# ECOMMERCE MOBILE APPLICATION WITH IBM WEBSPHERE

Maciej Kopeć

Bachelor’s Thesis

April 2012

Degree Programme in Software Engineering

School of Technology

| | | |
|---|---|---|
| Author(s)<br>Kopeć, Maciej | Type of publication<br>Bachelor´s Thesis | Date<br>13-04-2012 |
| | Pages<br>52 | Language<br>English |
| | Confidential<br>( ) Until | Permission for web publication<br>( X ) |
| Title<br>ECOMMERCE MOBILE APPLICATION WITH IBM WEBSPHERE | | |
| Degree Programme<br>Software Engineering | | |
| Tutor(s)<br>Peltomäki, Juha | | |
| Assigned by<br>Descom Oy | | |

Abstract

IBM WebSphere Commerce is an enterprise application that provides high levels of scalability, performance, security, reliability and manageability. It provides a modern tool for business information exchange and eCommerce.

WebSphere Commerce products include reliable, scalable and secure runtime environment with modern JEE programming model. WebSphere Commerce programming model consists of a mix of technologies including JavaServer Pages, Enterprise JavaBeans, JavaServer Faces, SOAP and Struts. WebSphere Commerce supports mobile versions of web store including native and hybrid applications.

This thesis adds a new detail to the development process of a hybrid mobile application for Android Operating System with a new feature for handling and scanning barcodes. In the first part of the thesis, theoretical basics are introduced. The architecture of WebSphere Commerce application is briefly introduced with  a programming model, mobile platform, barcode specification and libraries used in the thesis. In the following chapters, the development process is explained. Finally, there is the implementation chapter with the development of an Android hybrid application, implementation of the barcode generator and testing the application.

The last chapter presents the author’s personal experiences and conclusions concerning this thesis. It also presents possibilities for future development. The thesis shows how to implement a fully-featured mobile hybrid application working on Android Operating System.

Keywords
IBM WebSphere, WebSphere Commerce, Mobile application, Android, JEE, JSP, Barcodes

Miscellaneous

# CONTENTS

# LIST OF FIGURES

## LIST OF TABLES

## ACRONYMS AND TERMINOLOGY

**Ant** - is a tool for automates process of software building. Mainly used with applications crated in Java language. Building process in describe in XML file.

**B2B** - Business-to-business, also called as “classic e-business”. It describes relations between companies and their partners.

**B2C** - Business-to-consumer is defined as using e-resources to making transactions between companies and customers

**Business on demand** - it is one of the e-business models where organisation connects its core business systems to key areas using intranets, extranets, and the web.

**EJB** - Enterprise JavaBeans is technology working on the server side. EJB is a part of Java EE. There are few types of EJBs designed to solve different kind of problems (implementing application logic, persistent data or asynchronous processing)

**JEE** - Java Platform, Enterprise Edition is extended from Java Platform, Standard Edition by providing APIs for multitier architectures, ORM, fault-tolerance and Web services.

**JPA** - Java Persistence API is a Java framework for Object-Relational Mapping in Java Platform, Standard Edition and Java Platform, Enterprise Edition applications.

**JSF** - JavaServer Faces is a Java framework for designing user interfaces for JavaEE applications.  At the present, default views technology for JSF pages is Facelets technology. However, it is possible to use different technology, for example JSP

**JSON** – JavaScript Object Notation is lightweight data format. JSON is text format. Regardless of the name, JSON is independent of particular programming language.

**JSP** - JavaServer Pages is technology designed for creating dynamic WWW pages in formats: HTML, XHTML, DHTML and XML. JSP uses Java language inside HTML code. This solution is similar to PHP.

**JSTL** - JavaServer Pages Standard Tag Library is a component of the JEE. It is extending JSP specification by adding library of JSP tags.

**RAD** - Rational Application Developer is Eclipse-based integrated developing environment built by IBM. It is designed for developing JEE applications. Specially with IBM WebSphere.

