Fork me on GitHub

AngularJS

Fabian Imsand

https://github.com/fims/angular

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
University of Applied Sciences Western Switzerland

Overview

Your experience

- ☐ jQuery
- ☐ AngularJS
- ☐ Angular 2
- ☐ React
- ☐ Backbone
- ☐ Twig (PHP)
- ☐ Sencha
- ☐ Others?

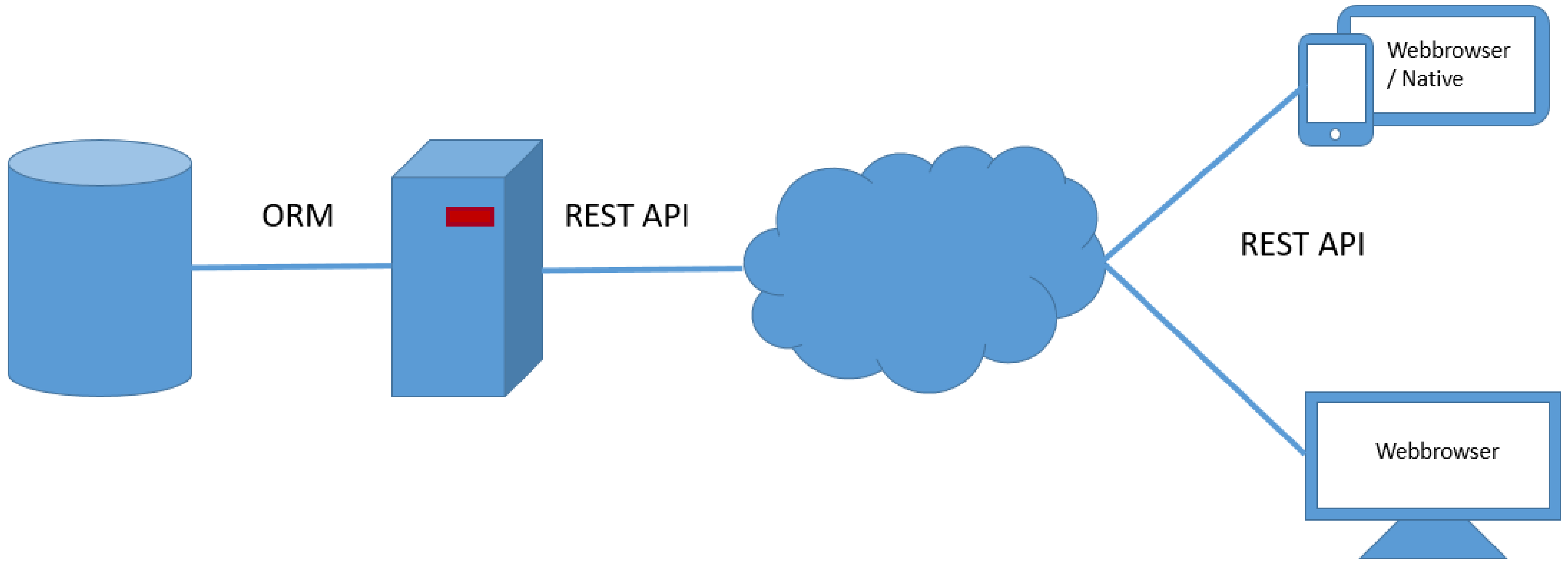
AngularJS 1 vs Angular 2

- Component-Based
  - Controller and $scope no longer used
  - Use components instead
- Improved dependency injection
- TypeScript
  - ES6 + Types + Annotations
  - Open Source (Microsoft)
- Lambdas
- Angular wants to add to what browsers can do, not replicate it
  - Following standardization plans
  - Looking towards the future

Angular 2 is awesome, but...

