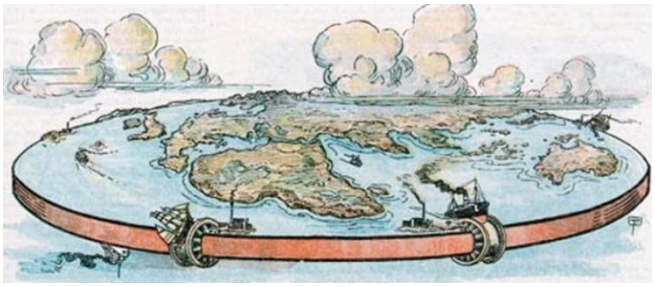
The Semester Project and Date Handling

Once upon a time (very) long ago the world was considered flat and there were no countries with silly ideas like placing the month before the day or similar. At that time, the sunrise would happen at the same "time" all over the world, and the same for the sunset. Then unfortunately, over a period, starting probably way back with Pythagoras the misconception that our planet is round started to spread and individual countries started to invent their own time-formats.



Ever since, life as a programmer has been so much harder :-)

Imagine the following date: 02/04/03. Which does it mean?

- 2nd of April 2003 (European style)
- 4th of February 2003 (USA style)
- 3rd of April 2002

Your answer will depend, mostly, on which country you live in.

Also, imagine you were asked to save a timestamp for an operation triggered in a web-environment. Which time should you save? The user's local time (perhaps in Denmark) or the server time (perhaps located in the US) and no matter which you save, how would you save it, so we still had the information about the "origin" of the time stamp?

Date/Time-assumptions in the Semester Project

To abstract away, most of such date/time-related problems, which a real-life Momondo system obviously will face, we have made several assumptions:

- All airports must be given as a IATA-code + time zone (The only place we store time zone information)
- TIMEs are all given as start times in local time, no end time is given but flight time must be given, so end time can be calculated (given the time zone included for all airports).
- Times are formatted as ISO 8601 (http://en.wikipedia.org/wiki/ISO_8601) but we disregard the time-zone indicator of the String (assuming local time, and that we know the location)

We selected the ISO 8601 format because it's the de facto standard for dates on the WEB and because it's very easy to handle from JavaScript (and Java).

In the following I will describe the ISO Date Format, followed by a number of hints related to:

- Converting *JavaScript* Date Objects to JSON encoded ISO 8601 strings
- Converting JSON with ISO 8601 Stings into *JavaScript*
- Creating *Java* Date instances from a ISO-8601 string
- Java Date Objects To and From JSON encoded ISO 8601 strings using Gson
- Handling Java Date Objects to DB-dates and back from DB-dates to Java Date Objects
- How we will cheat with the UTC-time
- Getting dates from Date Pickers.

Lars Mortensen, Spring 2016

The ISO 8601 Date Format

This format was chosen because it's the recommended format on the web (http://www.w3.org/QA/Tips/iso-date) and very easy to use from almost all programming languages.

This format has a number of advantages:

- it's completely unambiguous (no matter what country/time-zone you live in)
- It's human readable
- It sorts correctly
- The fractional seconds make it ideal for server timestamps (to establish chronology)

To see an example, open Chrome Developer Tools and the Console tab and type in the following:

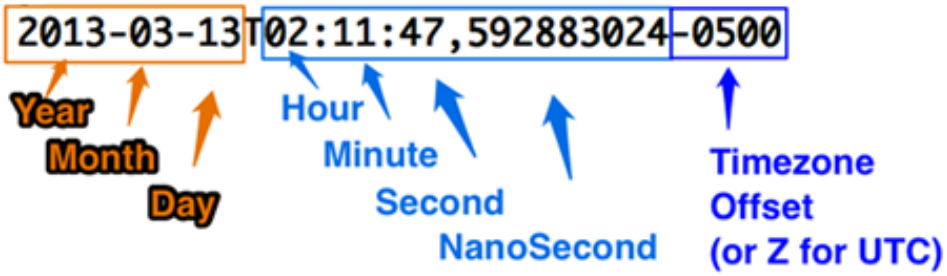
`new Date().toISOString();` Result→ *2015-11-25T08:40:26.013Z* (obviously with another value)

And

`JSON.stringify(new Date());` Result→ *2015-11-25T08:40:26.013Z*

Observe that the time is adjusted to UTC (see: https://en.wikipedia.org/wiki/Coordinated_Universal_Time) which is indicated by the final "Z" in the ISO-8601 string.

The individual parts of the full (date + time) ISO-8601 string is sketched in this figure.



Ref: http://apiux.com/2013/03/20/5-laws-api-dates-and-times/

An alternative to this format is the Unix-timestamp. JavaScript dates are internally represented with such a value, a time value that is milliseconds since midnight 01 January, 1970 UTC.

You can get this value using JavaScripts `getTime()` method as sketched below:

`new Date().getTime();` Result → *1448443201595* (obviously with another value. Put this value into a Date Constructor to see when I did this).

The obvious drawback of this format is its lack of readability and that it cant handle dates before 1970. This is how the JavaScript Date Object internally stores the date values.

Converting Java Date Objects to JSON encoded ISO 8601 strings

As you saw in the previous section, JavaScripts `JSON.stringify(..)` method will automatically convert dates into a ISO-8601 compliant string as demonstrated below.

