CA 3 – Open Data with AngularJS and REST

This CA builds a technical proof of concept solution for a future (imaginary) company that has an idea for a web-application that should act as a *shared source* for all information related to all companies in Denmark. All business data used in this CA will be REAL data.

When you have completed this CA, you will have implemented a system that:

- Allow new Users to register themselves as users
- Allow registered users to request detailed information about all companies in Denmark
- Allow registered users to list daily exchange rates, historical rates and perform currency exchange calculations.

Note however, that most of the requirements for this CA are given with the focus on using all the new technologies introduced this semester, more than because they make actual business sense. For this reason we won't describe the imaginary company any further.
To make this CA more manageable, it is given as a number of individual sub-exercises.
Please note: This paper includes a lot of links, meant to be clicked while you read the description. Therefore this document is best read in an electronic version.

Learning Goals

- Increase your knowledge and expertise with REST APIs even further - your own, as well as external APIs.
- Introduce future Business Ideas for those of you with "entrepreneurial ideas", related to Open Data[1]
- Use server to server communication
- Write Single Page Applications using AngularJS
- Use New UI-controls to enhance user satisfaction
- Continue to use and expand your experience with ORM and JPA

Hand-in** and **documentation

- The code must be made available via GIThub.
- All documentation, the REST-API and the project web-pages must be made available via a web-site published on your (Digital Ocean) WEB-server.
- Documentation must include (each with a separate heading or a separate page):
  - A section explaining your test strategy, **including test results**
  - A section stating who did what. This should also include your own suggestion for the amount of Study Points each member should be awarded for this part. And a description of the strategy for assigning tasks to each group member and a documentation on how you have used git to commit your individual tasks.
- A clear and precise description of how to test the system.
  - How to test via Postman and how to use the web pages that uses the API.

Hand-in: Sunday November 6th, before 14.00.

[1] See https://en.wikipedia.org/wiki/Open_data and (in danish) http://www.opendata.dk/

How to spend your four days

Don't fall behind with any of the tasks. This is a suggested time schedule for the four days.

- Monday + Tuesday Part-1
- Wednesday **demo part 1 + 2** (see milestone-1 demo in the study point section below)
- Thursday, Friday : Part 3-6
- Weekend (Polish last parts, Angular, Documentation…)

How to complete this CA in only four days

This CA is time intensive and if you are pressed for time, it is more important that you complete a little bit of all tasks, than completing all the steps of only a few tasks.
This CA is meant to be impossible (for an average student) to complete in only four days. So you must take advantage of the fact that you are two or three persons in a team and distribute (sprint) tasks between team members.

Study Points for this (3-week) period + CA are given as sketched below:

For your participation in the class (one for each day)	9 points
For the two Study Points Exercises	12 points
Each member in a team can earn additional 19 points for the CA as sketched below:	
For your contribution to the code and documentation (verified via Git, + your documentation must include a note in a separate section, with who did what)	Max 5 points
For your milestone-1 demo	Max 4 points
The quality of your design and documentation including error handling both on frontend and backend.	Max 5 points
Quality/coverage and description/demo of your tests (see pt.5 for more information)	Max 5 points

CA-3 feedback + study point handouts
This will soon be determined.

Description of the project

The project can be done in **one** of two different ways

1. Create your own application based on your own idea – using the same technologies
2. Follow the functional requirements in the following sections of this document.

Choice 1: Go with your own idea

In this assignment you will not get any functional requirements, since those will depend on the business idea/concept. Instead you will get some non-functional requirements, that insures that you get to work with the technologies and praxises that you have learned in the past 3 modules in this semester.
To

Non-functional requirements:

1. Use JPA to for persistence
   a. Use the proper annotations for relationships
   b. Use façade classes to use with the entities
   c. Use Junit for testing
2. Use Restful web services with JAX-RS for communication between client and server
   a. Use JSON format in the communication
   b. Use Exception Mappers for error handling
   c. Use Rest Assured for testing
3. Use Angular for frontend MVC
   a. Use the Angular seed to ensure a proper architecture
   b. Use Service/Factory, modules, controllers, routes etc.
   c. Use Karma/Jasmine for testing
4. Get data from one or more other servers and store this data as historic data
   a. Screen scraping / web scraping
   b. Open data api
   c.

