Case Study:

The Web 2.0 Front for Denmark's National Healthcare Solution.

"4 stories from RIA newbies"

Christian Hvitved, Developer at Trifork

About This Talk

- Focus on the problems we encountered, and what we have learned creating a rich internet application – because the Programme Committee of JAOO heard about our problems

The Common Medicine Card (CMC)

- Project for the Danish Medicine Agency (The government)
- Central repository containing medicine cards for all danish citizens
- A medicine card shows the actual medication for a person
  Everyone in the healthcare sector must use cmc to see or change a persons medica
- Today this information is not shar
  healthcare sector

A project that you really think wi
improve the healthcare sector

TRIFORK.

CMC Overview

Citizen Browser

Doctor Browser

Hospital Systems (EHR)

Home Care Systems

General Practitioner Systems

Systems in the healthcare sector

Closed Healthcare network (Sundhedsdatanettet)

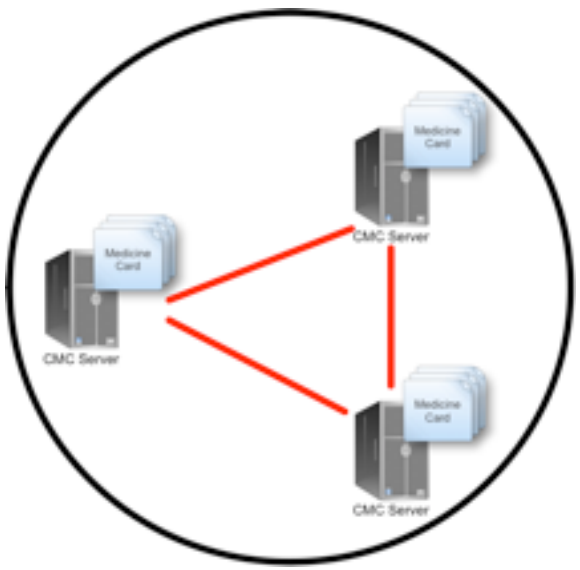
JSON Services

SOAP Web Services

Business Layer

Data Access Layer

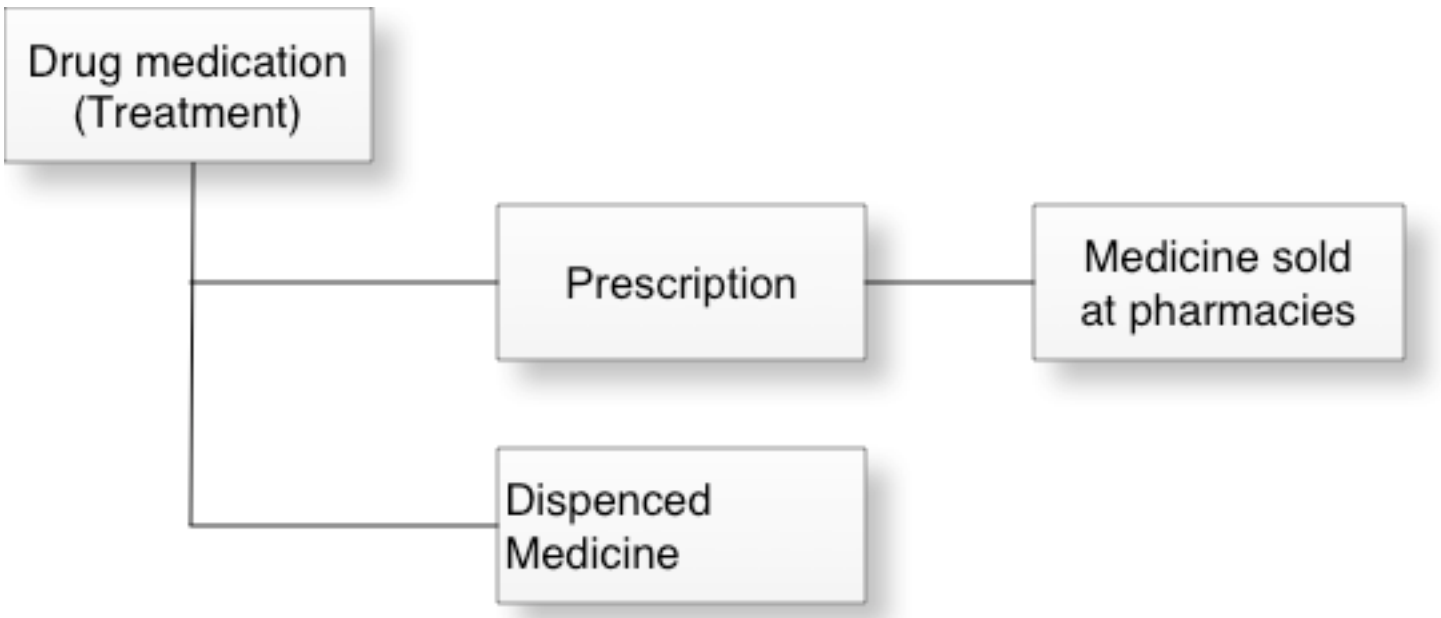
Relational Database



Distributed model
(Peer-to-Peer network)

