Information systems modeling

Tomasz Kubik

Spring annotation

Spring MVC and REST Annotations

- @Controller
  - used on classes playing role of controllers (capable of handling multiple request mappings); to enable autodetection the configuration must have proper settings for component scanning
- @RequestMapping
  - used both at class and method level
  - maps web requests onto specific handler classes and handler methods
  - when used on class level it creates a base URI for which the controller will be used.
  - when used on methods it will give the URI on which the handler methods will be executed
  - its `method` attribute with an HTTP method value narrows down the HTTP methods to invoke
- @CookieValue
  - used at method parameter level in the method annotated with @RequestMapping,
  - The HTTP cookie is bound to the @CookieValue parameter for a given cookie name
- @CookieValue
  - used at method parameter level in the method annotated with @RequestMapping,
  - The HTTP cookie is bound to the @CookieValue parameter for a given cookie name
- @CrossOrigin
  - used both at class and method level to enable cross origin requests (helpful in cases when different servers serve data and scripts, see Cross Origin Resource Sharing (CORS)),

https://springframework.guru/spring-framework-annotations/

Spring annotation

Composed @RequestMapping Variants (introduced in Spring framework 4.3 in order to better express the semantics of the annotated methods, can be used with Spring MVC and Spring WebFlux)

- @GetMapping
  - used for mapping HTTP GET requests onto specific handler methods
  - shortcut for @RequestMapping(method = RequestMethod.GET)
- @PostMapping
  - used for mapping HTTP POST requests onto specific handler methods
  - shortcut for @RequestMapping(method = RequestMethod.POST)
- @PutMapping
  - used for mapping HTTP POST requests onto specific handler methods
  - shortcut for @RequestMapping(method = RequestMethod.PUT)
- @PatchMapping
  - used for mapping HTTP POST requests onto specific handler methods
  - shortcut for @RequestMapping(method = RequestMethod.PATCH)
- @DeleteMapping
  - used for mapping HTTP POST requests onto specific handler methods
  - shortcut for @RequestMapping(method = RequestMethod.DELETE)

https://springframework.guru/spring-framework-annotations/

Spring annotation

- @ExceptionHandler
  - used at method levels to handle exception at the controller level
- @InitBinder
  - used at method level
  - plays the role of identifying the methods which initialize the WebDataBinder - a DataBinder that binds the request parameter to JavaBean objects
- @Mapping
  - meta annotation that indicates a web mapping annotation.
- @Mappings .
  - configure mappings of source fields to their target fields (needed when beens have different field names and cannot be mapped automatically)
  - accepts an array of *@Mapping* annotation which we will use to add the target and source attribute)
- @MatrixVariable
  - used to annotate request handler method arguments so that Spring can inject the relevant bits of matrix URI

https://springframework.guru/spring-framework-annotations/

Spring annotation

- @PathVariable
  - used to annotate request handler method arguments
  - applies if certain URI value acts as a parameter (it can be specified using a regular expression).
- @RequestAttribute
  - used to bind the request attribute to a handler method parameter
  - used to access the objects which have been populated on the server side
- @RequestBody
  - used to annotate request handler method arguments
  - indicates that a method parameter should be bound to the value of the HTTP request body
- @RequestHeader
  - used to annotate request handler method arguments
  - maps controller parameter to request header value
- @RequestParam
  - used to annotate request handler method arguments
  - helps to retrieve the URL parameter and map it to the method argument
- @RequestPart
  - used to annotate request handler method arguments
  - can be used instead of @RequestParam to get the content of a specific multipart and bind to the method argument annotated with @RequestPart.
- @ResponseStatus
  - used on methods and exception classes marking them with a status code and a reason that must be returned.

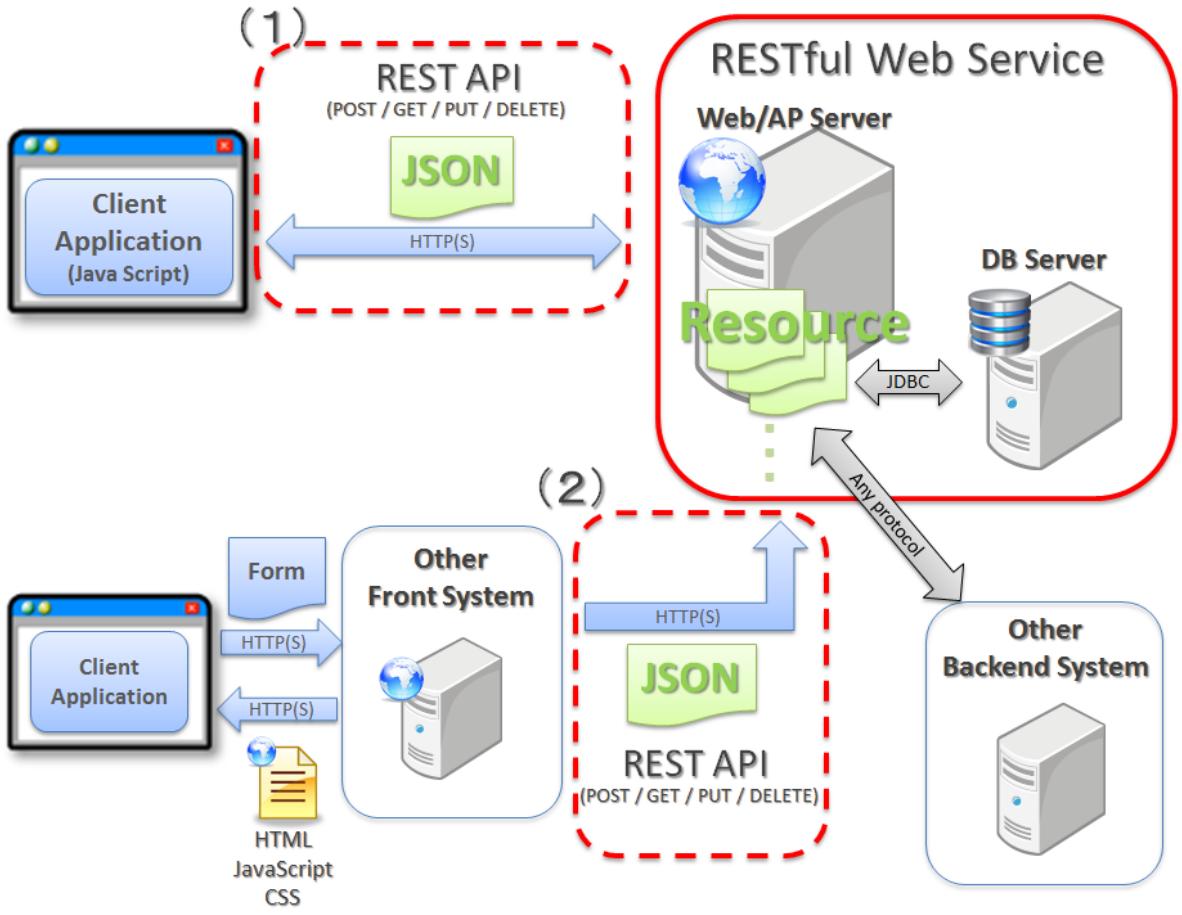
https://springframework.guru/spring-framework-annotations/