JavaScript	JSON
`var date = new Date(2015,11,24);` `JSON.stringify(date);`	`"2015-12-23T23:00:00.000Z"`

`var event  =  {};`
`event.name = "Christmas";`
`event.date = new Date(2015,11,24);`
`JSON.stringify(event);`
	`{`
	`  "name":"Christmas",`
	`  "date":"2015-12-23T23:00:00.000Z"`
	`}`

Converting JSON with ISO 8601 Stings into *JavaScript*

Converting from JSON into a JavaScript object takes a little bit more effort.

The third line in the example below, will not, as you might expect, hold a date but just the ISO-8601 string.

```
var date = new Date(2015,11,24);
var jsonDate = JSON.stringify(date); //Convert into a JSON-string
var dateStr = JSON.parse(jsonDate);  //Convert back into JavaScript
```

Since JSON does not have a default date-type, it cannot know that the string actually represents a date. Fortunately JavaScript provides a Date Constructor that takes an ISO-8601 string, so adding this line to the example above will provide a new Date Object.

```
var dateFromJson = new Date(dateStr);
```

Similar with the second example from the previous section:

```
1. var event  =  {};
2. event.name = "Christmas";
3. event.date = new Date(2015,11,24);
4. var json = JSON.stringify(event);
5. var newEvent = JSON.parse(json);
6. newEvent.date = new Date(eventFromJson.date);
```

In line 5 the date property of the newEvent object will not be a date, but the ISO-8601 representation. To get a Date instance you need to use the Date Constructor that takes an ISO-860 string (line-6)

Test the example in the Console to see this for real.

Creating *Java* Date instances from a ISO-8601 string

All the "simple" `java.util.Date` constructors has been (for good reasons) deprecated, so when creating a Date object, not representing the current data and time, you will almost always use the `SimpleDateFormat` class. The examples below shows how to create Java Date Objects, given almost any format (the second example is using an ISO-8601 string)

```
SimpleDateFormat sdf1 = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
Date date1 = sdf1.parse("01-01-2016 06:00:00");
System.out.println(date1);

DateFormat sdfISO = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");
Date date2 = sdfISO.parse("2001-07-04T12:08:56.235-0700");
System.out.println(date2);
```

Java Date Objects To and From JSON encoded ISO 8601 strings

When using Googles *gson* library you should create your instance like this

```
Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'").
setPrettyPrinting().create();
```

With a *gson* instance created as above, you could take a class like this:

```
public class Event {
  String name;
  Date date;
  ... //Constructor + getters and setters
}
```

And convert into JSON, and back again, as sketched below:

```
Event ev =
  new Event("Christmas",new SimpleDateFormat("dd-M-yyyy").parse("24-12-2016"));
Sting jsonStr = gson.toJson(ev);
System.out.println(jsonStr);
Event ev2 = gson.fromJson(jsonStr,Event.class);
System.out.println(ev2.getName() +", "+ev2.getDate());
```

Java Date Objects to DB-dates, and back from DB-dates to Java Date Objects

Last semester you probably struggled a bit with conversions between plain Java dates (java.util.Date) and SQL-dates (java.sql.Date). By now, you should have noticed that JPA handles this for us, using one of the three possible values of `TemporalType` with the `@Temporal` annotation.

- `@Temporal(javax.persistence.TemporalType.TIMESTAMP)`
- `@Temporal(javax.persistence.TemporalType.DATE)`
- `@Temporal(javax.persistence.TemporalType.TIME)`

How we will cheat with the UTC-time

Imagine a request for a ticket from Berlin (SXF) to Copenhagen (CPH) January 6th 2016.

```
api/flightinfo/SXF/CPH/2016-01-06T00:00:00.000Z/2
```

Here we interpret only the date part of the ISO-string and disregard the time and most important the final Z (indicating UTC-time). It would probably have been better, if the protocol had only required the date part, but that's too late to change.

The following is an example response for the request given above. Here we use both the date + time parts from the ISO-string to get the flight date + time January 6th 2016 08.00. Again we disregard the Z (indicating UTC) and interpret the time as 08.00 Berlin time.

```
{
  "airline": "AngularJS Airline",
  "flights": [
    {
      "flightId": "COL2215",
      "numberOfSeats": 2,
      "date": "2016-01-06T08:00:00.000Z",
      "priceTotal": 170,
      .. .
```

Date Pickers and how to disregard the time zone

If you create a basic HTML5 date picker and bind it like this (assuming you are using Angular):

```
<input ng-model="myDate" type="date" class="form-control">
```

This is the value you will get, if you select December 24th 2015 and convert it to ISO-8601:

2015-12-**23**T23:00:00.000Z

If you have kids they will be so disappointed. Just when they thought it was Christmas, it turns out that it is only the 23th ;-)

Fortunately enough, you can explain to your kids that they don't have to worry. It's UTC-time (again indicated by the Z) and since this was written in Denmark (GMT +1) it is actually Christmas :-)

If we want to disregard the locale completely as explained in the previous section we transform the date like this:

```
var year = $scope.myDate.getFullYear();;
var month = $scope.myDate.getMonth();
var day = $scope.myDate.getDate();
$scope.date = new Date(year,month,day,1);
```

Here we create a new date from the original + 1 hour (GMT +1). Now if your get the ISO-string it will print: 2015-11-24T00:00:00.000Z

Note: We only do this because we have decided to disregard the "Z", so it is not meant as a hint for real life scenarios.

References:

ISO 8601: https://en.wikipedia.org/wiki/ISO_8601

UTC: https://en.wikipedia.org/wiki/Coordinated_Universal_Time

JavaScript Date Object: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

API's and Date/Time: http://apiux.com/2013/03/20/5-laws-api-dates-and-times/