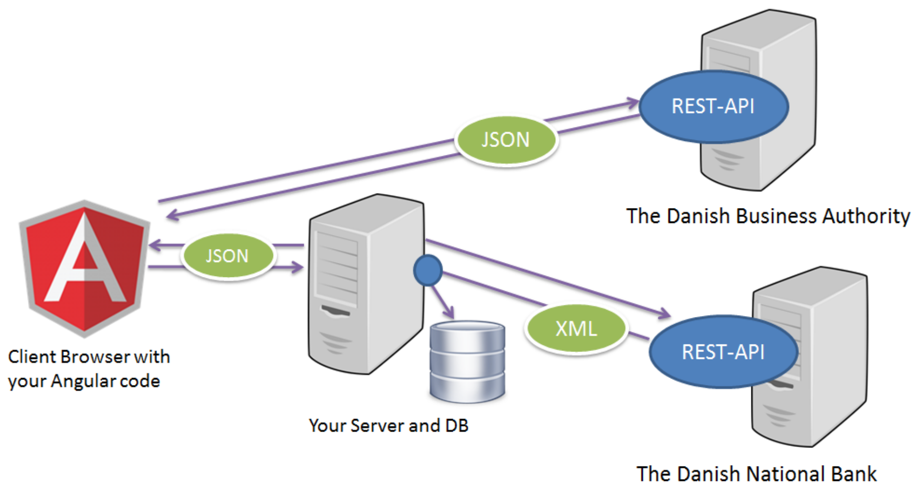
If you choose this option we would not expect a fully developed application, since you only have 4 days for this CA. Instead we would expect a well documented prototype, where some parts of the code was implemented and the rest was described in terms of E/R diagram, interfaces, rest api, test suites, user stories, website navigation diagram etc. We would then expect you to use this CA as starting point to be further elaborated and completed in the 3 week period set aside for working on the semester project (See separate file for this description) as well as in module 4 – working with system development processes and techniques.
So to sum it up: If you choose to go with your own idea you will be working with this idea for the rest of the semester.

Choice 2: A lookup service for CVR and exchange rates

In this project assignment you will create a web application that can present data from 2 different sources (CVR – Danish register for vat numbers, and exchange rates from Danish national bank). It is up to you to create a company name and a description of your company´s services in the "home" section of you application.

CA-Tasks

The final architecture/data-flow for this CA is sketched in this figure.



The following pages will introduce the individual tasks you have to do for this CA.

1-Setting up the Project Architecture

In this step you will build the basic infrastructure for the application, design the login-system and deploy a first version to Digital Ocean to verify that the system is "designed for deployment".

a) Use the semester seed as the starting point:

```
git clone -b seedDB https://github.com/Lars-m/semesterSeedSpring2016.git
```

Feel free to twist it anyway you like, especial the provided UI. It's so "boring" for us to see many "identical solutions" when you hand in ;-)

b) Change the "main page" to have (at least) five top-menu entries as sketched below:

[Home] [Documentation] [Company Info] [Exchange Rate Info] [All Users] [Login Section]

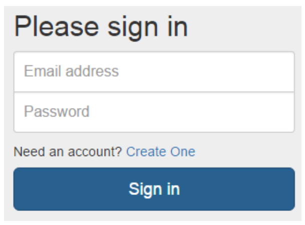
The two "green" entries must be visible and accessible, only for users logged on with the User-role and the "red" entry, only with the Admin-role.
For now, just add some hardcode text in the views for each menu-entry.

C) Create a new MySQL database for this CA and change the provided Persistence.xml to use this database (database-name, user and password) (If you delete Persistence.xml and create a new, make sure to include the persistence-unit named PU_DIGITAL OCEAN, and use the same name for the PU)

D) Delete existing users from the DB, and add two new users (See the class `DeploymentConfiguration` in package `Digital Ocean_deploy`):

- User-1: userName: **user**, password: **test**, role: **User**
- User-2: userName: **admin**, password: **test**, role: **Admin**

D) Add an option for new users to create an account.
If you use the existing UI-layout, just add another menu-entry like "*create new account*" or change the login scenario to have its "own page" like this one found on getbootstrap.com.

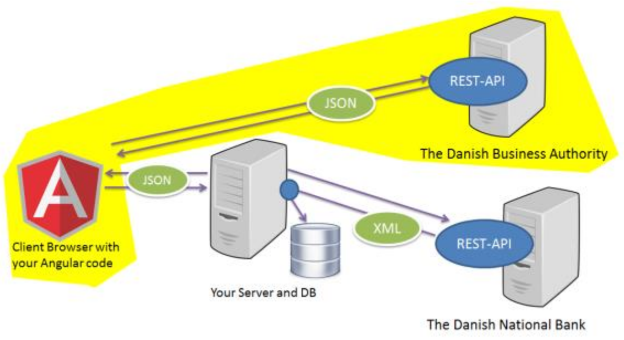


Provide new users with the role: **User**[2]

F) Upload the project to DIGITAL OCEAN.
U*pload* the project to DIGITAL OCEAN and verify that you can access the database and everything runs as expected. After that don't upload more until you are ready for the final hand-in.