Spring annotation

- @ControllerAdvice
  - applied at the class level
  - used to define @ExceptionHandler, @InitBinder and @ModelAttribute methods that apply to all @RequestMapping methods
- @RestController
  - used at the class level
  - marks the class as a controller where every method returns a domain object instead of a view
  - a convenience annotation which combines @Controller and @ResponseBody
- @RestControllerAdvice
  - applied on Java classes and used along with the @ExceptionHandler annotation to handle exceptions that occur within the controller
  - convenience annotation which combines @ControllerAdvice and @ResponseBody.
- @SessionAttribute
  - used at method parameter level to bind the method parameter to a session attribute
- @SessionAttributes
  - applied at type level for a specific handler
  - used when you want to add a JavaBean object into a session
- @ResponseBody
  - used to annotate request handler methods
  - indicates that the result type should be written straight in the response body in whatever format you specify like JSON or XML. Spring converts the returned object into a response body by using the HttpMessageConveter.

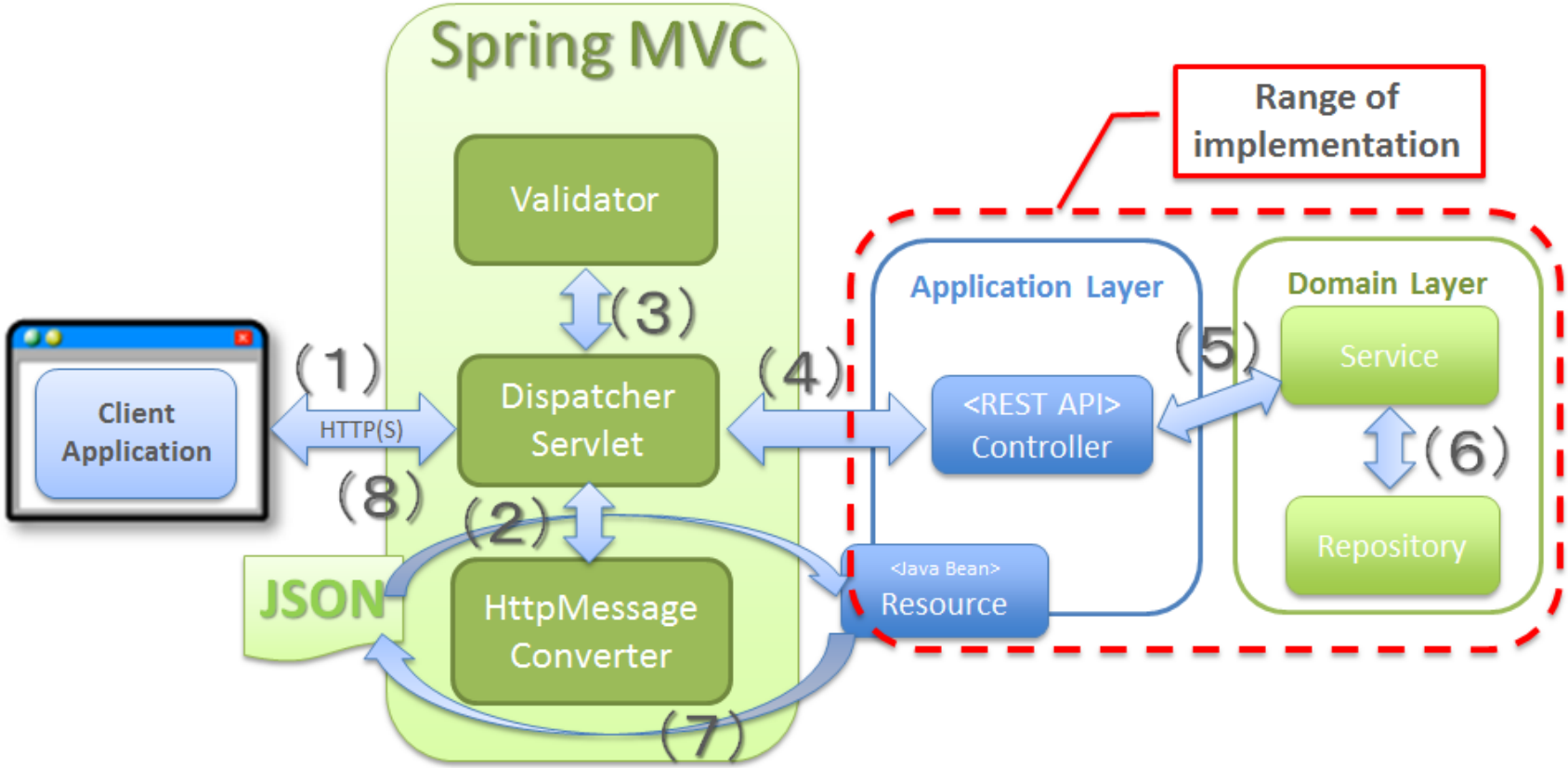
https://springframework.guru/spring-framework-annotations/

