<http://www.codingpedia.org/jhadesdev/how-does-spring-transactional-really-work/>

<https://stackoverflow.com/questions/40247030/detached-entity-passed-to-persist-in-spring-data>

<https://stackoverflow.com/questions/13036159/spring-data-override-save-method>

<http://websystique.com/springmvc/spring-4-mvc-rest-service-example-using-restcontroller/>

<https://stackoverflow.com/questions/28823600/error-creating-bean-with-name-defaultservlethandlermapping>

<https://dzone.com/articles/build-rest-web-service-using-spring>

<http://www.cyberforum.ru/java-j2ee/thread2000186.html>

<https://stackoverflow.com/questions/24710503/how-do-i-post-urlencoded-form-data-with-http>

<https://spring.io/blog/2015/06/08/cors-support-in-spring-framework>

<https://docs.spring.io/spring-security/site/docs/current/reference/html/cors.html>

https://speckyboy.com/free-bootstrap-admin-themes/

## URL-encoding variables using only AngularJS services

With AngularJS 1.4 and up, two services can handle the process of url-encoding data for POST requests, eliminating the need to manipulate the data with transformRequest or using external dependencies like jQuery:

1. [**$httpParamSerializerJQLike**](https://code.angularjs.org/1.4.1/docs/api/ng/service/$httpParamSerializerJQLike) - a serializer inspired by jQuery's [.param()](http://api.jquery.com/jquery.param/) (recommended)
2. [**$httpParamSerializer**](https://code.angularjs.org/1.4.1/docs/api/ng/service/$httpParamSerializer) - a serializer used by Angular itself for GET requests

**Example usage**

$http({

url: 'some/api/endpoint',

method: 'POST',

data: $httpParamSerializerJQLike($scope.appForm.data), // Make sure to inject the service you choose to the controller

headers: {

'Content-Type': 'application/x-www-form-urlencoded' // Note the appropriate header

}

}).success(function(response) { /\* do something here \*/ });

## [See a more verbose Plunker demo](http://plnkr.co/edit/UJRolpqzxWDBBNtlEChD?p=preview)

## How are $httpParamSerializerJQLike and $httpParamSerializerdifferent

In general, it seems $httpParamSerializer uses less "traditional" url-encoding format than $httpParamSerializerJQLike when it comes to complex data structures.

For example (ignoring percent encoding of brackets):

• **Encoding an array**

{sites:['google', 'Facebook']} // Object with array property

sites[]=google&sites[]=facebook // Result with $httpParamSerializerJQLike

sites=google&sites=facebook // Result with $httpParamSerializer

• **Encoding an object**

{address: {city: 'LA', country: 'USA'}} // Object with object property

address[city]=LA&address[country]=USA // Result with $httpParamSerializerJQLike

address={"city": "LA", country: "USA"} // Result with $httpParamSerializer

$http.post(loginUrl, "userName=" + encodeURIComponent(email) +

"&password=" + encodeURIComponent(password) +

"&grant\_type=password"

).success(function (data) {