**REST/RESTful** - Representational State Transfer is architectonical pattern. REST introduce uniform interface, stateless communication, cacheable resources and layered system. A RESTful is a web service implemented using HTTP protocol and the principles of REST.

**SEO** - Search Engine Optimization is process of optimization site for search engines.

**Struts** – it is open-source framework for application developed with Java language. It take care of *view* and *controller* from MVC design pattern.

# 1 INTRODUCTION

Java language serves multiple purposes. One of the most popular purposes is creating enterprise web applications. At the beginning, simple technologies like servlet were enough to meet requirements of developers. However, it became obvious that servlet technology is not enough to build enterprise applications.

The response to the increasing requirement for Java language was the introduction of Java Platform Enterprise Edition. JEE defines a standard of application development in Java language based on multi-component architecture. This platform defines a set of programming interfaces, which an application server must provide. Those components are usually embedded in an application server that supports Java Platform Enterprise Edition.

There are many application servers capable of running enterprise applications. One of them is IBM WebSphere Application Server. It is a multiplatform application server following specifications of Java Platform Enterprise Edition. WebSphere Application Server is based on open technologies such as XML and Web services. WebSphere Application Server provides a wide variety of services. For example, database connections, thread handling, load distribution, security model. All those services can be used to improve applications by developers.

IBM WebSphere Commerce provides many ready-to-use tools with which developers are exempted from the effort to develop applications from scratch. Delivery of the starter store allows developers to focus on adjusting the product to customers' needs. IBM supplies developers with starter store called *Madison*, which can be used as a starter application for developing custom store. *Madison* is ready to use web application with basic functionality. It also comes with a mobile version of store which was used in this thesis. The main objective of this thesis was  to develop a mobile store for Android Operating System with search engine based barcodes.

# 2 IBM WEBSPHERE AS AN ECOMMERCE PLATFORM

Commerce business is changing very rapidly. Customers require from vendors easy access to their products and services. The Internet network is becoming a modern tool for business information exchange and eCommerce. If a company wants to remain competitive, it is crucial to move most of their services to accessible systems. Enterprise applications depend on confidential data. Therefore, it is necessary to provide high levels of scalability, performance, security, reliability and manageability. IBM WebSphere Commerce meets these requirements.

IBM WebSphere Commerce products include reliable, scalable and secure runtime environment with modern JEE programming model. The products come with sample *Madison* store that was used in this thesis. It is also ensures an extensive customer interaction platform for cross-channel commerce. WebSphere Commerce can be used in medium-sized companies as well as those big ones. Depending on the needs of the company, IBM provides three editions of their product. It is a standardized platform, which offers ability to do business-to-business (B2B) and business-to-customer (B2C).

There are three editions of WebSphere Commerce:

- WebSphere Commerce-Express
- WebSphere Commerce Professional
- WebSphere Commerce Enterprise.

**WebSphere Commerce-Express** is dedicated to companies that employ less than 1,000 employees. It is designed as a comprehensive solution for medium-sized companies to support business-to-customer requirements.

**WebSphere Commerce Professional** edition is designed for companies that want to do *business on demand*. It supports a benefit system, auctions and multiple workspaces.

**WebSphere Commerce Enterprise** is a comprehensive platform for high volume business-to-customer and business-to-business models. It provides extended sites, organisations system, advanced roles and requisition lists (IBM - WebSphere Commerce, 2011).

The table below compares the key differences between the three editions of WebSphere Commerce.