- Is currently a RC
- Way more libraries for AngularJS
- Smaller community (growing)
- AngularJS is «Industry standard»

Full stack design



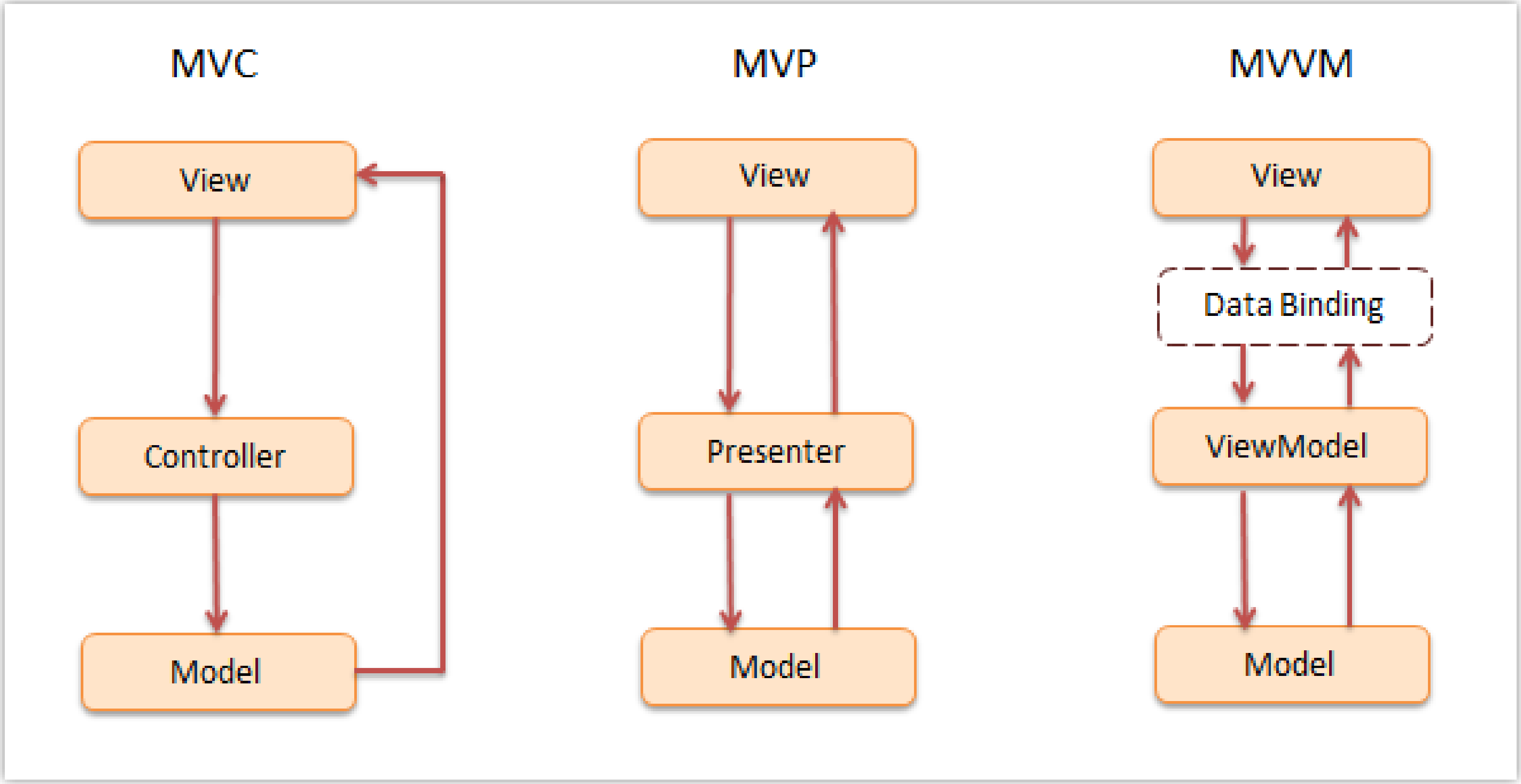
REST

- HTTP for calls between machines / devices
- Single page
- Web Services
- JSON
- Call on Objects
  - /categories/1/articles/1
- GET-> receive data
- POST -> create a new resource
- PUT -> update an existing resource or create a new one
- DELETE -> delete a resource
- Status codes -> 200 / 201 / 400 / 401 / 403 / 404 / 500

Introduction to AngularJS

- Structural framework for dynamic web apps
- Extend HTML‘s syntax
- Normaly
  - Library (jQuery)
  - Framework (Ember)
  - Angular creates new HTML constructs
- Complete client-side solution
- {{}}
- MVVW / MV*
- Directives
- ng-* (https://code.angularjs.org/1.5.1/docs/api)
- Step-0: The one where we start

