Riku-Hermanni Kuusisto

TRANSITION FROM JAVA APPLET TO MODERN WEB APPLICATION

Degree Programme in Information Technology

2018

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TRANSITION FROM JAVA APPLET TO MODERN WEB APPLICATION

Kuusisto, Riku-Hermanni
Satakunta University of Applied Sciences
Degree Programme in Information Technology
September 2018
Supervisor: Trast, Ismo
Number of pages: 31
Appendices:
IBM Content 2017 22 giugno Roma Strategy and Roadmap Fery Clayton.pdf
WebSphere.pdf

Keywords: modern design patterns, web application, MVC model

The purpose of this thesis was to demonstrate the need for replacement of a Java-based software and introduce the MVC model implementing replacement web application. In this thesis I examine modern design patterns and present alternative techniques while introducing my own knowledge and findings that are not present in the source literature.

The material consists of my findings of the software and systems that are being used. There is no separate summary of the findings, but they are a natural part of this thesis. I research the behavior and the need for improvement based on the source literature and my own experience over the matter.

As a product of the research I developed a web application that can be deployed as a standalone app or integrated in IBM Content Navigator or IBM Case Manager plugin. I implemented the user interface using Dojo Toolkit. For back-end, I implemented a Java servlet that runs on a Java application server such as IBM WebSphere, Apache Tomcat or Oracle Glassfish. The Java servlet make use of Spring and JAXB technologies.

During my research, it occurred to me that there is a wide range of available technologies that can be used to make corresponding solutions. Seldom there is only one way or technology that will bring the desired results. While making the architectural decision of the used technologies, it is wise to compare many technologies and choose the one which is efficient enough and compatible with the environment that the solution is going to be deployed to.

JAVA-SOVELLUKSEN KORVAAMINEN MODERNILLA WEB-SOVELLUKSELLA

Kuusisto, Riku-Hermanni
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
syyskuu 2018
Sivumäärä: 31
Liitteet:
IBM Content 2017 22 giugno Roma Strategy and Roadmap Fery Clayton.pdf
WebSphere.pdf

Asiasanat: modern design patterns, web application, MVC model

---

Tiivistelmä:

Opinnäytteen tarkoitus oli osoittaa työn kohteena olleen sovelluksen korvaustarve ja esitellä korvaavan MVC-mallin toteuttavan web-pohjaisen sovelluksen pääpiirteet verrattuna alkuperäiseen. Työssäni tutkin moderneja suunnittelumalleja ja esittelen vaihtoehtoisia tekniikoita tuoden esiin myös omia tietoja ja havaintoja, jotka eivät perustu saatavilla olevaan lähdemateriaaliin.

Työni aineisto koostuu havainnoista, jotka olen tehnyt olemassa olevista sovelluksista. Lähdemateriaalista ei ole varsinaista yhteenvetoa, vaan se on aseteltu osaksi tätä työtä. Tutkin sovellusten toimintaa ja kehittämistarpeita keräämäni ammatillisen kokemuksen sekä verkkolähteiden pohjalta.

Opinnäytteeni tuloksena syntyi olemassa olevalle Java-sovellukselle korvaava web-pohjainen toteutus, jonka voi liittää osaksi IBM Content Navigator -tuotetta tai IBM Case Manager -liittännäistä. Toteutin sovelluksen käyttöliittymän Dojo Toolkit -kirjastokokoelmaa käyttäen. Taustajärjestelmä on Java-sovelluspalvelimella kuten IBM WebSphere, Apache Tomcat tai Oracle Glassfish, ajettava sovellus. Sovellus hyödyntää mm. Spring ja JAXB -teknologioita.

Opinnäytettä työstäessäni ymmärsin, että tarjolla on laaja kirjo teknologioita, joilla voi toteuttaa toisiaan vastaavia ratkaisuja. Vain harvoihin ratkaisuihin on olemassa yksi oikea tapa tai teknologia – käytettävää teknologiaa kannattaa tarkastella mahdollisimman monelta kantilta, ja etsiä teknologia, joka on riittävän tehokas ja yhteensopiva kohdeympäristön kanssa.

CONTENTS

1 INTRODUCTION 5
2 MODERN WEB APPLICATIONS THEORY 6
2.1 Concepts 6
2.1.1 User Interface 6
2.1.2 Extensible Markup Language 6
2.1.3 Hyper Text Markup Language 7
2.1.4 Cascading Style Sheet 7
2.1.5 Application Programming Interface 9
2.1.6 JavaScript programming language 9
2.1.7 Representational State Transfer 9
2.1.8 JavaScript Object Notation 9
2.1.9 Case Management 10
2.2 Common Design Patterns 11
2.2.1 Single purpose principle 11
2.2.2 Model-View-Controller 11
2.2.3 Dependency Injection 12
2.3 JavaScript Frameworks 12
2.3.1 Dojo Toolkit 12
2.3.2 JSX 12
2.3.3 ReactJS 12
2.4 Java Technologies 13
2.4.1 Spring Web MVC 13
2.4.2 Java Generics 13
2.5 Data Warehousing 15
2.5.1 FileNet P8 15
2.5.2 P8 Object Store 15
3 IMPLEMENTING INTERACTIVE WEB APPLICATIONS USING DOJO TOOLKIT 16
3.1 IBM Case Manager 16
3.2 Dojo 17
3.3 Spring 18
3.4 The Case Study 19
3.4.1 Premise 19
3.4.2 Architectural specification 20
3.4.3 User Experience – Queue configuration 21

3.4.4 RESTful service endpoints 24
3.4.5 RESTful service 25
4 CONCLUSION 28
REFERENCES 29

1 INTRODUCTION

This thesis is conducted for Elinar Oy Ltd, a Finnish system integrator and software company focused on electronic content management and artificial intelligence using IBM software exclusively.

The purpose of this thesis to integrate a RESTful service with IBM Case Manager using custom widget developed using Dojo Toolkit. This is a common requirement when the user interface of Case Manager must be extended with complex UI components that have a controller running in a back-end application. I have picked an example application from among the projects that I have been working on for Elinar. The application consists of a metadata input form, a restful service and external configuration stored in an object store.