TABLE 1. Comparison of editions WebSphere Commerce (IBM - WebSphere Commerce, 2012)

| Features | Express | Professional | Enterprise |
|---|---|---|---|
| Search engine optimization | ✓ | ✓ | ✓ |
| Marketing tools | ✓ | ✓ | ✓ |
| Mobile commerce | ✓ | ✓ | ✓ |
| Auctions |  | ✓ | ✓ |
| B2B starter stores |  |  | ✓ |
| Gift center |  | ✓ | ✓ |
| Organizations |  |  | ✓ |
| Promotions | ✓ | ✓ | ✓ |
| Workspaces |  | ✓ | ✓ |
| Extended sites |  |  | ✓ |

This thesis is based on WebSphere Commerce version 7 Enterprise Edition that provides an extensive platform for both, B2C and B2B business models and multiple stores. Despite the fact that WebSphere Commerce includes a wide variety of features there are many issues to solve and develop. The intention of the author of this thesis is to show how to customize a *Madison mobile* store with new capabilities such a barcode scanner.

## 2.1 Application architecture

WebSphere Commerce application architecture is made up of seven layers, which are specifically designed for particular roles (see FIGURE 1). The roles are described more in Figure 1.

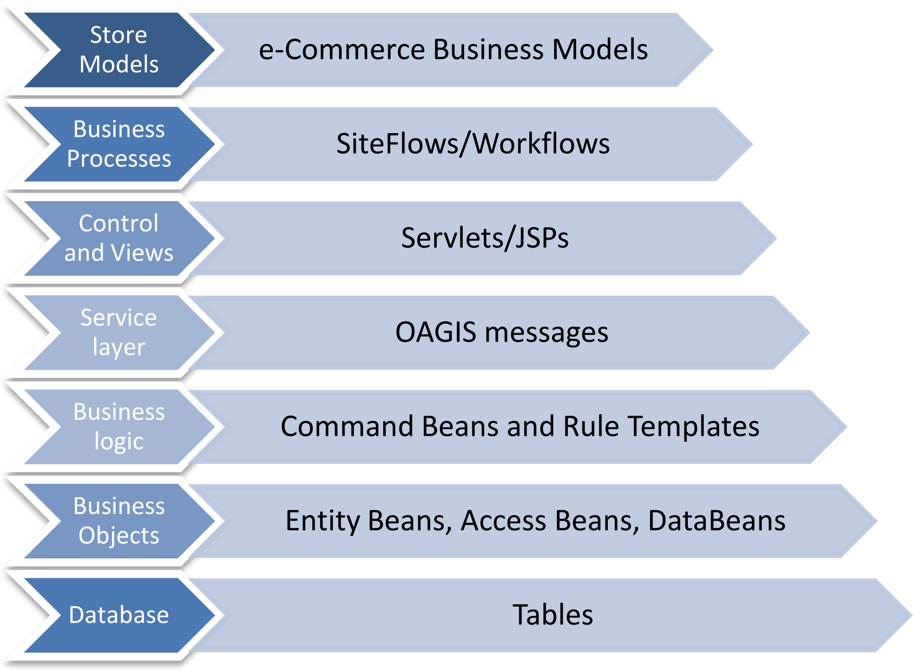


**FIGURE 1. WebSphere Commerce application architecture**

**Database**

The WebSphere Commerce contains over 600 tables, which are used to store data of the WebSphere site and store (IBM Info Center, 2011).

**Business objects**

Business objects are representation of the entities. They encapsulate data logic and work as an interface between business logic and persistent data in database (IBM Info Center, 2011). Those objects are represented by entity beans, access beans and data beans (e.g. User or Order).

**Business components**

Business components are entities of business logic. They are responsible for performing procedural business logic (IBM Info Center, 2011). Controllers and task commands represent those components (e.g. *OrderProcessCmd*).

**Controls and view**

Controls are responsible for determining which command controller implementation is the most appropriate to be used with a certain view. That implementation can depend on the store settings. The main purpose of views is to display  the results of commands and user interactions. (IBM Info Center, 2011). They are implemented as JSP pages (e.g. *ProductDisplayView*).

**Service layer**

This layer provides an independent mechanism that allows access to the business logic of WebSphere Commerce. The service layer supports two transport types: local Java binding and Web services (IBM Info Center, 2011).