MVC / MVP / MVVM

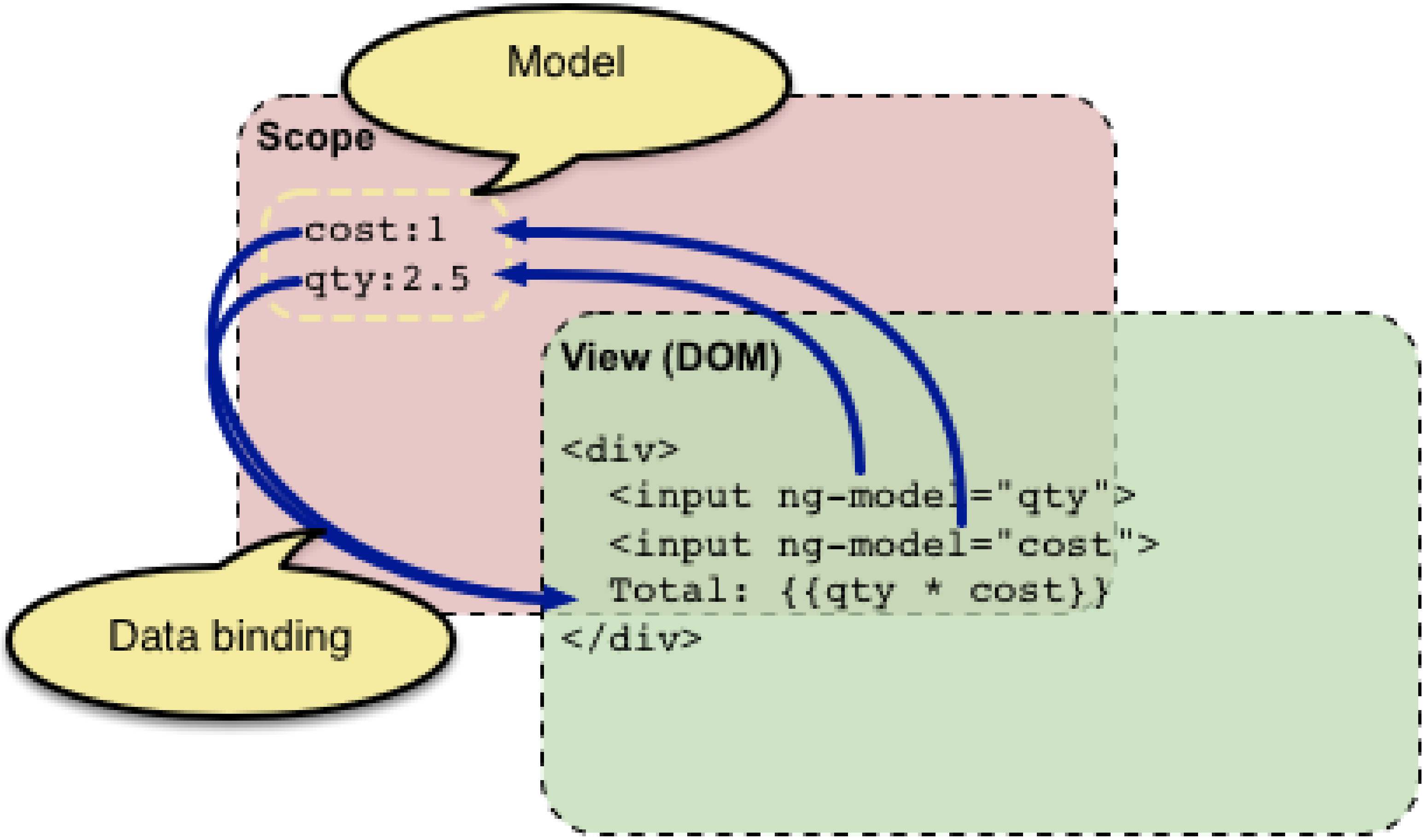

MVC
View
Controller
Model
MVP
View
Presenter
Model
MVVM
View
Data Binding
ViewModel
Model

Tips, Tricks, Best Practices

- Inspector
- JSON Lint
- Gulp, Grunt, Bower, NPM, Yeoman
- REST Console (e.g. Postman)
- Twitter Bootstrap
- Font awesome
- Step-1: The one where we set up our basic project