2-Using Open Data, AngularJS and Cool Controls

In this step you will start to use your first "Open Data".



Now it's time to add the actual content to the Application. Everything in this task, relates to the menu-entry *Company Info* and involve only client side Code (make sure you understand how the `Same Origin Problem` is solved for this part).

Important! Read the last page and the section "Using the CVR API" before you **start doing anything** involving the links below.

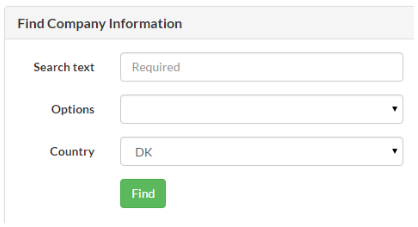
The Danish Business Authority (Erhvervsstyrelsen) has put a lot of effort into the Open Data Idea (https://data.virk.dk/what-is-virk-data). Among many things, they provide a very simple REST-API to lookup Company information given a CVR (VAT), Name or phone number.

1) First test with this example: http://cvrapi.dk/api?vat=3167%208021&country=dk

Then investigate the API further, using the documentation: http://cvrapi.dk/documentation (try with the same CVR-number as above 3167 8021, to see the JSON nicely formatted)

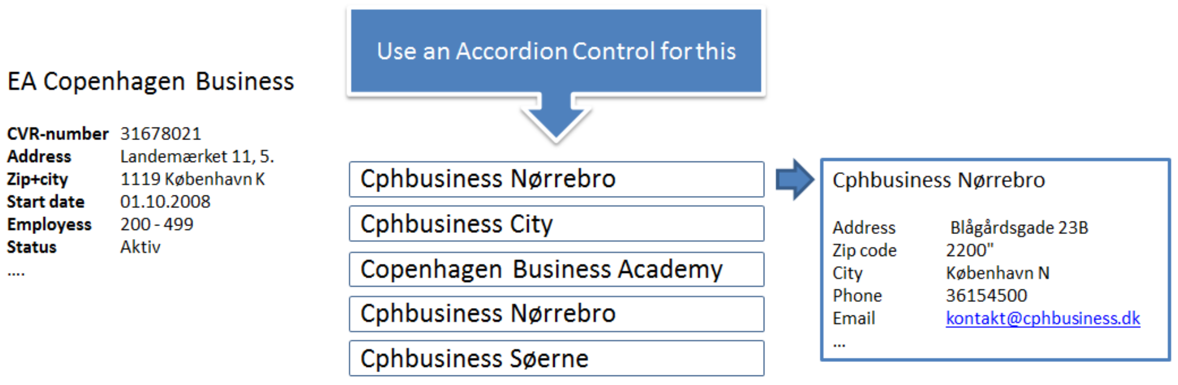
Before you continue, make sure you understand how to use the API, build queries, and how responses are returned, both for OK and ERROR responses.

2) Your task is to build a page with a search options like this (almost similar to this: http://cvrapi.dk/documentation) and present the result in a user friendly way as sketched in the mock below, exemplified with data from this response: http://cvrapi.dk/api?vat=3167%208021&country=dk


Find Company Information

Search text	Required
Options	
Country	DK

[2] *Don't consider real-life scenarios like sending the user an email to verify that he is a real person. Do that at the end if you have time.*



EA Copenhagen Business

CVR-number	31678021
Address	Landemærket 11, 5.
Zip+city	1119 København K
Start date	01.10.2008
Employess	200 - 499
Status	Aktiv

....

Use an Accordion Control for this

Cphbusiness Nørrebro

Cphbusiness City

Copenhagen Business Academy

Cphbusiness Nørrebro

Cphbusiness Søerne

Cphbusiness Nørrebro

Address	Blågårdsgade 23B
Zip code	2200"
City	København N
Phone	36154500
Email	kontakt@cphbusiness.dk

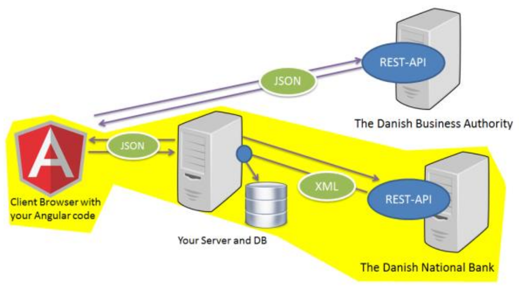
...