Demo

Demo Time



Setting the Stage

- This presentation will tell our stories of creating a rich web application.
  - We had experience with traditional web development.
  - This was the teams first rich web app
  - We have, and are still learning a lot

1st Story

Choosing Programming Language

Choosing Programming Language

- Flex (Silverlight) was not an option (Political decision)
  - Requires Flashplayer
- Started out using javascript
  - Javascript is getting a lot of attention
  - The common programming language for browsers
  - Chose the right tool for the job
    - Polyglot programming / pragmatic programmers
  - Javascript is not regarded as a toy language anymore Many regard it as a decent programming language with "good parts and bad parts"
- 2 Developers
  - Some knowlegde about javascript
  - No real javascript experience

First Sprint – Outcome

- Javascript problems (Typical problems when moving from statically typed to dynamically typed languages)
  - tooling: debuggers, IDE support
  - Hard time structuring javascript code
  - Learn a new framework in the new language
    - Harder to explore new frameworks in a dynamically typed language
- Abandoned Javascript for Google Web Toolkit (GWT)
  - We had experience with GWT

Google Web Toolkit

- Pros
  - Use Java Tooling
  - Easy to structure and navigate code
  - Code is written tested and debugged in java
    - It actually works when compiled to javascript
  - GWT uses statically typing for optimizations
    - Only emit code that is actually used
    - Create different Permutations for each browser

Google Web Toolkit

- Cons
  - Java to javascript layer is introduced – a layer of indirection to the runtime environment
  - Java is not a very elegant language (anonymous inner classes all over)
    - GWT compiles java source files – so other JVM languages cannot be used
  - Hosted mode browser is platform specific
    - Cannot use firebug in hosted mode
    - gwt compile is slow -> long turnaround when not testing in hosted mode browser
    - Solved in GWT 2.0

GWT < > Javascript

- We chose GWT
  - and we don't regret it
  - We can leverage our java experience
- Is javascript the programming language or the assembly language (jvm) for the browser?
  - GWT
  - Microsoft's project formerly knows as Volta

2nd Story

Choosing Framework

Nice GUI Component Model

- We chose to use Ext JS (http://www.extjs.com)
  - Widely used javascript framework
  - Has some good looking widgets
  - Build the UI by combining these widgets.
  - Swing like model with layout-managers etc
  - Programmers work with high level composable widgets (not html)

- We kept using Ext with GWT (Ext GWT)
  (http://www.extjs.com)

But alas.. it did not work...

- Hard to customize the widgets to our needs
  - Had to look at the component internals
  - Wrestle the code and do a lot of low level html
- Things broke as we combined widgets in different ways.
- Browser differences exploded

- Ext GWT did not fit our needs
  - We tried to work around it for too long!

Why did it not work for us?

- The component model did not work
  - We were wrestling with the HTML/CSS most of the time
- Complex HTML
  - Difficult to understand and debug
  - Browser differences gets worse as html gets complex and nested
    - A component worked in one context but not in another

EXT Example

- EXT example
- CMC grid example

Clean HTML + CSS

- Took the opposite approach
- Start defining the HTML for a component
- Create semantically correct HTML
  - Only use tables if you want to display tabular data
  - Do not nest everything inside divs. etc
  - No layout decisions
- HTML creates the structure and contains the data
- Design is pushed to CSS

Changed the way we work

- We do not use Ext GWT's or GWT's high level widgets
- We work at a lower level
  - Components are HTML centric
  - Often using the DOM API
- We have created many things from scratch

Clean html is as important as clean code

Know your Platform

HTML, DOM, CSS, Javascript (GWT)

- We did not master these technologies
  - In the old days “real” programmers did not care much for the browser technologies – you could always make it work
  - “Real men code server side” - not anymore, the client is the hard part!
- We teamed up with a designer (html/css wizard)

Do not try to abstract away the html - work with it

3rd Story

HTML Templates

Templates

- Focusing on html we created html templates
  - Plain html files which we load from resource bundles
  - Possible to insert “active”elements into the template

Template Example - HTML

```
<dl class="create-effectuation">
     <dt>Effektueringstidspunkt:</dt>
     <dd><input id="{datetime}" type="text" class="mkt-text-input" /></dd>

     <dt>V&aelig;lg l&aelig;gemiddel:</dt>
     <dd>
          <select id="{drug-combo}" name="drug" class="mkt-form-field"/>
     </dd>

     ...

</dl>
<div class="buttons">
     <button id="{button1}">Opret</button>
     <button id="{button2}">Fortryd</button>
</div>
```