Conceptual overview

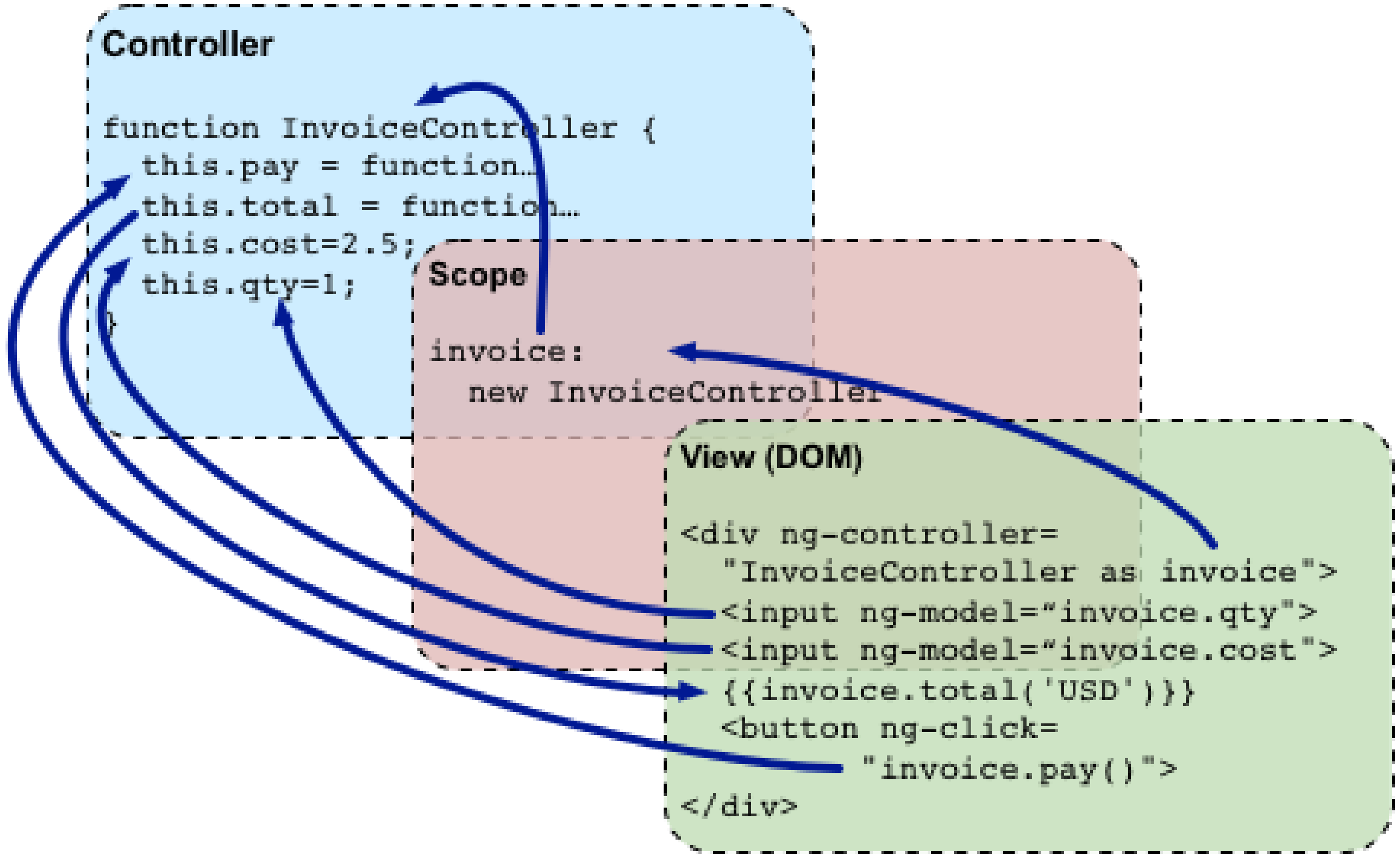
- Data binding



```
<div>
  <input ng-model="qty">
  <input ng-model="cost">
  Total: {{qty * cost}}
</div>
```