At the time of writing, Dojo Toolkit is used for UI development for Case Manager and Content Navigator software. According to an IBM Content Navigator Roadmap, support for React framework will be added to Content Navigator in the late 2017. (Feri Clayton, 2017)

As Dojo plays the dominative role in all UI components for Content Navigator and Case Manager, it would be a tremendous act to completely replace the toolkit with React. According to discussion on IBM developer forums, support for Dojo and React will co-exist for the time being. (Manjum, 2017)

2 MODERN WEB APPLICATIONS THEORY

2.1 Concepts

2.1.1 User Interface

abbr. UI

User interface is a product of design that predicts what human, the user, might have to do. User interface is what the user sees the service or application represents.

2.1.2 Extensible Markup Language

abbr. XML

XML is a document definition standard that allows for structured representation of information. The information is contained within XML tags that are keywords surrounded by angle brackets. Tags can describe the meaning of the contained information.

2.1.3 Hyper Text Markup Language

abbr. HTML

HTML is an XML based, structured document definition language which is used for creating web-based user interface layouts. The language consists of standardized elements called HTML tags. The tags can contain text or other tags. Attributes, like *class=”logo”* are included to distinguish tags with the same name from each other.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Simple page</title>
<link href="css/singlePageTemplate.css" rel="stylesheet" type="text/css">
<script src="js/script.js" type="text/javascript"></script>
</head>
<body>
<div class="container">
  <header>
    <h4 class="logo">HELLO WORLD</h4>
  </header>
  <section class="hero" id="hero" onClick="sayHello(this);"
   data-value="world">
    <h2 class="hero_header">HELLO <span class="light">WORLD</span></h2>
    <p class="tagline">A simple test website</p>
  </section>
</div>
</body>
</html>
```

Figure 1 Sample HTML document with a header and a content section

2.1.4 Cascading Style Sheet

abbr. CSS

A class-oriented description of visual styles which is natively used by HTML renderers such as web browsers. CSS is used to add styling to text or images displayed on a web page or to render shapes and transitions for responsive web design. Among the other new features, version 3 brings keyframe animations to CSS enabling more responsive and versatile web applications.

```css
.spinner{
    position: relative;
    left: 0;
    top: 97%;
    height: 6%;
    width: 100%;
}
.spinner .dot{
    position: relative;
    display: block;
    left: 0;
    top: 0;
    width: 0.1%;
    height: 100%;
    background: #2C9AB7;
    animation: spinner 4s infinite;
    animation-timing-function: ease-in-out;
}
```

Figure 2 CSS style for spinner

```css
@keyframes spinner{
    0%  {width: 1%; left: 0;}
    12% {width: 8%; }
    25% {width: 1%; left: 49.5%;}
    37% {width: 8%; }
    50% {width: 1%; left: 99%;}
    62% {width: 8%; }
    75% {width: 1%; left: 51.5%;}
    87% {width: 8%; }
    100%{width: 1%; left: 0;}
}
```

Figure 3 CSS keyframe animation

2.1.5 Application Programming Interface

abbr. API

Application Programming Interface is a common term used for basic development packages such as class libraries. Usually any kind of programming language has a standard set of APIs that can be extended or used as provided.

2.1.6 JavaScript programming language

abbr. JS

A scripting language originally intended to use within web pages to make them dynamic in runtime, in contrast with server-side rendering. Lately JavaScript has been adopted in server-side scripting as well. Unlike many programming languages, JavaScript has very limited access to the hardware of the system. JS is run in a sandbox environment, and for a reason. Sandbox provides the security that is necessary for scripts that are run automatically when the user enters almost any web site.

2.1.7 Representational State Transfer

abbr. REST

REST or RESTful web service is a stateless web service which can serve JSON and XML transport formats. In the context of this document, JSON is used in conjunction with REST. The main principle for REST is to use HTTP methods, such as GET, POST and PUT, explicitly, be stateless and expose directory structure-like URIs. (Rodriguez, 2015)

2.1.8 JavaScript Object Notation

abbr. JSON

JSON is a transport format natively supported by JavaScript. In comparison with XML, instead of XML node tree, JSON describes a serialized JavaScript object which often has less overhead in terms of data transfer.

```
{   // {} notation represents an object
    "listings": [   // [] notation represents a list
        {
            "make": "BMW",
            "name": "BMW X5",
            "tecnicalDetails": {
                "engine": {
                    "fuel": "Diesel",
                    "size": 3.0
                },
                "topSpeed": 200,
                "acceleration": 10.1,
                "torque": 410,
                "power": 160,
                "mass": 2650,
                "gearbox": "automatic",
                "mileage": 351000,
                "drive": "Four wheel"