Architecture for building a RESTful Web Service



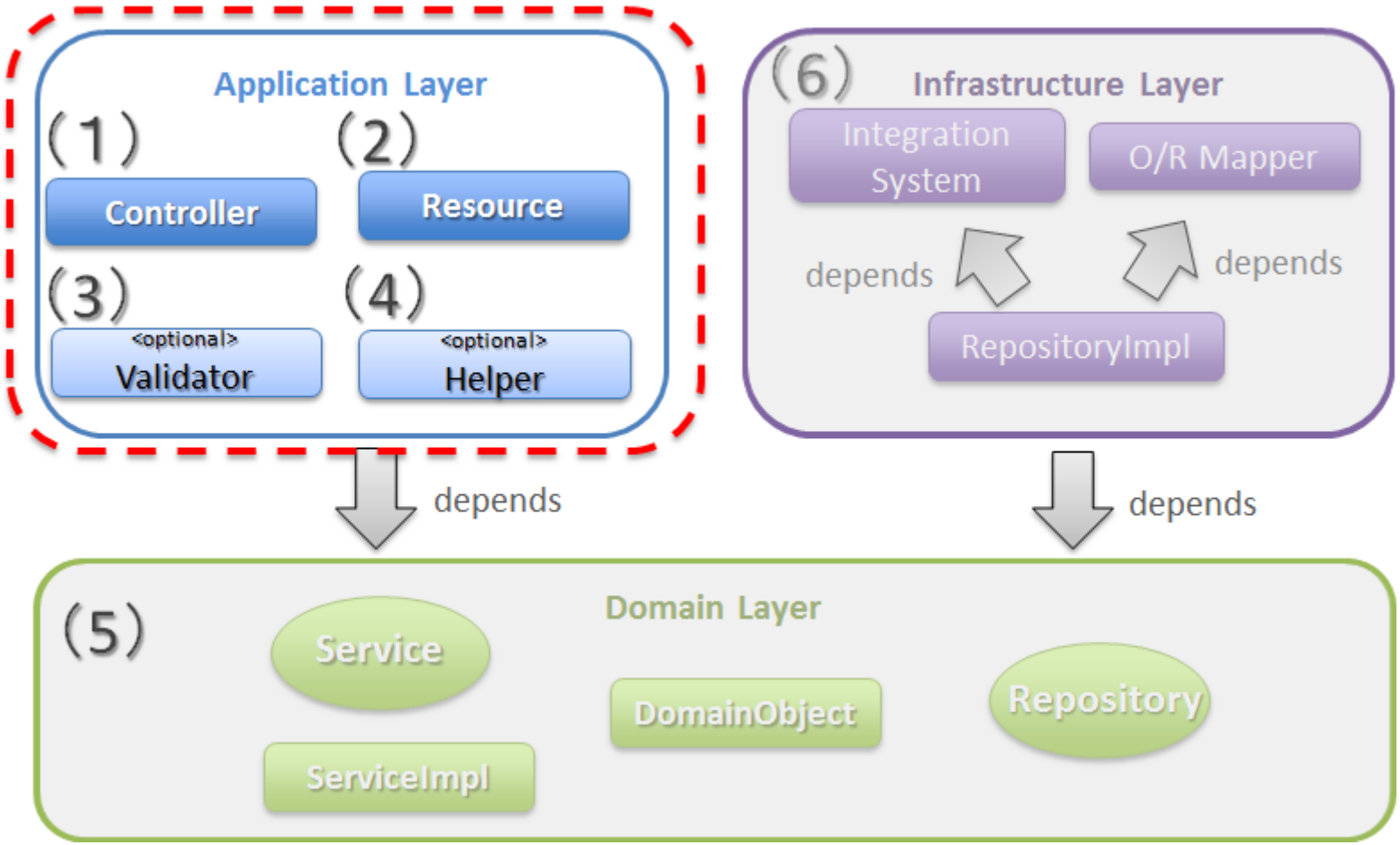
https://terasolunaorg.github.io/guideline/5.2.0.RELEASE/en/ArchitectureInDetail/WebServiceDetail/REST.html#restaboutresourceorientedarchitecturex