Use an accordion control to present the list of *productionunits* provided with the response. When you press an accordion entry, it should show the details as sketched above.
Make sure your page also presents the results for erroneous responses.
Hint: *There are many Angular Controls "out there" which provide cool controls you can use for your applications. One of them is UI Bootstrap http://angular-ui.github.io/bootstrap/, which (among many things) provides an Accordion Control.*
The seed ships with the necessary code to use UI Bootstrap controls and you can see an example-use of the accordion on the "Using the Seed" view.

3 More Open Data (this time provided as XML)

In this step you will use Open Data in a new way, since data for this task, is delivered as XML.

The Danish National Bank (Danmarks Nationalbank) provides a daily service with exchange rates given as XML (Test the link below to see the format).
Our imaginary company has got the idea, to fetch and store these data on a daily basis, so that they, in time, can provide historical data related to Currency Rates.

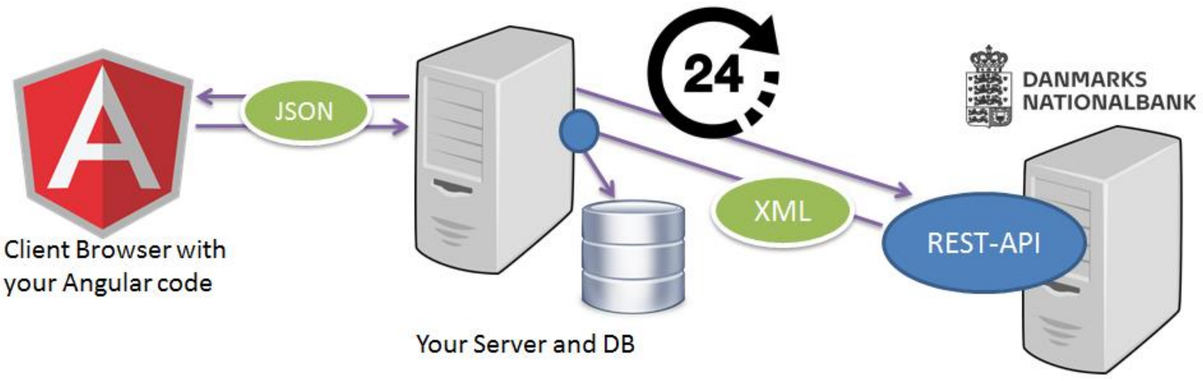


http://www.nationalbanken.dk/_vti_bin/DN/DataService.svc/CurrencyRatesXML?lang=en

Your tasks are to:

a) Create (normalized) Entity Classes to hold information about Currency Rates for each day.

b) Add a feature to let your server start a periodical task that, on a daily basis, will fetch Currency Rates and store the result in the Database as visualized below.



Hints-1 Performing Periodic Task from within Tomcat

Use this Stackoverflow article to see how you can use, start and stop, an `ExecutorService` *to execute at task periodically*

http://stackoverflow.com/questions/9173132/stop-scheduled-timer-when-shutdown-tomcat

Hints-2 Reading the Currency Exchange Rates from Danmarks Nationalbank.

There are several strategies you can use (for XML-parsing. The example below uses Javas SAX-API (streaming API) to read the remote XML-document with the Currency exchange Rates.

Run the example to see what you get, and change it to use the values obtained to update your local database

```
package xml;

import java.io.IOException;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.net.URL;

public class XmlReaderDemo extends DefaultHandler {

  @Override
  public void startDocument() throws SAXException {
    System.out.println("Start Document (Sax-event)");
  }

  @Override
  public void endDocument() throws SAXException {
    System.out.println("End Document (Sax-event)");
  }

  @Override
  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    System.out.print("Element: " + localName+": " );
   for (int i = 0; i < attributes.getLength(); i++) {
      System.out.print("[Atribute: NAME: "+ attributes.getLocalName(i) + " VALUE: " + attributes.getValue(i)+"] ");
   }
   System.out.println("");
  }


  public static void main(String[] argv) {
    try {
      XMLReader xr = XMLReaderFactory.createXMLReader();
      xr.setContentHandler(new XmlReaderDemo());
      URL url = new URL("http://www.nationalbanken.dk/_vti_bin/DN/DataService.svc/CurrencyRatesXML?lang=en");
      xr.parse(new InputSource(url.openStream()));
    } catch (SAXException | IOException e) {
      e.printStackTrace();
    }
  }
}
```

C) Add a REST service: `api/currency/dailyrates`

This service should return a JSON representation with all currency rates, similar to the XML-version you originally received. This service must be accessible only for requests identifying the caller with the USER-role.