**Business logic**

Business processes are sets of business components and views combined together. They create workflow and site flow processes (IBM Info Center, 2011). Order process is an example of business processes.

**Store models**

All lower layers of the application architecture create business models. Every business model includes the sample store model (IBM Info Center, 2011). WebSphere Commerce provides five business models: B2B direct, Consumer direct, Demand chain, Hosting and Supply chain.

## 2.2 Multiple-Tier Architecture

Multiple-tier architecture is a client-server architecture in which the presentation, the functional process logic and the data management are developed and maintained as independent modules. (Three-tier, 1998)

The most important part in multiple-tier architecture is the middle-tier server. This tier is responsible for handling requests from clients and hides the complexity involved with backend systems and databases. The middle-tier is able to support many types of clients, for example Web browsers, mobile devices or Java applications. Those clients can handle the user interface; however, they do not connect to a database or execute business logic. These tasks are handled by the middle-tier.

IBM WebSphere Application Server supports IBM WebSphere Commerce. It is an application server that can be run on following operating systems: IBM AIX, HP-UX, IBM i, Linux, Solaris, Windows, z/OS (Ticknor, et al., 2011).

Middle-tier servers provide many business services such as catalogue lookup, order entry and payment system. They also provide system services such as remote access, session and transaction management (see FIGURE 2)

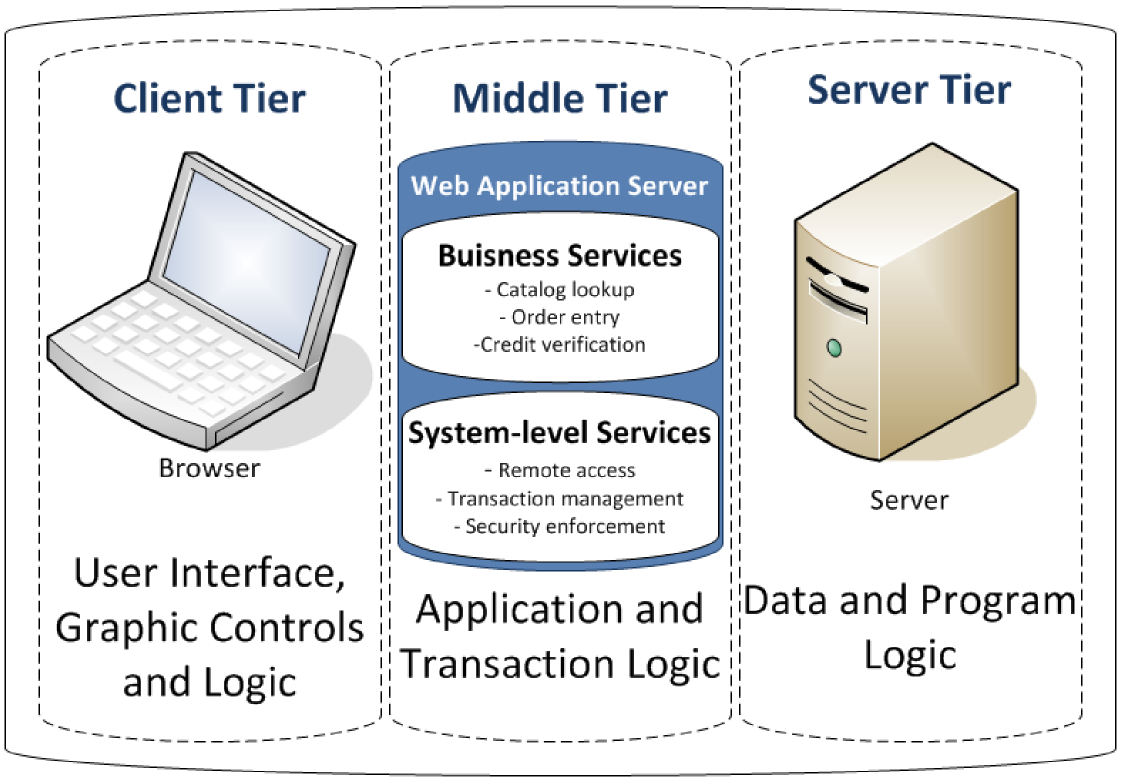


FIGURE 2. Multitier architecture, the middle tier provides business and system services to clients