RESTful Web Service development



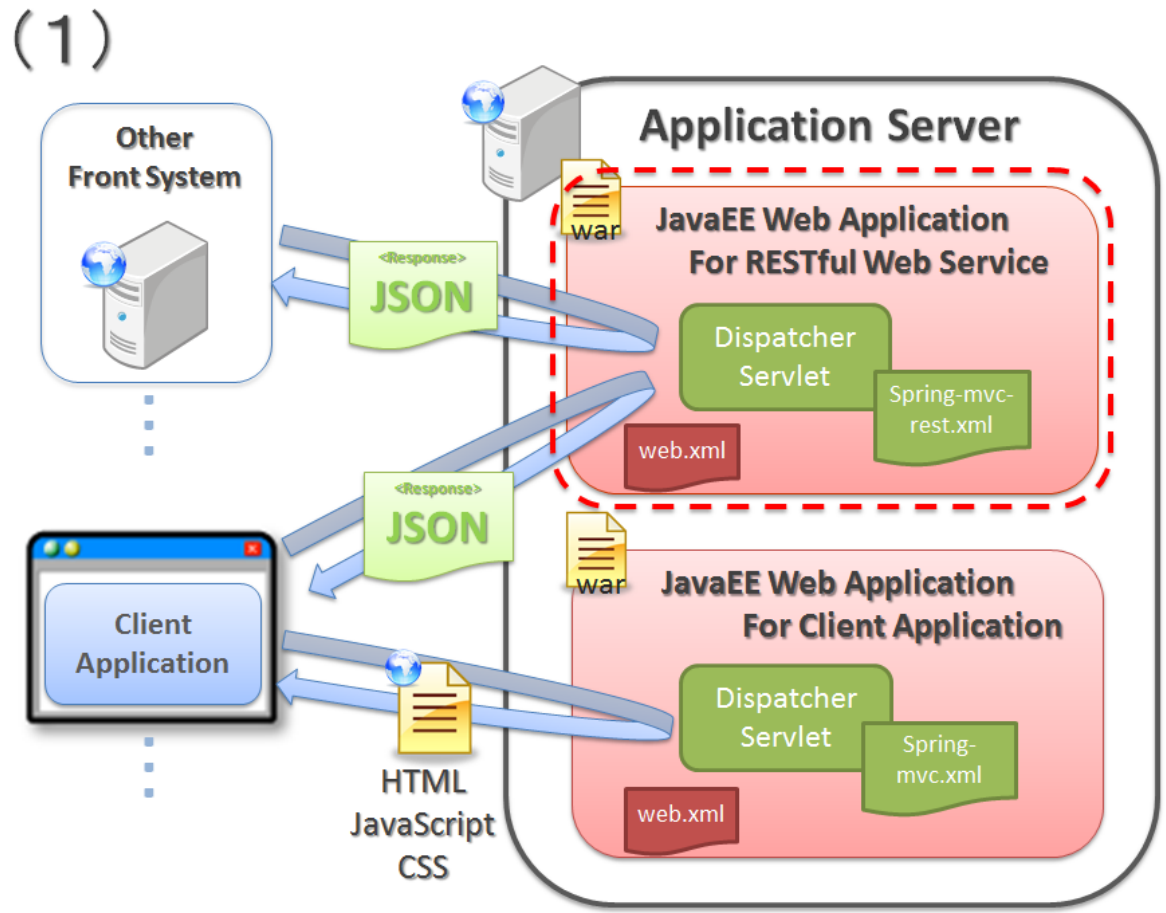
https://terasolunaorg.github.io/guideline/5.2.0.RELEASE/en/ArchitectureInDetail/WebServiceDetail/REST.html#restaboutresourceorientedarchitecturex

Configuration for RESTful Web Service module



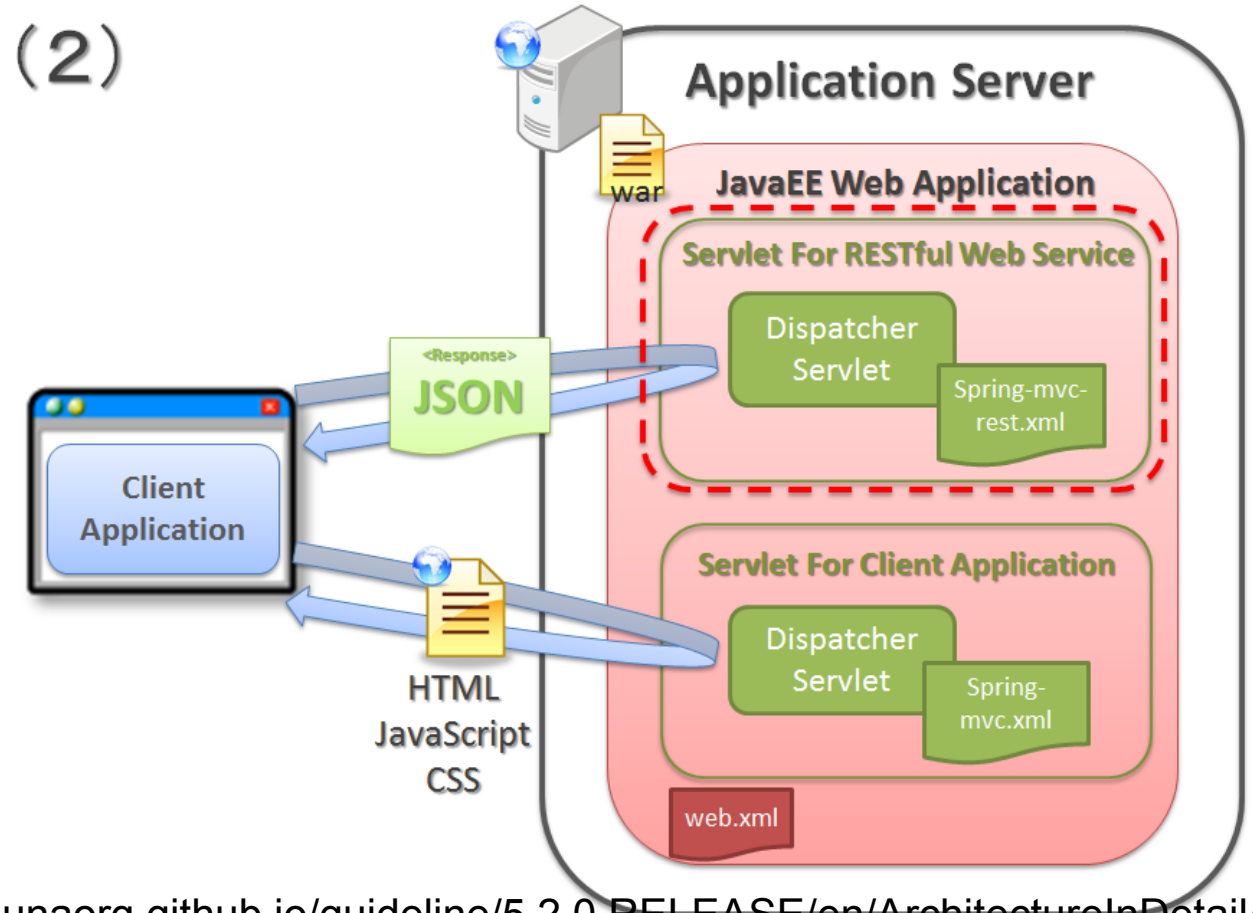
https://terasolunaorg.github.io/guideline/5.2.0.RELEASE/en/ArchitectureInDetail/WebServiceDetail/REST.html#restaboutresourceorientedarchitecturex