            },
            "vehicleClass": "Car",
            "price": 10300
        },
        {
            ...
        }
    ]
}
```

Figure 4 A sample JSON representation of vehicle sales listing

JSONP is a technique where name of a JS function is included in HTTP request parameter usually named “callback”. The servlet that process the request then wraps the JSON object with the function provided.

While HTML can use JavaScript source files for scripting, it cannot utilize JSON objects without padding. Hence JSONP or JSON with “padding”

2.1.9 Case Management

Case Management is concept of managing bundles of documents and metadata, i.e. cases, through various workflows and processes. A (business) process can be considered as a path from customer engagement to billing via milestones such as successful sale, project kick-off and project hand-over. A workflow is a guideline or a set of steps for how certain task, such as project hand-over is completed. (Capital BPM, 2017)

2.2 Common Design Patterns

2.2.1 Single purpose principle

Single purpose principle suggests that a single module, class or function should serve one and only one purpose. E.g., a method named getDataFromDatabase should only connect to the database, get the information specified in the function call arguments and return it to the caller. According to the single purpose principle, the method described should not alter the data received because altering the data is not something that the name of the method specifies.

2.2.2 Model-View-Controller

abbr. MVC

MVC paradigm is used in a modern application design. The paradigm suggests that the parts of the MVC are separated from each other. Model components are used to provide the data model. Views are UI components that display the data and user controls. Controllers provide interfaces for accessing, persisting and altering the model.

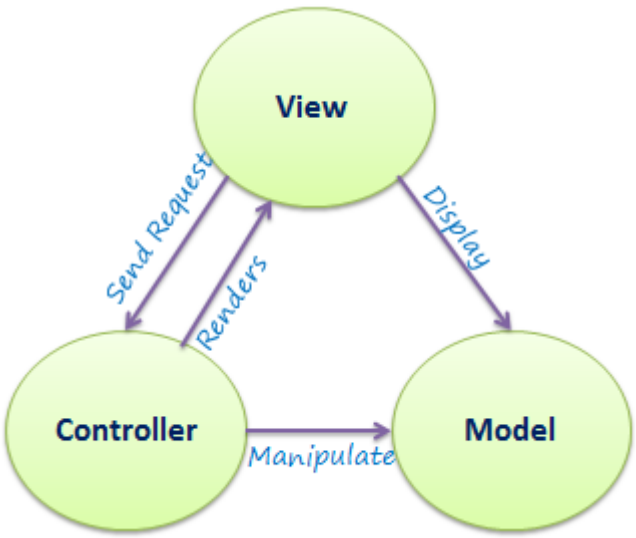


Figure 5 A diagram visualizing MVC model relations (Tutorials Teacher)

2.2.3 Dependency Injection

Dependency injection is one form of inversion of control. In procedural programming, it is common that dependencies of the procedure are resolved inside the procedure. In object-oriented programming this control pattern is often inverted thus moving responsibility of resolving dependencies from the called object to the caller. (Spring - Dependency Injection)

2.3 JavaScript Frameworks

2.3.1 Dojo Toolkit

Dojo Toolkit is a JavaScript toolkit that uses asynchronous module definition or AMD for loading modules. AMD enables that any dependency will be loaded only when needed and only once. Therefore, the subsequent calls to the dependency use the already loaded code instead of reloading it.

2.3.2 JSX

A statically typed, class-based object-oriented programming language that uses JavaScript and type annotations as expressions and statements. JSX is created to improve productivity and quality of the code. (Oku, 2013)
“JSX is an XML-like syntax extension to ECMAScript without any defined semantics. It's NOT intended to be implemented by engines or browsers. // It's intended to be used by various preprocessors (transpilers) to transform these tokens into standard ECMAScript.” (Facebook Inc., 2014)

2.3.3 ReactJS

A modern, JavaScript based technology for creating web applications developed by Facebook. When an application is compiled before deployment, ReactJS takes an advantage of JSX. Instead of JSX, ReactJS can be written with Babel dialect as well.

Babel transforms code to pure JavaScript on runtime allowing changes to the application without need of re-compiling and even usage of JavaScript expressions that are not yet supported by web browsers.

```
class HelloMessage extends React.Component {
    render() {
        return (
            <div>
                Hello {this.props.name}
            </div>
        );
    }
}
```

Figure 6 Using JSX

```
class HelloMessage extends React.Component {
    render() {
        return React.createElement(
            "div",
            null,
            "Hello ",
            this.props.name
        );
    }
}
```

Figure 7 Using Babel

2.4 Java Technologies

2.4.1 Spring Web MVC

A Java MVC API that is a part of the Spring framework. Spring MVC provides APIs such as CORS, Web Security and HTTP/2 for web servlet.

2.4.2 Java Generics

A generic type is a generic class or interface that is parameterized over types. Generifying can make the code more robust and maintainable. Begin by examining a non-generic Box class that operates on objects of any type.

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

Figure 8 An example of class containing single Object property

Since its methods accept or return an Object, the compiler cannot enforce typed usage of the class. For instance, first a String could be passed as the object for Box, but later it could be expected to be Integer and the error would only occur in runtime.

A generic class is defined with the following format:

class name<T1, T2, ..., Tn> { /* ... */ }

```
/**
 * Generic version of the Box class.
 * @param <T> the type of the value being boxed
 */
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Figure 9 A generic adaptation of the Box class

All occurences of Object are replaced by T. A variable can be any non-primitive type, i.e. any class, interface, array type or another type variable. (Oracle, n.d.)

A real example of generic method from the web application that is a product of this thesis:

```
try {

    List<QueueDefinition> queueDefinitions = this.configurationCache.getQueueDefinitions(queueName);
    return createResponse(queueDefinitions);
} catch (AbstractException ex) {

    log.error("Stack: {}", ex);
    throw ex;
} catch (Exception ex) {

    log.error("Stack: {}", ex);
    throw new InternalServerException("IBF2500");
} finally {
    logExit();
}
```

```
protected <E> ResponseMessage<E> createResponse(E el) {
    ResponseMessage<E> message = new ResponseMessage();
    message.setPayload(el);
    message.setId(this.id);
    message.setMessageCode("200");
    return message;
}
```

The createResponse method accept any non-primitive object as a parameter. ResponseMessage object with typed payload is then returned. This leaves the evaluation of the input parameter and expected output type to the compiler.

2.5 Data Warehousing

2.5.1 FileNet P8

abbr. P8

FileNet P8, is an object database developed by FileNet Corporation, later maintained by IBM. P8 provides a set of basic object types which can then be extended to meet the requirements of complex data models. P8 provides audit trail for all objects and operations. Audit trail allows the supervisors to export reports of any access or modifications to the persisted data as well as the creation and removal of the objects. P8 can run code modules when triggered by events. It can be extended with a workflow system for process management purposes such as case management.

2.5.2 P8 Object Store

Object store is an isolated container for objects and process management. P8 can serve multiple object stores simultaneously. An object store relies on the underlying database such as IBM DB2 or Microsoft SQL Server.

3 IMPLEMENTING INTERACTIVE WEB APPLICATIONS USING DOJO TOOLKIT

3.1 IBM Case Manager

IBM Case Manager is a case management product developed by IBM. It provides toolset for designing case data models, views, workflows and roles. A designer client called Case Builder is used to design the required case components.

The process of designing and developing a solution for case management with Case Manager can be briefly described as follows:

1. The solution definition is created in the design object store.
2. The properties for solution are defined. A property can be configured with name, datatype, default value and depending on the datatype, with various other settings.
3. Document types are defined. A document type can have any of the properties defined for a solution. A document type can also extend an existing document type. A use case could be a set of different insurance document definitions where some of the properties are common between the documents and some are specific to the type of the document.
4. Roles are defined. Roles can be called for example Customer Service Representative, Correspondence Team, or Fraud Investigator. A task assigned to a role can be accessed by any member assigned to the respective role unless additional restrictions apply.
5. Case types are defined. A solution can contain multiple case types. A case type has a unique set of properties, views, folders, business rules and tasks. Only properties defined for a solution can be used for case type.
6. Pages are defined. A page can be configured with one of the pre-defined layouts or with a customized layout. Various widgets can be placed on a page. Any managed visible or hidden UI component is referred to as a widget in the Case Manager. Widgets can be configured to listen and send events to and from each other. Case Manager recognizes two types of events. Published (or wired) events that are only accessible between wired components, and broadcasts that are accessible by any widget on the page.

