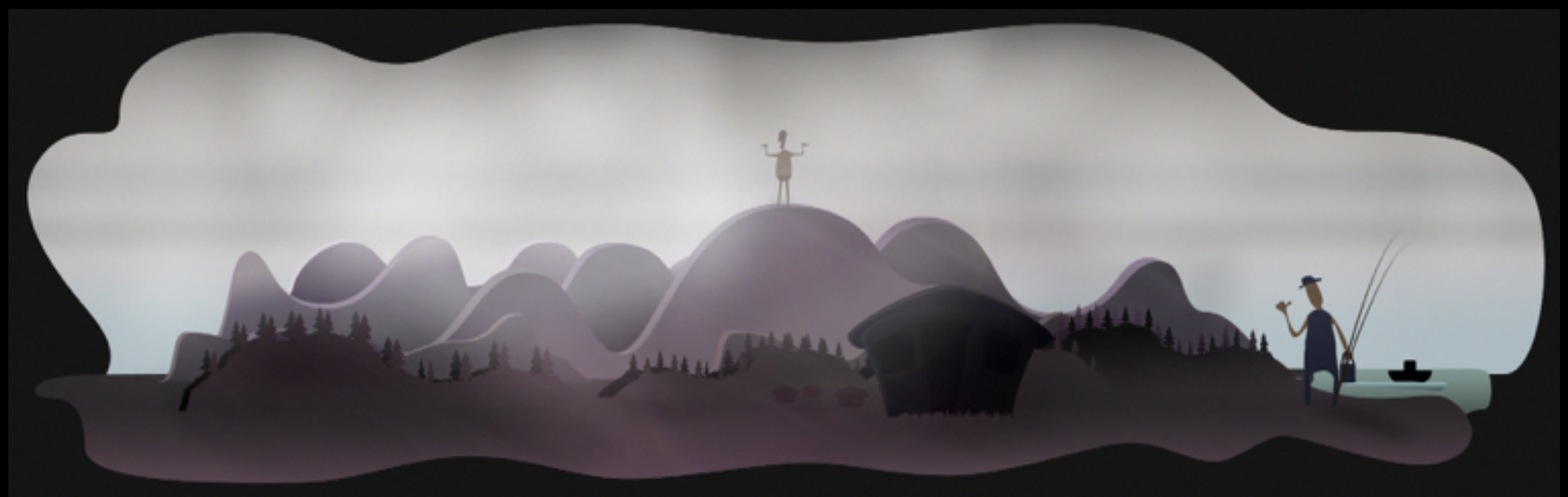
- Ingredient.cs model
- IngredientApiController.cs controller
- IngredientTreeController.cs tree

Promises

- Pattern to handle async operations
- In the old days (6 months ago), this was handled with callbacks
- A promise will always return something, either it will succeed or fail, eventually.







$http

```
$http.get(“url”).then(function(response){
    //on success
    response.data…
}, function(error){
    //on error
});
```

$q

```
function asyncOperation(){
    var deferred = $q.defer();
    $timeout(function(){
        if(var)
            deferred.resolve(“yeeeaaaaah”);
        else
            deferred.reject(“boooooh”);
    });
    return deferred;
}

asyncOperation().then(function(response){
        //response is “yeeaaahhh”;
});
```

Exercise 6

- Ingredient editor
- Follow conventions to easily wire it up
- Get ingredient by ID
- Modify and store
- Working with promises

navigationService

```
navigationService.hideDialog();


navigationService.changeSection(“media”);
```

```
navigationService.showSearch();


navigationService.syncTree(
                {tree: 'content',
                path: ["-1","123d"],
                forceReload: true});
```

treeService

```
treeService.reloadNode($scope.currentNode);

treeService.removeNode($scope.currentNode);

treeService.removeChildNodes($scope.currentNode);

treeService.getTreeRoot($scope.currentNode);

treeService.getMenu($scope.currentNode);
```

Exercise 7

- Delete ingredient editor
- Communicate with the tree for showing loading
- Removing from tree
- Toggling the dialog

```
propertyEditors:[
    {
        name: "Ingredient Editor",
        editor: {…}
        ,
        prevalues: {
            fields:[
                {
                    label: "Hide quantity field",
                    description: "desc",
                    key: "hideQuantity",
                    view: "boolean"
                }
            ]
        }
    }
]
```

$scope.model.config

```
<div ng-if="model.config.hideQuantity == '1'">
</div>
```

Exercise 8

- Property Editor configuration
- Prevalue Editors
- model.config
- modify the view

isParameterEditor

```
propertyEditors:[
    {
        name: "My first editor",
    }
],
css: [
    “/app_plugins/first/style.css”
]
```

Exercise 9

- Loading a stylesheet

isParameterEditor

```
propertyEditors:[
    {
        name: "My first editor",
        alias: "My.First",
        isParameterEditor: true,
        editor:{
            view: "~/app_plugins/first/editor.html"
        }
    }
]
```

Exercise 10

- Enable a property editor as a parameter editor

Assets Service

```
assetsService.loadJs(“/js/file.js”)
.then(function(){
    alert(“im done!);

});
```

Watchers

```
$scope.$watch(“model.value”, function(){});
```

```
$scope.$watchCollection(“model.value”, function()
{});
```

```
$scope.$watch(“model.value”, function(){}, true);
```

$watch

```
$scope.$watch(“model.value”, function(new, old){

    //perform whatever you want here on changes
    //compared new and old if you need to

}, true);
```

Exercise 11

- assetsService

- $scope.$watch