Hints: *Since rates will only change once per day, you can performance optimize your application drasticly, by storing a cached value (do that in the "timer" service implemented in the previous step) and use this value whenever this REST service is called.*

D) Add a table using `AngularJS` and `ng-repeat` (on the Exchange Rate Info View) with information about all available rates for the current day (Inspired by the ones found here: http://www.nationalbanken.dk/da/statistik/valutakurs/Sider/Default.aspx).

E) Add a new REST service: `api/currency/calculator/:amount/:fromcurrency/:tocurrency`

This service should return the amount you get in a given currency (*toCurrency-argument*) for an amount given in the *amount-argument,* in the currency given in the *fromcurrency-argument*.

As for the previous service, this must be accessible only for requests identifying the caller as a USER.

F) Create a Calculator view using AngularJS to provide users with a calculator service. Come up with your own design or get inspired by some of many calculators "out there" as for example: http://valutaomregneren.dk/

Challenge: *If you are using (as you are in this exercise) your calculator in the same application as the one that shows all currency rates, you already have the rates on the client and could do the calculation solely client-side. Come up with a smart service that fetches the data used for 3b), caches the data, and provides a calculator service, similar to the server implementation implemented in 4a). Change your Calculator UI/Controller, to use this service*

4 The Admin (All Users) Page

1) Add a new GET-REST service: `api/admin/users`
This should provide a JSON list with information about all users in the system.
This service may only be called by requests that identify itself with the Admin-role.

2) Add a new DELETE-REST service: `api/admin/user/:id`
This should delete the user with the given id.
This service may only be called by requests that identify itself with the Admin-role.

3) Create the necessary UI-code to *show all users* and *delete a user* on the "All Users" page.

5 Testing the Application

For API-testing we expect you to use Postman.
As a minimum your API-test should include a test, verifying all four CRUD operations. This must be demonstrated during the project presentation

API-testing:
Your endpoints needs to have at least one happy-path[3] test, and one test that in one way or another test a fail scenario by either:

- Try to give malformed input
- Try to access content that is restricted
- Try to post malformed input

Backend testing:
Test all methods in your façade with both happy-path and exception triggering unit tests.
Remember to have a separate persistence unit for the tests:
If you forgot how to operate with 2 PU's then look at this project:
https://github.com/Lars-m/RestDemoPerson

6 Complete the Documentation Page(s) and upload all to OPENSHFT

The project must be available online as a running application including a page with all documentation.

3 https://en.wikipedia.org/wiki/Happy_path

7 Using the CVR API

First time we started using this API, it ended with this response when request was made from within the school after a few minutes:

```
{
  "error": "QUOTA_EXCEEDED",
  "message": "Your quota has been exceeded. Reach out if you are certain this is a error."
}
```

It seems they count requests made from a given IP (remember they see the public IP, so you will all be seen as "one"), and only allows a certain amount (I haven't found this documented anywhere) of requests. This makes sense for an open API that does not require any kind of authentication in order to prevent DOS-attacks.

To overcome this problem you should do the following.

1)
For all requests you make, set the `User-Agent` request header to a value identifying yourself, as for example "`CVR API-CA3 SCHOOL Exercise-YOUR NAME-YOUREMAIL@cphbusiness.dk`"

From within Postman, just select *headers* and "User-Agent" and provide a value as sketched above.

With Angular and `$http`, use the info here to set a specific request-header:
https://docs.angularjs.org/api/ng/service/$http/#setting-http-headers

I am not sure whether this will raise your/our quota, but it is the required way of making requests, see: http://cvrapi.dk/documentation (in Danish)

2)
Since CVRAPI count requests from a given IP (not your internal 10.xxx IP, but the schools public IP) you can (and should) set up your own WIFI Access point, using your phone to get another IP.

If you never done this before, google "*mobile wifi access*" or "*mobile hot spot*" + your phone type (*android or IPhone*) for details of how to do this.

Using the CVRAPI with the semester seed

The default behaviour for the client-side of the seed is, to include an authentication-header with the jwt-token on all outgoing requests. This is by design, to simplify calls to the REST-API. This however, will also be true for requests made to http://cvrapi.dk. Since this API does not require authentication, it will not accept requests made with an authentication header.
You can make a request without the authentication header as sketched below:

```
$http({
  url: ' http://cvrapi.dk/....,
  skipAuthorization: true
  method: 'GET'
});
```