7. The solution is deployed in the object store. Deployment generates the views and models defined in a solution definition and makes the solution available for use.

(IBM, n.d.)

Tailored widgets are feasible to develop when the basic functionality of the Case Manager is too limited for the intended purposes. Case Manager allows for external data fields to be included in *properties view*, but there are situations when even that is not enough. For example, a business solution would require a highly-customized UI with built-in automation and even an access to custom objects in the object store. In such occasion a custom widget is usually the best approach for achieving the goal.

A Case Manager widget consists of three parts. First, a Java plugin for Content Navigator which loads the plugin and enables the widget to be referenced in any application running atop Navigator. The second part is a JSON formatted registry information that Case Manager uses to recognize the widget and its properties. Properties can contain information such as event listeners and event publishers or custom properties. The third part is the actual widget written in JavaScript using Dojo.

A Navigator plugin for Case Manager widgets is not exclusive to one widget. It can provide a whole category of widgets. That is, the plugin can contain many widgets virtually for any purpose. My preference is to create plugins using single responsibility principle.

3.2 Dojo

Dojo provides characteristics of object-oriented programming to otherwise prototype-based JavaScript language. It implements the model and view layer components of MVC paradigm. One of the Dojo's main features is the ability to control its components beyond the limits of traditional DOM handling.

Dojo code is separated into different packages by the purpose of the classes. The most frequently used packages are called dojo and dijit. The others are special packages such as code for mobile and experimental, work in progress features.

- *Dojo* package contains Dojo's model classes such as *xhr* for remote procedure calls and *Memory* for indexed list type containers.
- *Dijit* package contains Dojo's UI components such as *BorderLayout* for a user resizable layout and *FilteringSelect* for a choice list with free text filtering.

Dojo classes are written similarly to RequireJS classes. An additional layer is added by declare function which returns a class like object instead of a function. A class definition is wrapped inside a function named define. When executing code, require function is to be used instead.

```
define(["dojo/_base/declare",
    "dojo/_base/lang",
    "dojo/Deferred"], function (declare, lang, Deferred) {
    return declare("icmdocumenttemplate.util.BlockingQueue", null, {

        /**
         * Add function to the queue
         */
        queue: [],
        interval: null,

        addPromise: function (f) {
            let queueLen = this.queue.length;
            this.queue.push(f);

            let remove = lang.hitch(this, function(){
                this.queue.pop();
            });
            f().then(remove, remove);
        },

        waitForFinish: function (timeout, interval) {...}

    });
});
```

Figure 10 An example of simple dojo class. BlockingQueue gathers asynchronous calls and waits until all of the calls are completed. Function waitForFinish returns a promise which resolves when all promises in the queue have been resolved.

3.3 Spring

Spring, a very powerful Java framework, is widely used in this project. The RESTful service that provides endpoints for the user interface relies heavily on the Spring Web MVC implementation. One of the strengths of Spring is that wiring several Java

components in a single context is made ridiculously easy and efficient by the vast usage of Java annotations.

3.4 The Case Study

3.4.1 Premise

The case study was conducted to resolve a problem with customized in-baskets in the ICM. Configuring in-baskets requires quite a few steps to complete and as the cherry on top of the cake, there is little to none validation applied to the configuration. I.e. one has to know exactly what has to be set and how in order to have a working configuration.

For example, there is filtering configured for an in-basket. As seen in Exhibit a, there is a missing single quotation mark after OV_Osku. Usually similar errors are handled by the executing software, but this is not the case. The software run the SQL clause happily and the results were quite obvious: filtering did not work.

```
SolutionIdentifier = 'OV_Osku and F_Subject = 'Ilmoitus'
```

Exhibit a Faulty SQL WHERE clause

According to IBM (IBM, n.d.), the desired way to create and edit in-baskets is using a tool called Process Designer. In our case, this would have been impossible because any changes made to the in-baskets in Process Designer were not saved at all. This is most likely a bug introduced in the version of the Case Manager used by the customer, because similar problem did not occur in earlier versions. To circumvent the bug, we used another tool called Process Configuration Console. In opposite to Process Designer which only affects entities in the scope of a solution, the Process Configuration Console is used to configure system wide entities including in-baskets. The problem with the system wide configuration is that it affects every ICM solution deployed in the system. With filtering introduced above we could limit the configuration to the specific solution, but it is not a good practice to filter something exclusively on lower level when it should be a built-in feature in higher level execution.

To circumvent the problems indicated above, and to liberate us from using the Java applets that Process Designer and Process Configuration Console are, a port of the Process Configuration Console was to be created.

3.4.2 Architectural specification

Initially the plan was that to replace the original in-basket widget with a customized widget that would rely on a custom back-end application. Soon it became quite clear that there is no reason to replace the client widget, which already functions as it's supposed to.

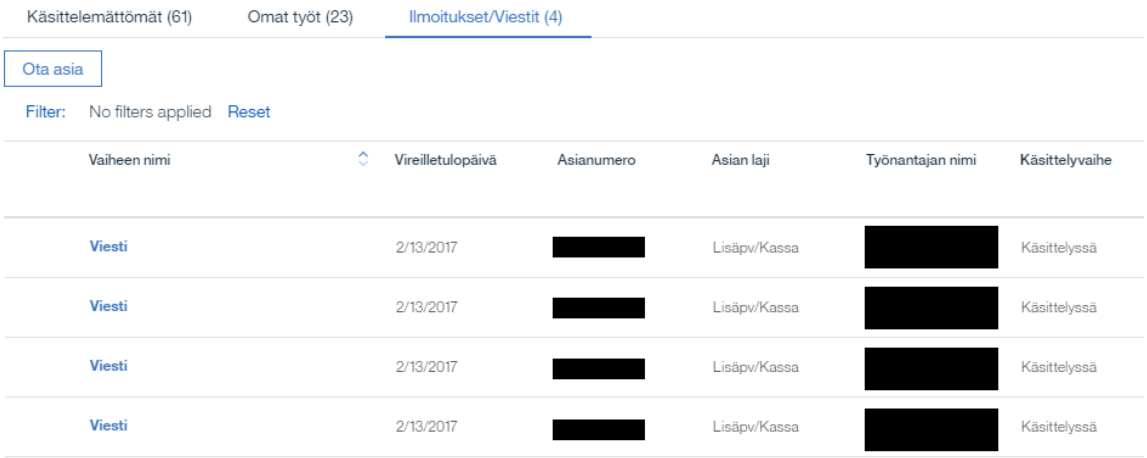


Figure 11 A screenshot of the in-basket widget

There was a change of plan regarding the back-end and configuration solution as well. Initial architecture design would have required a separate data model and a tailored handling of all the business work objects, thus a much more convenient approach was considered. Instead of creating a separate configuration and handling, I decided to make a partial Dojo based port of the original Process Configuration Console software. While keeping the functionality almost identical, some corners were roughed for more convenient and streamlined user experience.