Web application exclusive to RESTful Web Service



https://terasolunaorg.github.io/guideline/5.2.0.RELEASE/en/ArchitectureInDetail/WebServiceDetail/REST.html#restaboutresourceorientedarchitecturex

RESTful Web Service and client application as a single application



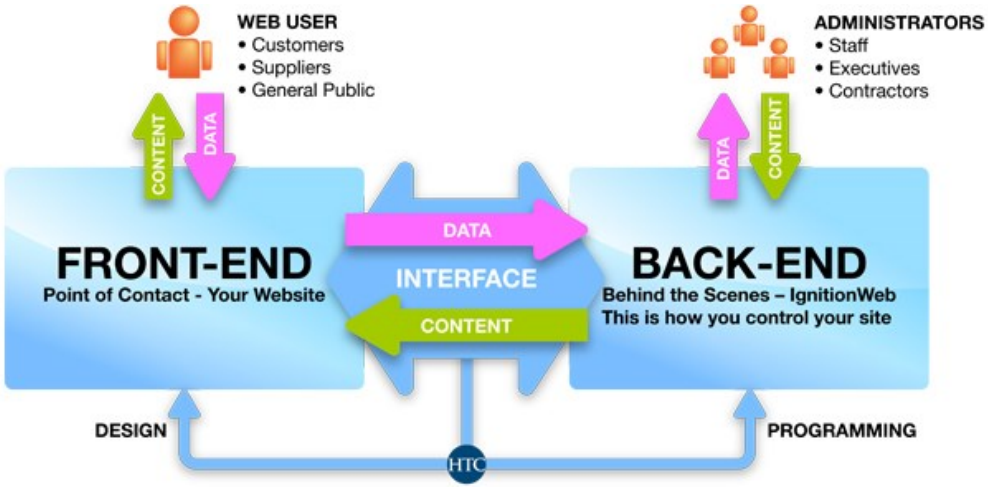
https://terasolunaorg.github.io/guideline/5.2.0.RELEASE/en/ArchitectureInDetail/WebServiceDetail/REST.html#restaboutresourceorientedarchitecturex

Cache-Control

- **Cache-Control: Part of Hypertext Transfer Protocol -- HTTP/1.1**
  - https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9
- **Spring MVC - Cache-Control support**
  - https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/cache-control.html
- **HTTP-304 Conditional Cache Control using Spring MVC**
  - https://medium.com/simars/http-304-conditional-caching-in-rest-api-using-spring-mvc-ae49f95367de
- **How to enable HTTP response caching in Spring Boot**
  - https://stackoverflow.com/questions/24164014/how-to-enable-http-response-caching-in-spring-boot
- **HTTP cache with Spring examples**
  - http://dolszewski.com/spring/http-cache-with-spring-examples/

Web applications development

- Front-end Development
  - manages everything that users visually see first in their browser or application
  - developers are responsible for the look and feel of a site
  - languages
    - HTML, CSS, and Javascript.
- Back-end Development
  - refers to the server side of development
  - developers are primarily focused on how the site works
  - languages
    - Java, PHP, Ruby on Rails, Python, and .Net
- Full-stack Development
  - developer masters both aspects of creating and maintaining a website



https://www.aog.jobs/blog/frontend-vs-backend-web-development-whom-do-you-need-for-your-project/