Conceptual overview

- Controllers



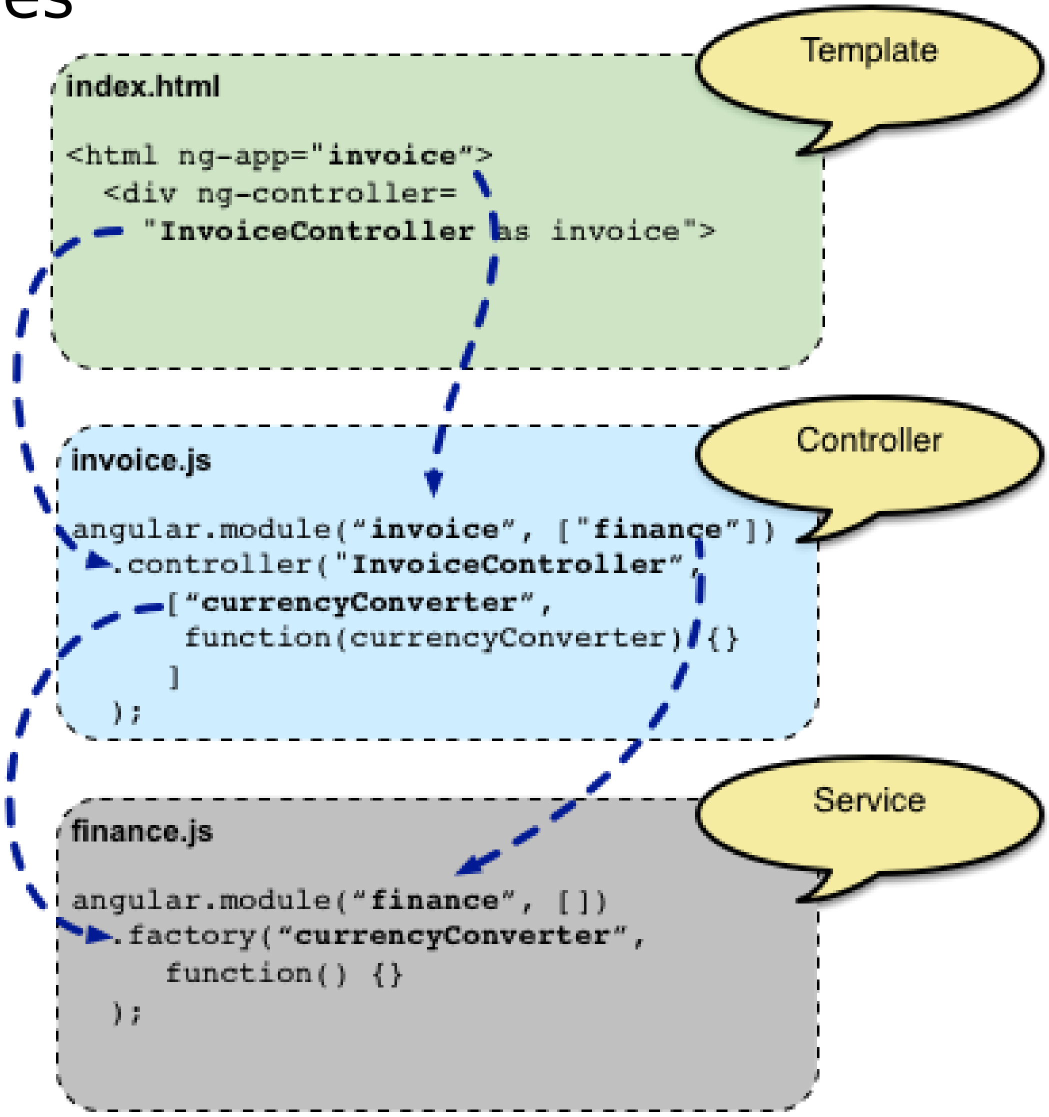
```
function InvoiceController {
  this.pay = function…
  this.total = function…
  this.cost=2.5;
  this.qty=1;
}
```

```
invoice:
  new InvoiceController
```

```
<div ng-controller=
  "InvoiceController as invoice">
  <input ng-model="invoice.qty">
  <input ng-model="invoice.cost">
  {{invoice.total('USD')}}
  <button ng-click=
         "invoice.pay()">
</div>
```