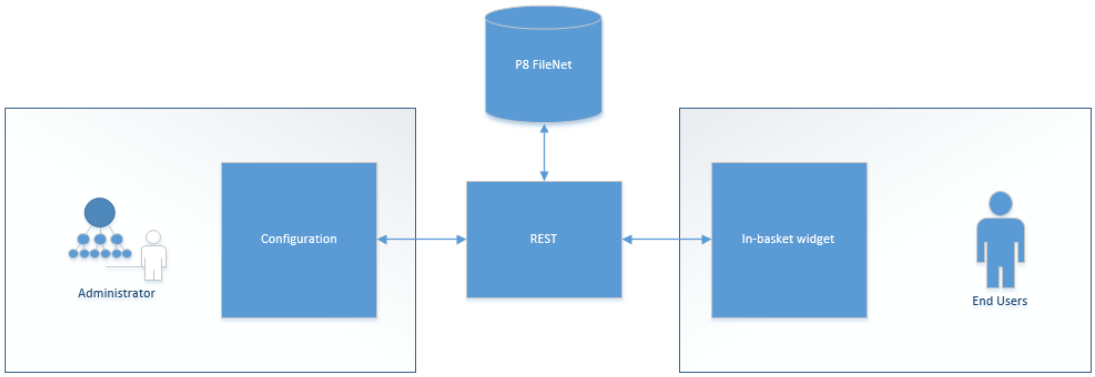


Figure 12 An illustration of the relations between components

3.4.3 User Experience – Queue configuration

1. User selects the queue for in-baskets

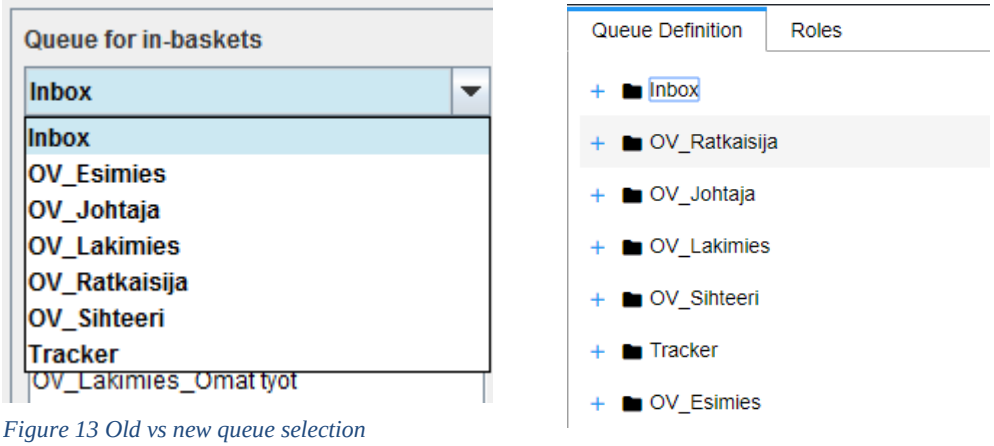


Figure 13 Old vs new queue selection

2. User selects the in-basket or creates a new in-basket. An empty in-basket can be created, or an existing in-basket can be used as a template.

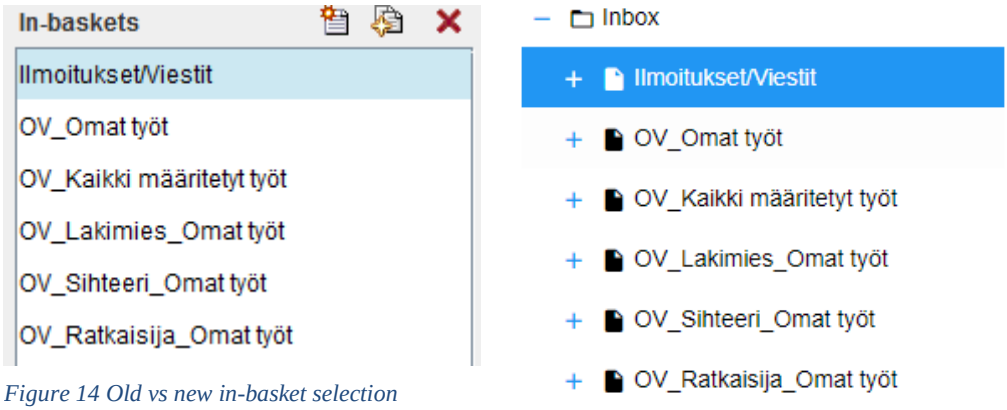
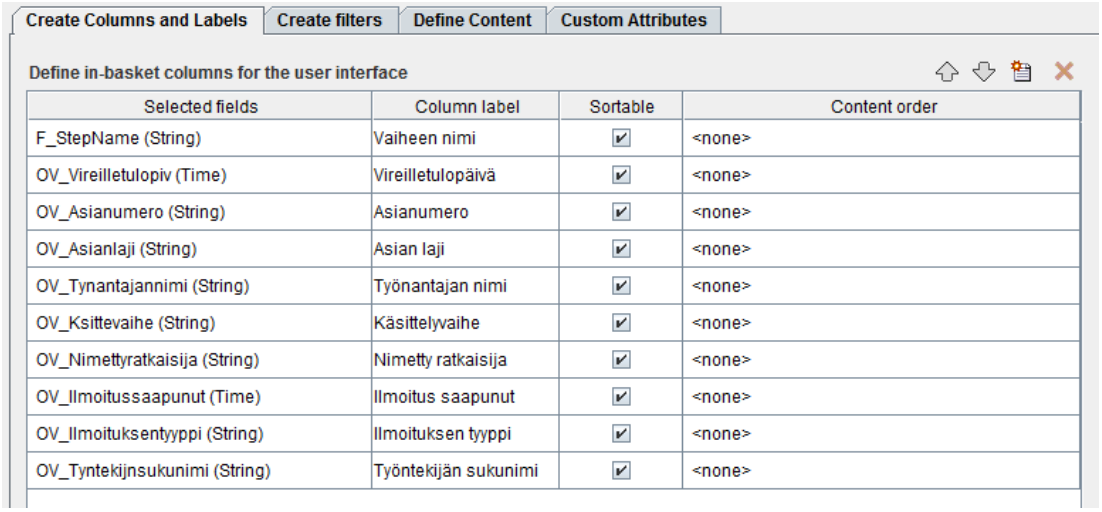