Show html in application

form layout is not done with a table
    Label - input

dl = definition list
dt = definition term
dd = definition description

can look at static html page
    select has options

Template Example - Java

```
public class CreateEffectuationPanel extends Composite {
  private DateTimeField effectuationDateTime = new DateTimeField();
  private DrugComboBox drugComboBox = new DrugComboBox();

  ...

  private void setupHTML() { //called from the constructor
     String html = HtmlResources.INSTANCE.createEffectuation().getText();
     TemplateHTMLPanel panel = new TemplateHTMLPanel(html);

     panel.addAndReplace(effectuationDateTime, "datetime");
     panel.addAndReplace(drugComboBox, "drug-combo");
     ...
     initWidget(panel);
  }

  ...
}
```

Use the active components in the code.

Add click listeners, keyboardlisteners etc

Templates - Conclusion

- Useful when working with HTML
  - What JSP's are to Servlets
  - Alternative to using the DOM API or string manipulation in a java class
  - View the static html page in a browser
- A widget is typically layed out using a template
- Could probably be improved to be "typesafe" using GWT generators
  - ~~`panel.addAndReplace(drugComboBox, "drug-combo");`~~
  - `panel.setDrugComboBox(drugComboBox);`
- UI Binder in GWT 2.0

4th Story

Java to JSON Binding

JSON Services

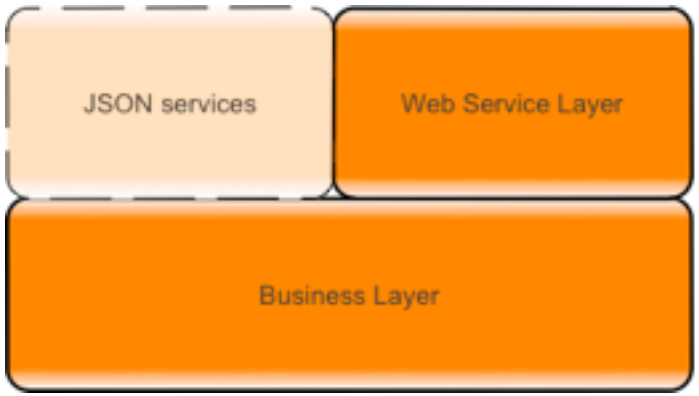
- SOAP-XML-WS* services existed
- Make same services available as HTTP/JSON services:

Expose same services as JSON

Server side support
emit and recieve json

RestFul

But was pretty easy to add



JSON at the Client

- GWT has a JSON library

```
 {"PriceListVersionDate":"2009-05-06",
  "DrugStructure":[
     {"DrugIdentifier":28100559669, "DrugName":"Ventoline", ... },
     {”DrugIdentifier":28101785695,"DrugName":"Ventoline", ... },
    ...
  ]
 }
```

```
private void parseJSON(String text) {
      JSONObject json = JSONParser.parse(text).isObject();
      String pricelistVersionDate = json.get("PriceListVersionDate").isString().stringValue();
      System.out.println(pricelistVersionDate);
      JSONArray drugs = json.get("DrugStructure").isArray();
      for (int i = 0; i < drugs.size(); i++) {
            JSONObject drug = drugs.get(i).isObject();
            String name = drug.get("DrugName").isString().stringValue();
            double id = drug.get("DrugIdentifier").isNumber().doubleValue();
            System.out.println(name + " (" + id + ")");
      }
}
```

JSON objects do not show which data they contain

JSON to Java Binding on the Client

Twist is it has to run i GWT

1. XSD -> Java

JAXB →

JAXB annotated java classes

---

2. Use JAXB classes in the GWT client

→

GWT Runtime enviroment

---

3. Auto generate JSON serializers/deserializers for JAXB classes

GWT generator

JAXB annotations →

JSON serializer / deserializer java classes

---

4. Create an API for calling JSON services

JSON Services - Conclusion

- Java to JSON binding
  - java classes representing the data that can be send and received
  - We stay in our statically typed world
- Could be a nice open source project
- All this is necessary because we use Java (and not javascript)
- The web application uses exactly the same services as other systems using CMC
  - Added HTTP/JSON interface to CMC

That was our team's war stories...

The most important things we have learned

And how we have done things

Hope this can help others not making the same mistakes

What worked for us

- Focus on the HTML.
  - Make semantically correct and clean HTML
- Master the browser technologies
  - Know the DOM to manipulate HTML
  - Design using CSS
- We found it much easier to structure and debug Java code (than javascript code)
  - We created templates making it easy to work with HTML
  - Created a Java to JSON binding library

Questions?