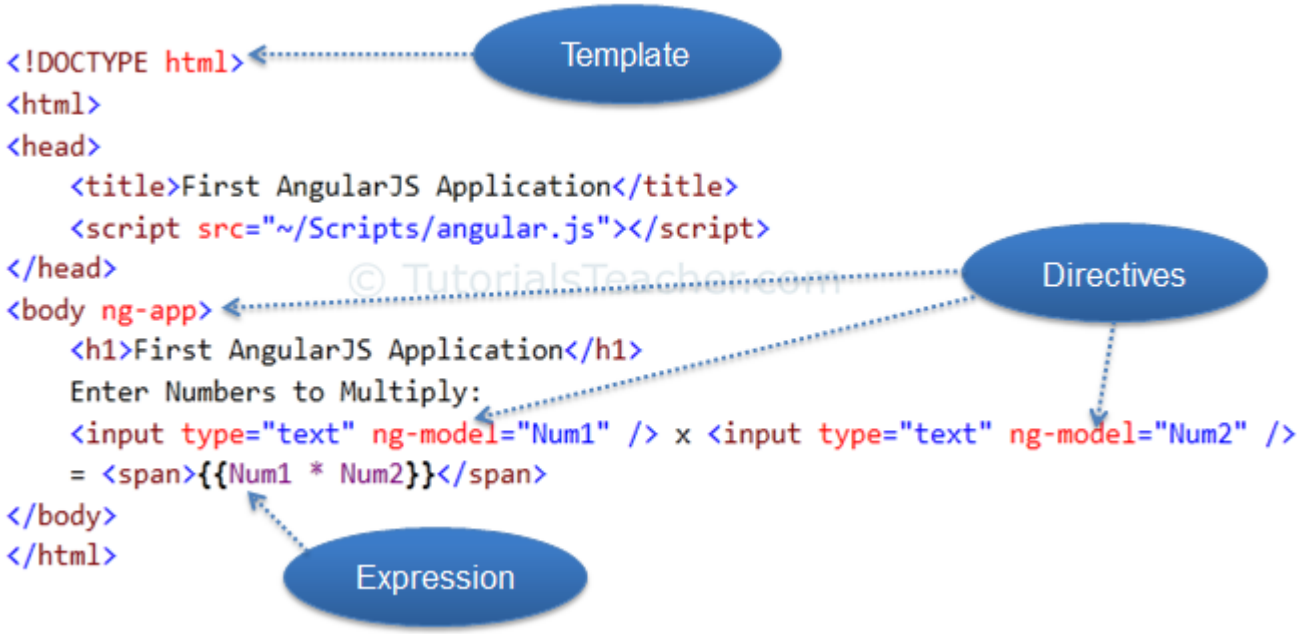
Back-end programming in Spring

- Spring
  - provides comprehensive infrastructure support for developing Java applications
  - is packed with some nice features like Dependency Injection and out of the box modules like:
    - Spring JDBC
    - Spring MVC
    - Spring Security
    - Spring AOP
    - Spring ORM
    - Spring Test
- Spring boot
  - an extension of the Spring framework that elliminates the boilerplate configurations required for setting up a Spring application.
  - paved the way for a faster and more efficient development eco-system.
  - offers, among the others:
    - opinionated ‘starter’ dependencies to simplify build and application configuration
    - embedded server to avoid complexity in application deployment
    - metrics, helth check, and externalized configuration
    - automatic config for Spring functionality – whenever possible

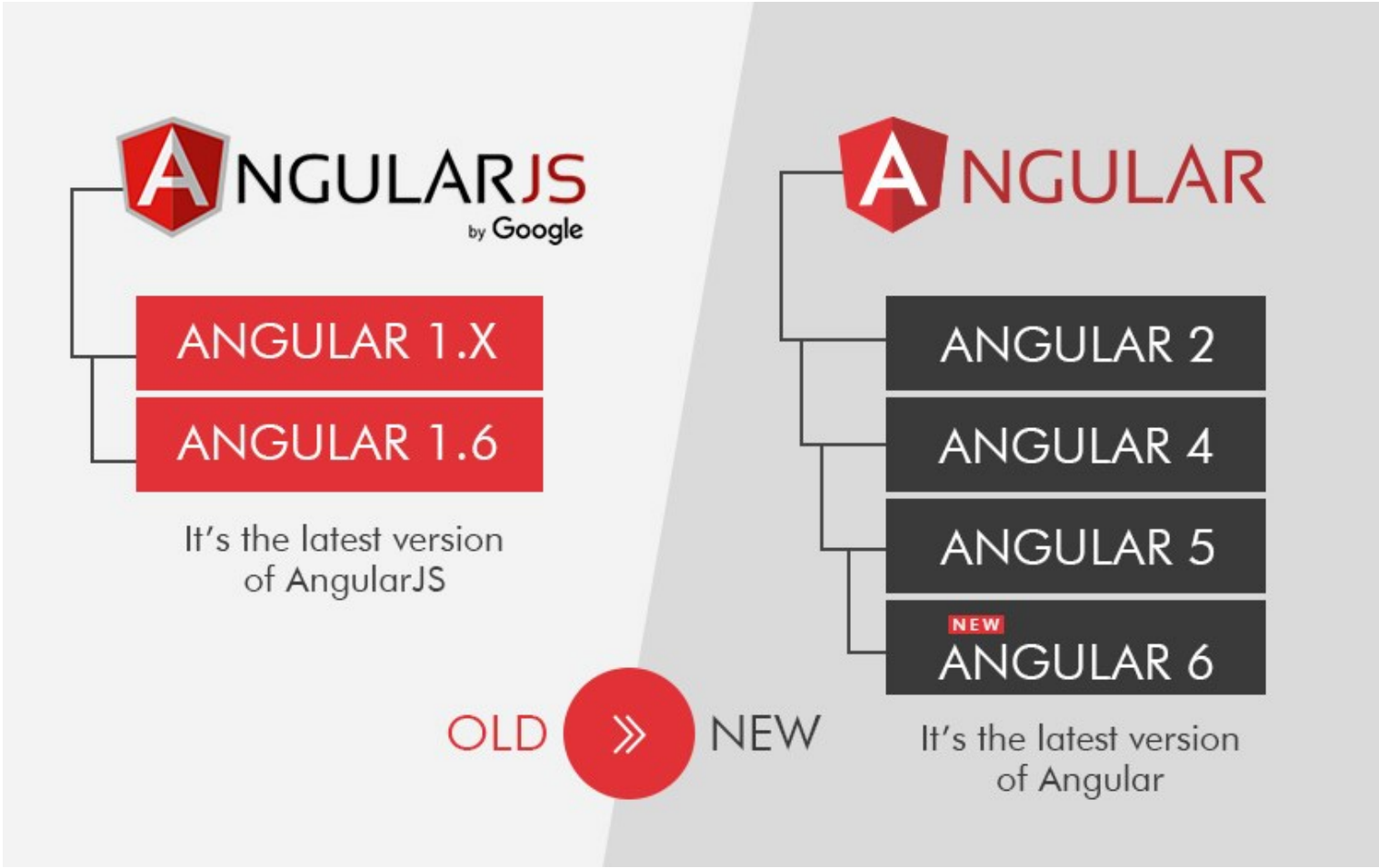
https://www.baeldung.com/spring-vs-spring-boot