Figure 14 Old vs new in-basket selection

3. User defines the configuration for the in-basket



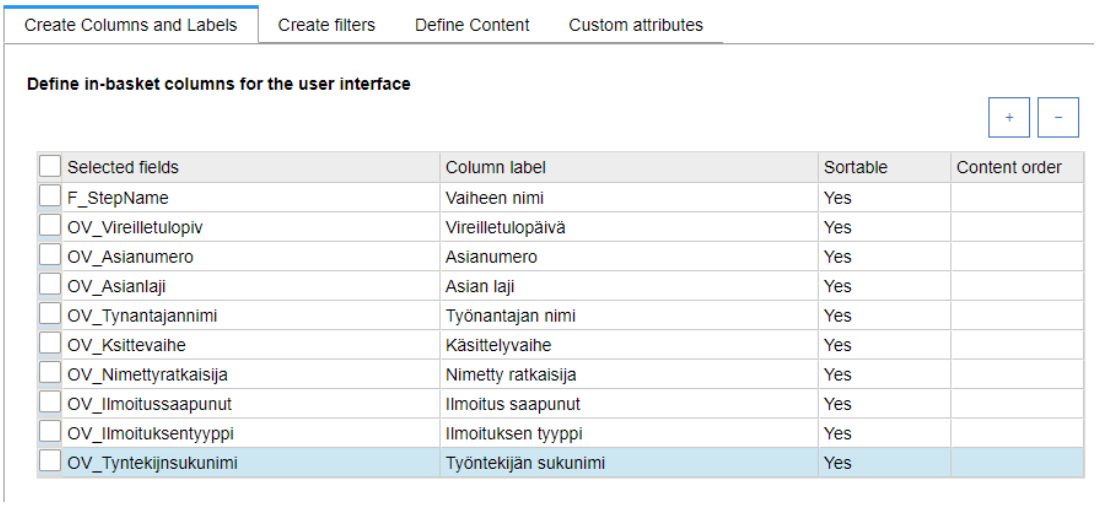
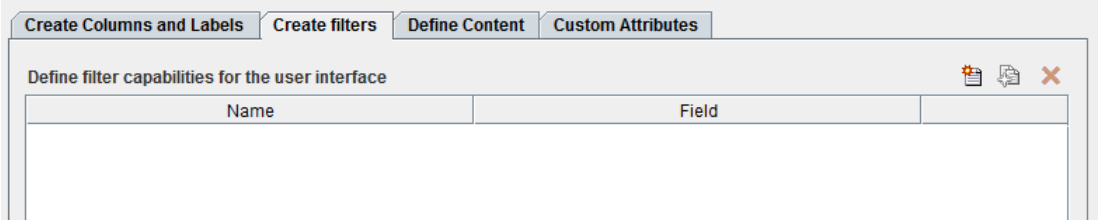


Figure 16 Old vs new column definition



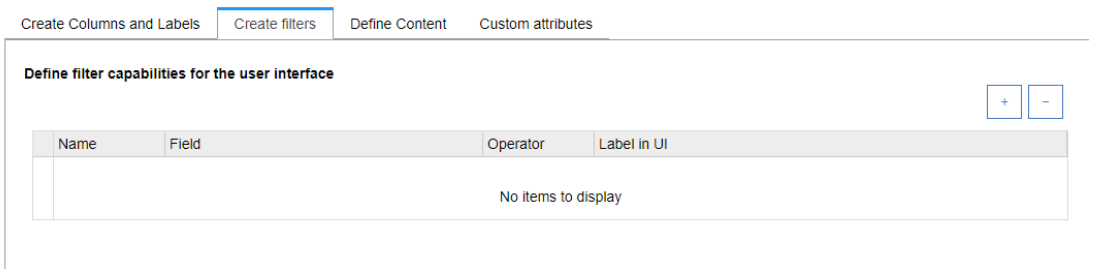


Figure 15 Old vs new filter creation. Filters defined here can be applied in the client user interface