With HTML5, many programmers try to implement business logic using this markup language with JavaScript. HTML and JavaScript are implemented in Client Tier. Therefore, it is not recommended to implement business logic with HTML and JavaScript. It is possible to send a request from Client Tier to Middle Tier, which will execute business logic.

## 2.3 Programming model

### 2.3.1 Programming technologies

WebSphere Commerce programming model consists of a mix of technologies including JavaServer Pages, Enterprise JavaBeans, JavaServer Faces and Struts (see FIGURE 3).

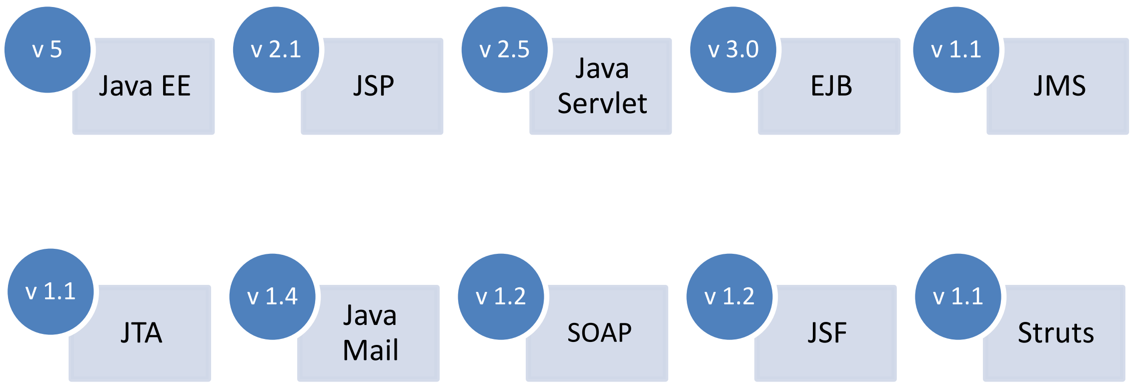


FIGURE 3. Supported JEE technologies in WebSphere Application Server 7.0

One of the most important technologies used in building enterprise applications are Enterprise JavaBeans, which are used to provide business logic and access to data. It is also possible to integrate messaging systems and Web services clients with application through Enterprise JavaBeans. There are three types of EJBs.

**Session EJBs**

Session EJBs are task-oriented objects. An EJB client invokes these objects. Session EJBs are divided into two types: *Stateless* and *stateful*. These objects cannot be persistent.

**Stateless session EJBs**

The most desired session EJBs are stateless EJBs, since they are in general scales better than stateful session EJBs. Stateless beans are designed to handle many requests from many different clients. To achieve this goal, stateless beans cannot contain any information about the state of a concrete client. Thus, all instances of stateless beans can be assigned to any client.

**Stateful session EJBs**

Stateful session EJBs are opposite to stateless beans. Stateful beans contain information about the state in the session bean between calls. Each instance of stateful bean must be associated only to one client. These kinds of beans are useful when there is a need to call several methods that depend on state information.

**Entity EJBs**

Entity EJBs are used to provide an object-oriented view of data. However, data itself is stored by an external persistence mechanism, for example database. There are two types of entity EJBs: Container-managed persistence and bean-managed persistence.

**Container-managed persistence**

In container-managed persistence the EJB container handles all database access. There is no access to the database in the code; therefore, the code is not tied to a concrete database. As a result, a project can be redeployed using different servers. On the other hand, the container must provide the mapping tool for the developers to allow them to describe how the attributes of an entity bean map onto the columns in tables of a database.