Conceptual overview

- Services



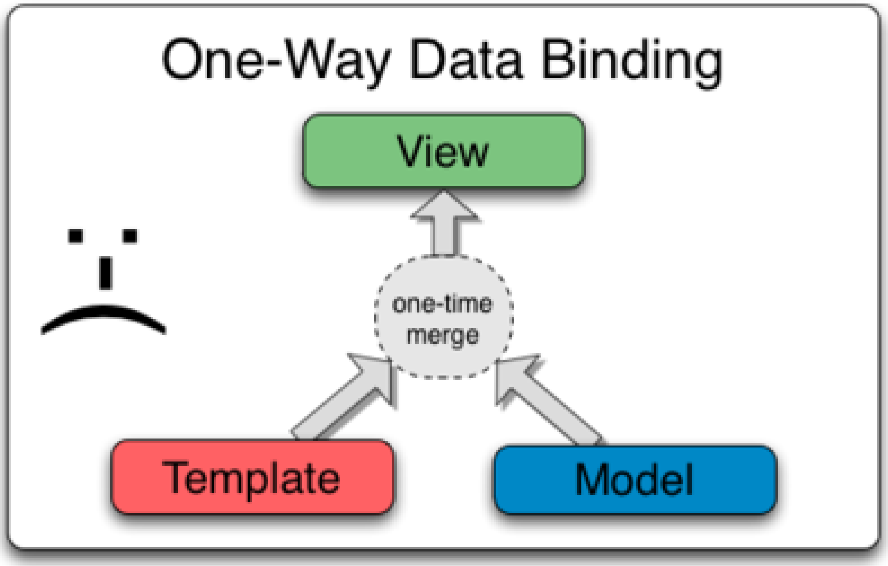
```
<html ng-app="invoice">
  <div ng-controller=
     "InvoiceController as invoice">
```

```
angular.module("invoice", ["finance"])
  .controller("InvoiceController",
    ["currencyConverter",
     function(currencyConverter) {}
    ]
  );
```

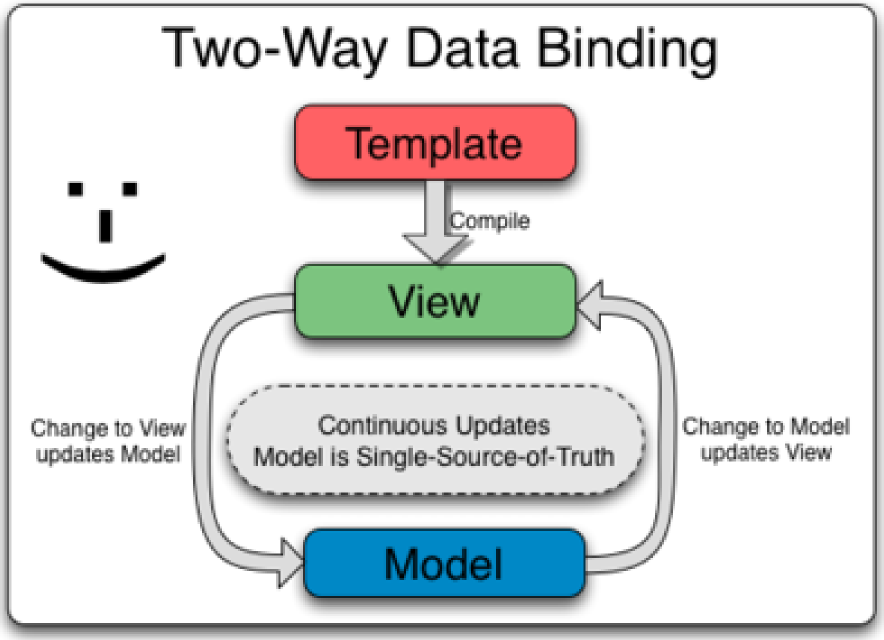
```
angular.module("finance", [])
  .factory("currencyConverter",
    function() {}
  );
```