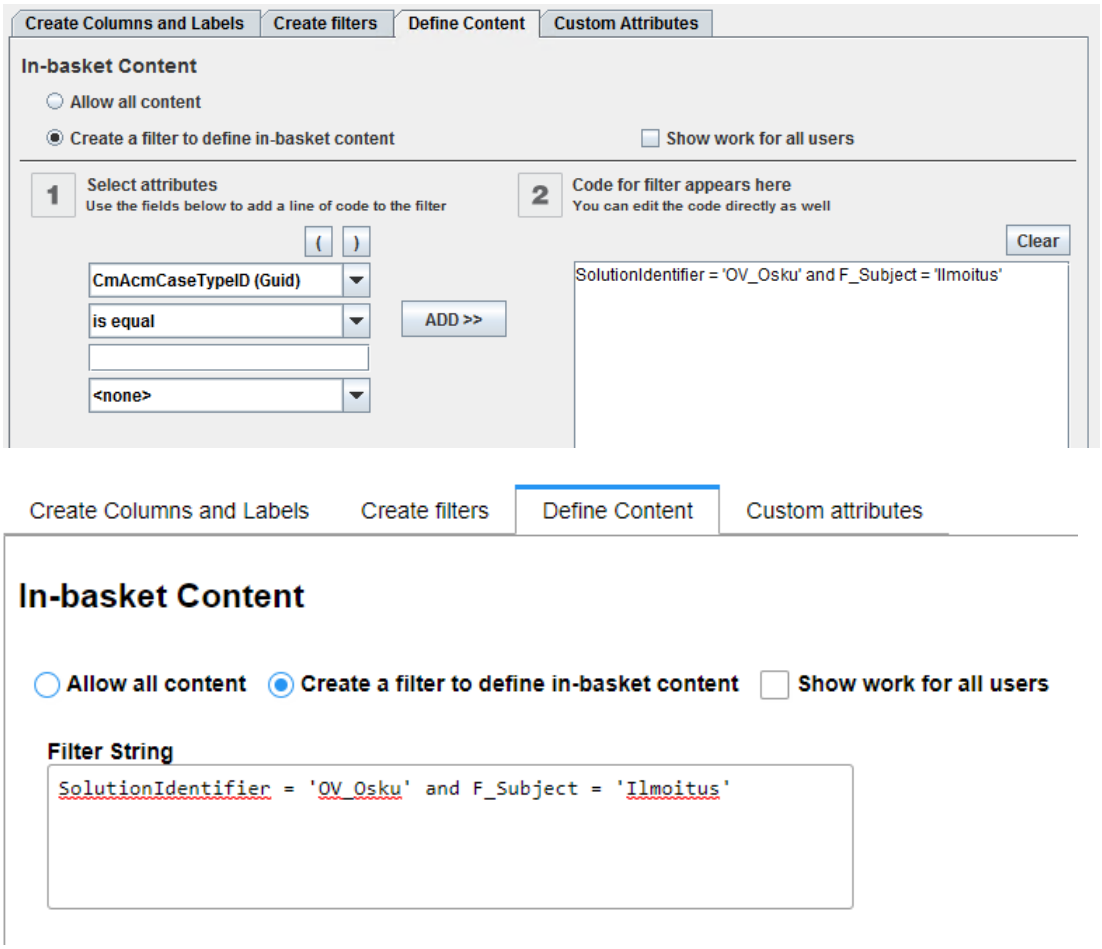


Figure 17 Old vs new content filtering. Attribute creation assistant seen in the original tool may be implemented in the future

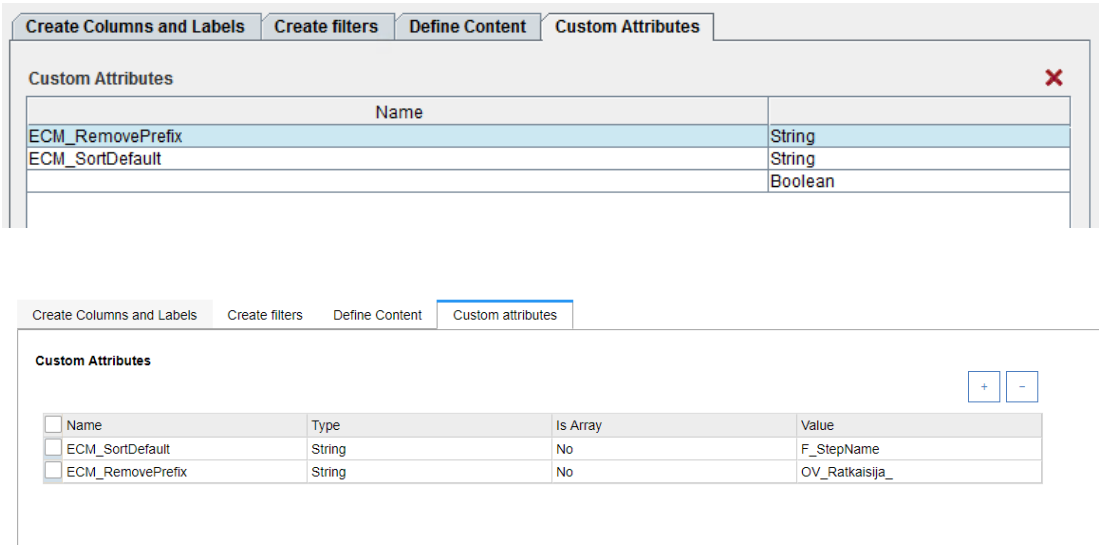


Figure 18 Old vs new custom attributes definition

4. User saves the configuration

As the figures 13-16 indicate, there are different implementations of how new row is added to the table in the original software. In the new design all controls are cohesive.

3.4.4 RESTful service endpoints

The purpose of the REST service is to act as a controller for the UI components. The configuration is serialized and saved in a XML file stored as a document in P8 FileNet object store.

The following endpoints are required for the UI components:

- /api/post/addInbasket
    - Used to create a new in-basket
    - HTTP POST method
    - Initial configuration is provided in the request body
    - Possible response codes are 201, 400, 401, 403, 500
    - The response to the successful creation contains the persisted entity
- /api/delete/removeInbasket/{id}
    - Used to remove an existing in-basket
    - HTTP DELETE method
    - The id of the in-basket is provided in the path variable id
    - Possible response codes are 204, 400, 401, 403, 500
    - The response has no content on successful deletion
- /api/put/updateInbasket
    - Used to update the configuration of the in-basket
    - HTTP PUT method
    - The configuration is provided in the request body
    - Possible response codes are 200, 400, 401, 403, 500
    - The response to the successful update contains the persisted entity
- /api/get/retrieveInbaskets?role={roleName|*}
    - Used to retrieve in-baskets

    - HTTP GET method
    - The name of the role is provided in the path variable roleName
    - Possible response codes are 200, 400, 401, 403, 500
    - The response contains all configured in-baskets for the role

The HTTP response codes are explained as follows:

- 200 OK
    - Returns content
- 201 Created
    - Returns created entity
- 204 No Content
    - Operation completed successfully, no content in the response
- 400 Bad Request
    - The request does not meet the specification.
- 401 Not Authenticated
    - The use is not logged in or the session has expired
- 403 Forbidden
    - The user logged in does not have appropriate permissions for the requested operation
- 500 Internal Server Error
    - An error occurred during the execution of the operation. Possible errors are caused by internal connection problems or software bugs.

3.4.5 RESTful service

The service is running as an enterprise application in IBM WebSphere Application Server or IBM WAS (Appendix 1 – WebSphere). Creating a standalone service would have been possible as well but WAS provides out of the box security context with LDAP integration and isolated class loaders for shared class libraries thus making it feasible deployment platform.