Front-end programming

- AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.
- AngularJS extends HTML with **ng-directives**.
- The **ng-app** directive defines an AngularJS application.
- The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.
- The **ng-bind** directive binds application data to the HTML view.



AngularJS vs Angular



https://www.grazitti.com/blog/stuck-heres-a-quick-guide-on-how-to-upgrade-from-angularjs-to-angular/
https://www.educba.com/angular-js-vs-angular/

https://www.tutorialspoint.com/hibernate

https://www.w3schools.com/angular/angular_intro.asp

AngularJS

- The **ngController** directive attaches a controller class to the view. This is a key aspect of how angular supports the principles behind the *Model-View-Controller* design pattern.
- MVC components in angular:
  - Model
    - The Model is **scope properties**; scopes are attached to the **DOM** where **scope properties** are accessed through **bindings**.
  - View
    - The template (**HTML** with **data bindings**) that is rendered into the View
  - Controller
    - The **ngController** directive specifies a Controller class; the class contains business logic behind the application to decorate the scope with functions and values.

https://docs.angularjs.org/api/ng/directive/ngController

Example (index.html)

```
<div id="ctrl-exmpl" ng-controller="SettingsController2">
  <label>Name: <input type="text" ng-model="name"/></label>
  <button ng-click="greet()">greet</button><br/>
  Contact:
  <ul>
    <li ng-repeat="contact in contacts">
      <select ng-model="contact.type" id="select_{{$index}}">
         <option>phone</option>
         <option>email</option>
      </select>
      <input type="text" ng-model="contact.value" aria-
labelledby="select_{{$index}}" />
      <button ng-click="clearContact(contact)">clear</button>
      <button ng-click="removeContact(contact)">X</button>
    </li>
    <li>[ <button ng-click="addContact()">add</button> ]</li>
 </ul>
</div>
```

https://docs.angularjs.org/api/ng/directive/ngController

Example (app.js)

```
angular.module('controllerExample', [])
  .controller('SettingsController2', ['$scope', SettingsController2]);

function SettingsController2($scope) {
  $scope.name = 'John Smith';
  $scope.contacts = [
    {type:'phone', value:'408 555 1212'},
    {type:'email', value:'john.smith@example.org'}
  ];

  $scope.greet = function() {
    alert($scope.name);
  };

  $scope.addContact = function() {
    $scope.contacts.push({type:'email', value:'yourname@example.org'});
  };

  $scope.removeContact = function(contactToRemove) {
    var index = $scope.contacts.indexOf(contactToRemove);
    $scope.contacts.splice(index, 1);
  };

  $scope.clearContact = function(contact) {
    contact.type = 'phone';
    contact.value = '';
  };
}
```