Data Binding

Classical Template Systems



Angular Templates


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model updates View
Model

- Step-2: The one with the basic data binding

Modules

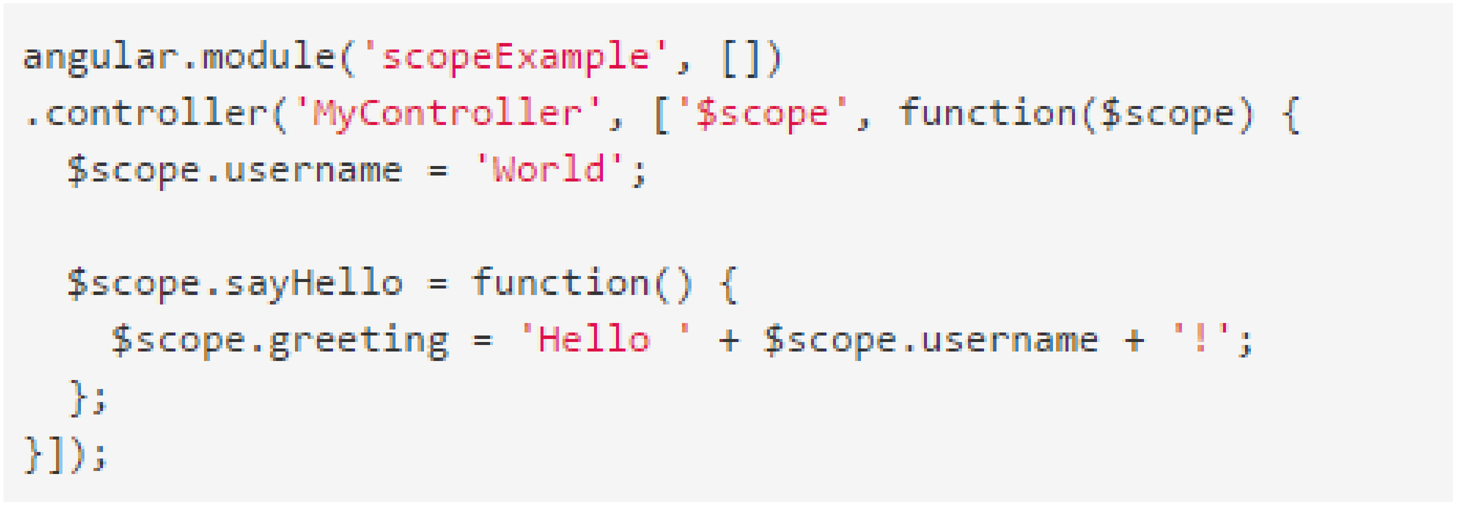
- Smaller tasks
- Distributed development
- Maintainability
- Reuse modules
  - Authentication / Login
  - Search / Filter
  - Core
  - Shop
  - ...
- Dependency injection

Controllers

- JavaScript constructor function
- Each controller has it‘s own scope
  - $scope
  - controller as -> instance assigned to a prototype on the new scope
- Should contain only business logic
- Keep Controllers slim
- Use services in Controllers with dependency injection
- Best practices
  - Don‘t use $scope -> controller as
  - Don‘t use ng-controller -> routing
- Step-3: The one with our first controller

Scopes

- Object
- Referes to application model
- Hirarchical structure (minic the DOM structure)
- Glue between application controller and view



```
angular.module('scopeExample', [])
.controller('MyController', ['$scope', function($scope) {
  $scope.username = 'World';

  $scope.sayHello = function() {
    $scope.greeting = 'Hello ' + $scope.username + '!';
  };
}]);
```

Dependency injection

- Design pattern
- Angular injector
  - Creates components
  - Resolves their dependencies
  - Provides them
- Dependency is an object
  - Can be used
  - “service”
- IMPLICIT ANNOTAION!!! (minify)
- Easier to test

Services

- Organize code
- Separate code
- Share code
- Lazily instantiated
- Singlestons
- Step-4: The one where we get data

Service vs factory vs provider

- All of the 3 are singletons
- Services
  - module.service(`serviceName‘, function);
  - Provides an instance of the function
- Factories
  - module.factory(`factoryName`, function);
  - Returns a new object
- Providers
  - module.provider(`providerName`, function);
  - Creates first a new object
  - And calls then a get function

Exercice 1

- Add the property “done” to the ToDo objects
- Display ToDos as “done” – Line through
- Change the
- Step-5 The one with the first exercise

More REST

- Step-6: The one where we aren’t RESTless anymore

Routing

- Navigation
- “Links” views with controller
- Step-7: The one where we navigate

Testing

- Help you understand the problem
- Isolate the unit
  - No DOM manipulation
  - Nu HTTP requests
- Easy to mock
- Dependency Injection
- Karma
  - Executes the tests
  - Like a web server
- Jasmine
  - Behavior driven development
- Step-8: The one where we test our application

AngularJS Best Practices

- Don’t use $scope -> controller as
- Don’t use ng-controller
- Use components instead of directives
- In most cases you can use services (HTTP)
- Don’t try to fix stuff with jQuery
- Use ng-* (https://code.angularjs.org/1.5.1/docs/api)
- Try to use libraries

Address book

- The one where it all comes together
- Person
  - CRUD
  - List / detail view
- Group
  - CRUD
  - Contains people