**Bean-managed persistence**

If there is a need to use non-relational data storage, developers can use bean-managed persistence. Developers handle all storage access required by the entity bean.

### 2.3.2 Model-View-Controller

Of the many requirements for enterprise applications, the ability to handle requests coming from different kinds of devices is the most important. The Model-View-Controller design pattern provides JEE applications with the possibility to work independently of the user interface. MVC keeps the business and presentation logic separate.

Model-View-Controller pattern breaks the problem of user interfaces into three units, namely: model, view and controller (see FIGURE 4). The *model* contains the application state. A *view* is responsible for interpreting the data received from the model and it sends the response to the user. The *controller* has to process user input, and depending on that, the input decides to update the model or display new view (Crawford & Kaplan, 2003, pp. 36-62).

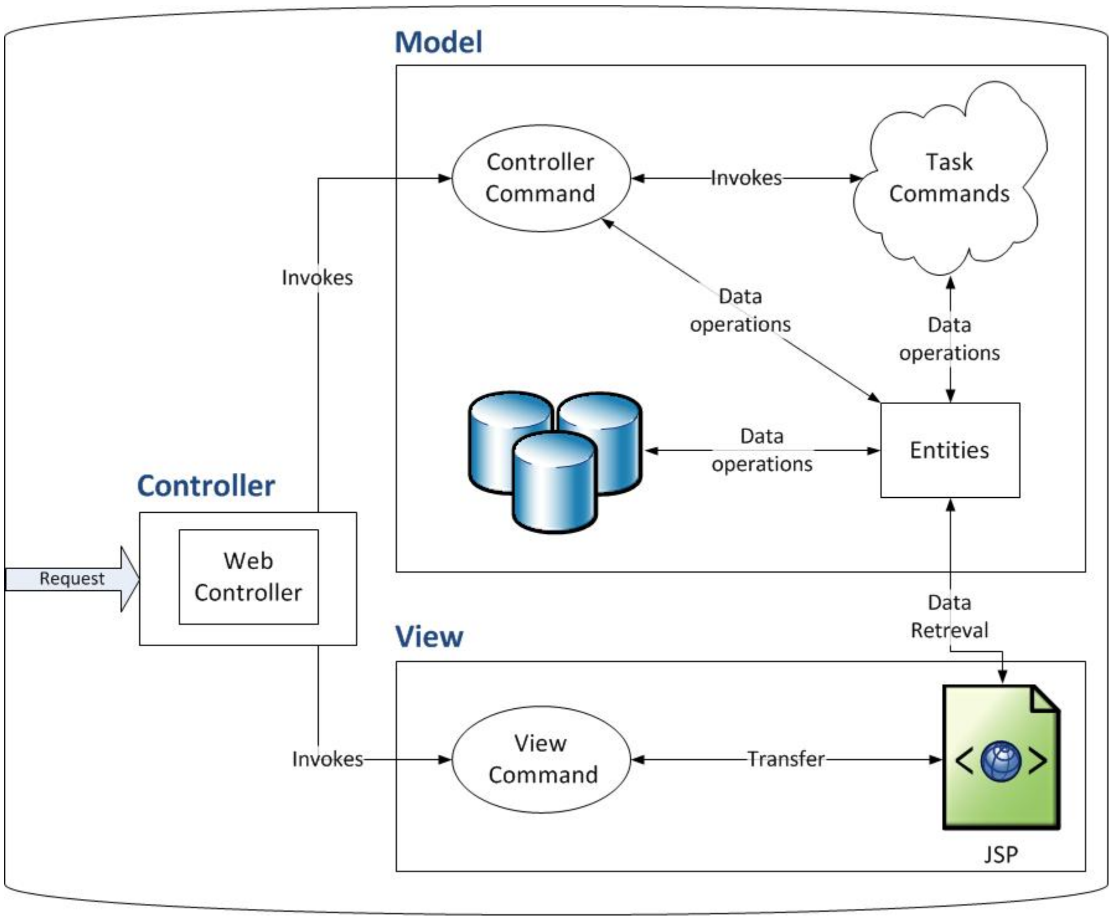


FIGURE 4. WebSphere Commerce implementation of the MVC design pattern

### 2.3.3 Struts framework

The Apache Struts is an open-source web framework for creating Java web applications. The Struts is designed to provide developers with a framework to create web applications that use MVC architecture (see FIGURE 5). Struts is able to collect data from HTTP requests. The main purpose is to pass data between the view and controller. Furthermore, Struts includes custom JSP tag libraries to help developers in creating forms. ActionServlet is provided by Struts framework. This controller servlet populates action forms from JSP inputs. After populating a form it delegates work to an action class where the developer implements the logic. Struts does not provide model classes. The developers have to provide those classes by themselves using EJB or JavaBeans (Struts 1, 2008).

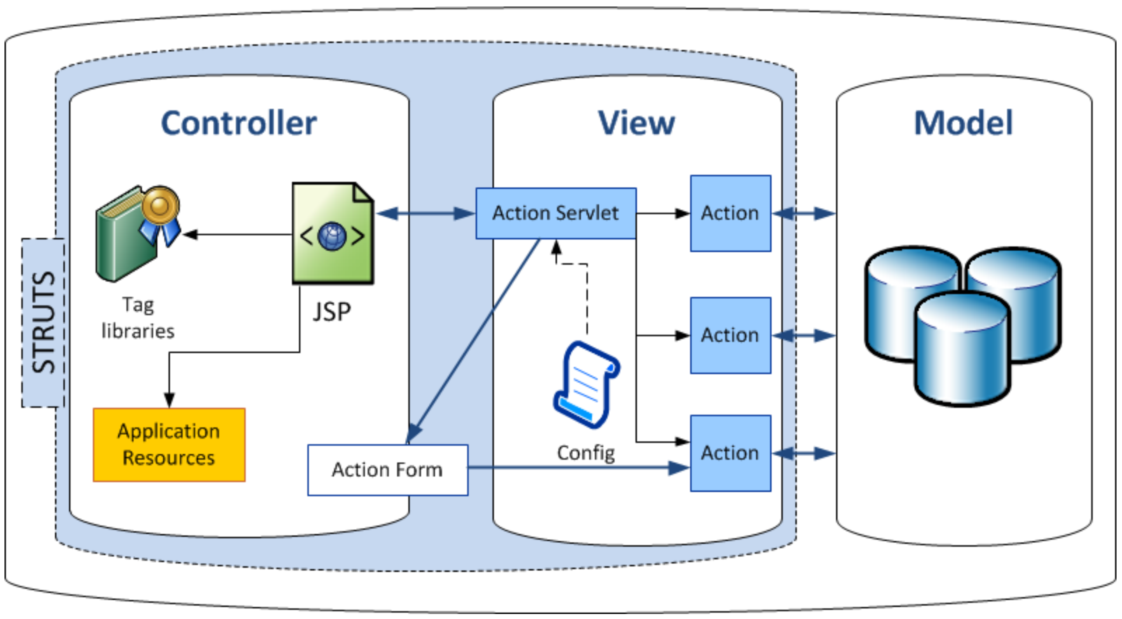


FIGURE 5. Struts as a part of MVC architecture

Struts configuration is placed into at least two files: *struts-config.xml* and *web.xml*. The *web.xml* is application descriptor, which represents the core of the Java web application. Struts *web.xml* defines servlet filters. It also defines the rest of configuration files. It is possible to define multiply *struts-config.xml* files. To add another configuration file, simply point to it in *config* field.

The *struts-config.xml* configuration file is a connection between the View and Model components of MVC. It is one of the most important components of Struts framework.

Struts configuration file contains three main elements:

- <form-bean>
- <global-forward>
- <action-mapping>.