https://docs.angularjs.org/api/ng/directive/ngController

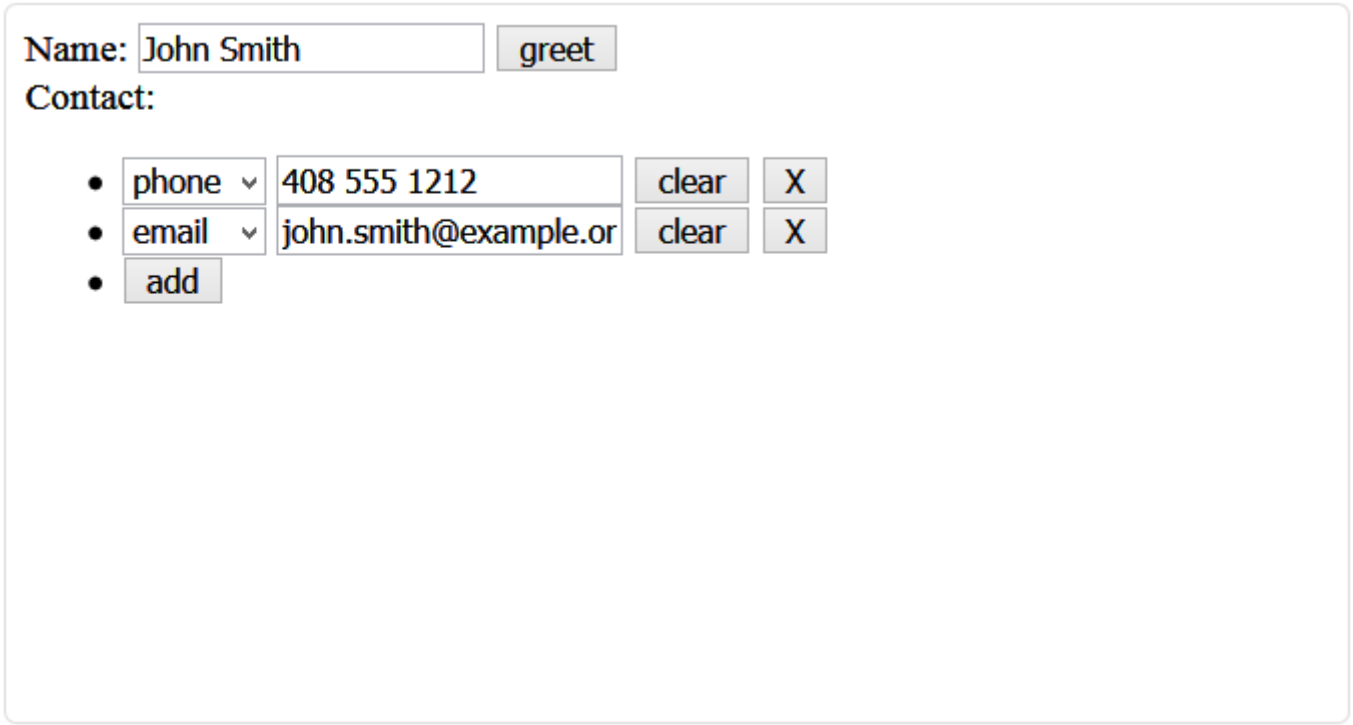
Example (protractor.js)

```
it('should check controller', function() {
  var container = element(by.id('ctrl-exmpl'));

  expect(container.element(by.model('name'))
      .getAttribute('value')).toBe('John Smith');

  var firstRepeat =
      container.element(by.repeater('contact in contacts').row(0));
  var secondRepeat =
      container.element(by.repeater('contact in contacts').row(1));

  expect(firstRepeat.element(by.model('contact.value')).getAttribute('value'))
      .toBe('408 555 1212');
  expect(secondRepeat.element(by.model('contact.value')).getAttribute('value'))
      .toBe('john.smith@example.org');

  firstRepeat.element(by.buttonText('clear')).click();

  expect(firstRepeat.element(by.model('contact.value')).getAttribute('value')).toBe('');

  container.element(by.buttonText('add')).click();

  expect(container.element(by.repeater('contact in contacts').row(2))
      .element(by.model('contact.value'))
      .getAttribute('value'))
      .toBe('yourname@example.org');
});
```

https://docs.angularjs.org/api/ng/directive/ngController

Example

- result



Protractor

- end-to-end test framework for Angular and AngularJS applications.
- runs tests against your application running in a real browser, interacting with it as a user would

https://www.protractortest.org

$http (basic usage)

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

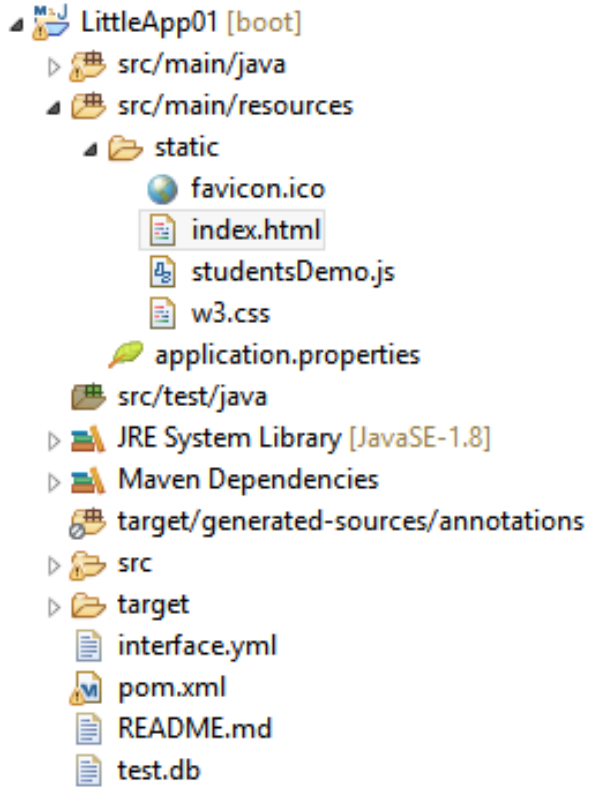
https://docs.angularjs.org/api/ng/service/$http#general-usage

$http – setting HTTP headers

```
var req = {
 method: 'POST',
 url: 'http://example.com',
 headers: {
   'Content-Type': undefined
 },
 data: { test: 'test' }
}

$http(req).then(function(){...}, function(){...});
```

Resources

- spring boot application with AngularJS



Student.java studentsDemo.js

```
1  angular.module('demo', []).controller(
2          'studentsCtrl',
3          function($scope, $http) {
4              $http.get('http://localhost:8080/littleApp/student').then(
5                      function(response) {
6                          $scope.students = response.data;
7                      });
8              $scope.name = "";
9              $scope.surname = "";
10             $scope.edit = true;
11             $scope.error = false;
12             $scope.incomplete = false;
13             $scope.hideform = true;
14             $scope.editStudent = function(id) {
15                 $scope.hideform = false;
16                 if (id == 'new') {
17                     $scope.edit = true;
18                     $scope.incomplete = true;
19                     $scope.rollNo = '';
20                     $scope.name = '';
21                     $scope.surname = '';
22                 } else {
23                     $scope.edit = true;
24                     $scope.name = $scope.students[id].name;
25                     $scope.surname = $scope.students[id].surname;
26                     $scope.rollNo = $scope.students[id].rollNo;
27                 }
28             };
```

http://tomasz.kubik.staff.iiar.pwr.wroc.pl/dydaktyka/InformationSystemsModeling/2019/LittleApp01.zip