The framework stack for the service includes:

- JavaEE 7

- Spring
    - Beans, context, web, webmvc, jdbc
- Jackson Databind
- Springfox Swagger 2
    - Extensive json REST endpoint description and testing tool

```
@RestController
@Scope("prototype")
public class RestService extends AbstractRestController implements ApplicationContextAware {

    private ConfigurationCache configurationCache;

    @PostConstruct
    public void init() { this.configurationCache = ctx.getBean(ConfigurationCache.class); }

    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Query completed"),
            @ApiResponse(code = 400, message = "Invalid request"),
            @ApiResponse(code = 401, message = "Unauthorized"),
            @ApiResponse(code = 403, message = "Not Authenticated"),
            @ApiResponse(code = 500, message = "Internal server error")})

    @ApiOperation(value = "Retrieve Queues")
    @RequestMapping(
            path = "/api/get/retrieveQueues",
            method = RequestMethod.GET,
            produces = "application/json;utf-8")

    public ResponseMessage<List<QueueDefinition>> retrieveQueues(
            @RequestParam(name = "queue", required = false, defaultValue = "*") String queueName)
            throws AbstractException {
```

Figure 19 An example of service declaration class with anntotations for Spring and Swagger

Annotations provided by the following packages:

- org.springframework.web.bind.annotation
- org.springframework.context.annotation
- javax.annotation
- io.swagger.annotations

Annotations explained

- RestController is a Spring annotation declaring the class a container class for RESTful endpoints
- Scope defines the runtime scope for any object instantiated from the class. Possible scopes:
    - prototype is instantiated every time the class is request from the application context
    - session is created once per user session

    - request is created once per user request
    - singleton is a default scope. It's instantiated when the application is started or the first time it is requested from the application context

- "PostConstruct annotation is used on a method that needs to be executed after dependency injection is done to perform any initialization"
- "ApiResponse describes a possible response of an operation"
- "ApiResponses is a wrapper to allow a list of multiple ApiResponse objects"
- "ApiOperation describes an operation or typically a HTTP method against a specific path"
- RequestMapping is "annotation for mapping web requests onto methods in request-handling classes with flexible method signatures."
- RequestParam defines the mapping for the web request parameter. In Figure 19 queueName is mapped to URI parameter 'queue' and is marked as optional with default value of '*'
- RequestBody (Figure 20) defines the mapping for object serialized in JSON format

(frantuma, n.d.; Oracle, n.d.; Spring, n.d.)

```
@RequestMapping(path = "/api/post/addInbasket",
        method = RequestMethod.POST,
        produces = "application/json;utf-8")
public ResponseMessage<InBasketDefinition> addInbasket(
        @RequestBody OperationRequest<InBasketDefinition> body)
        throws AbstractException {
```

Figure 20 An example usage of RequestBody annotation

4 CONCLUSION

Modern software development schemes, technologies and patterns allow for highly sophisticated and robust solution design and implementation. Latest frameworks blend model and view components of the MVC model and sometimes even throw in some controller behavior. While it seems to reduce the boilerplate to it's minimum, I am concerned about long term maintainability. In other words, new capabilities and behavior call in for strict and modular architecture design.

On the server side I had to use Java programming language and Java EE framework because the back-end solution must be a Java servlet. This was not a limiting factor in any manner. In my opinion Java EE with Spring Web and Web MVC frameworks establish a solid base for business solution.

While the technologies used in the case study were mostly enforced by the target environment, there could have been more agile and up-to-date technology used. For instance, developing web-based user interface using Dojo Toolkit is not as straightforward and maintainable as ReactJS. If I would re-create the user interface, I would probably build the view components with React and let Redux handle all the wiring between the modules.

REFERENCES

Capital BPM. (2017, May 15). *Workflow vs Business Processes: The Pizza Analogy*. Retrieved from Capital BPM: https://www.capbpm.com/workflow-process/

Facebook Inc. (2014). *Draft: JSX Specification*. Retrieved from Facebook Github: https://facebook.github.io/jsx/

Feri Clayton, I. (2017). Retrieved from ibm.com: https://www-01.ibm.com/events/wwe/grp/grp309.nsf/vLookupPDFs/IBMContent2017%2022giugno%20Roma%20Strategy%20and%20Roadmap%20FeryClayton/$file/IBMContent2017%2022giugno%20Roma%20Strategy%20and%20Roadmap%20FeryClayton.pdf

frantuma. (n.d.). *Swagger Core Annotation*. Retrieved from GitHub: https://github.com/swagger-api/swagger-core/wiki/annotations

IBM. (n.d.). *Case Management*. Retrieved from IBM Knowledge Center: http://www.ibm.com/knowledgecenter/casemanager

IBM. (n.d.). *Creating more than one in-basket in Process Designer*. Retrieved from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSCTJ4_5.2.1/com.ibm.casemgmt.help.doc/acmpdh03.htm

Manjum. (2017, Oct 18). *Is ICN going away from DOJO*. Retrieved from IBM developerWorks: https://developer.ibm.com/answers/questions/407836/is-icn-going-away-from-dojo/

Oku, K. (2013, Jun 8). *JSX - developing a statically-typed programming language for the Web.* Retrieved from LinkedIn SlideShare: https://www.slideshare.net/kazuho/jsx20130608pptx

Oracle. (n.d.). *Annotation Type PostConstruct*. Retrieved from Oracle Docs: https://docs.oracle.com/javaee/7/api/javax/annotation/PostConstruct.html

Oracle. (n.d.). *Generic Types*. Retrieved from The Java Tutorials: https://docs.oracle.com/javase/tutorial/java/generics/types.html

Rodriguez, A. (2015, February 09). *RESTful Web services: The basics.* Retrieved from IBM developerWorks: https://www.ibm.com/developerworks/library/ws-restful/ws-restful-pdf.pdf

Spring - Dependency Injection. (n.d.). Retrieved from Tutorialspoint: https://www.tutorialspoint.com/spring/spring_dependency_injection.htm

Spring. (2017, 11 27). *Web on Servlet Stack*. Retrieved from Spring Docs: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html

Spring. (n.d.). *Web Bind Annotation*. Retrieved from Spring Docs: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/

Tutorials Teacher. (n.d.). *MVC Architecture*. Retrieved from TutorialsTeacher.com: http://www.tutorialsteacher.com/mvc/mvc-architecture

U.S. Department of Health and Human Services. (n.d.). *Usability.gov*. Retrieved from User Interface Design Basics: https://www.usability.gov/what-and-why/user-interface-design.html