Form-bean element contains form bean definitions. The information of each form bean is in *<form-bean>* element. Detail information is provided in <form-property> elements. Those elements contain the property names and types of form bean.

Global-forward element contains global forward definitions. Forward name is name used to map a specific JSP page. The *<forward>* element contains also path value, which represent relative path of resource.

Action-mapping element contains action definitions. Each action mapping is defined in an *<action>* element.

Besides the implementation of MVC pattern Struts also speeds up the development of the web application by providing comprehensive JSP tag library. It also supports internationalization validation of user input and error handling  (Siggelkow, 2005, pp. 217, 257, 377).

Instead of using Struts, JSP tag library developers can use JSTL library, which provides mostly the same functionality as a library provided by Struts. The choice of the library depends on developer's habits.

## 2.4 Rational Application Developer

IBM Rational Application Developer for WebSphere Software V7 is an integrated development environment and platform for building Java Platform Standard Edition (Java SE) and Java Platform Enterprise Edition (Java EE) (Wahli, et al., 2007). Rational Application Developer is based on Eclipse IDE.

Rational Application Developer contains many tools and features, for example full support of Java EE, UML editors, static and runtime analysis and extended debugging and profiling. It also supports modern Java EE technologies (see FIGURE 6).

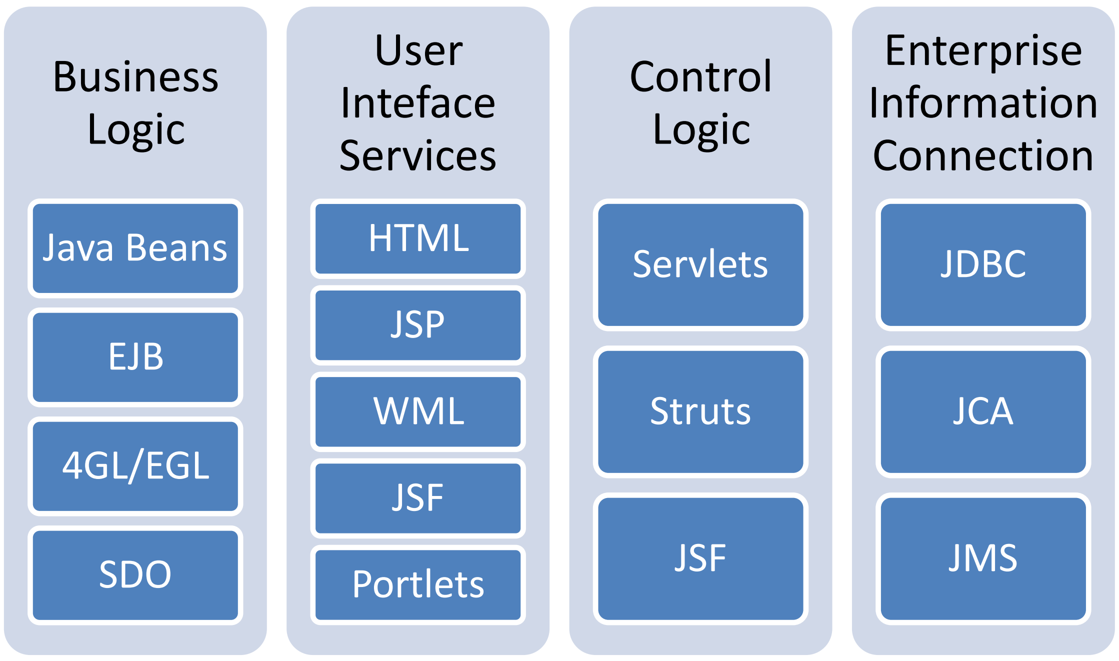


**FIGURE 6. Technologies supported by Rational Application Developer**

Rational Application Developer helps Java developers quickly develop and test Java applications. It is integrated with WebSphere Application Server.

## 2.5 Mobile platform

IBM WebSphere Commerce starter stores supports storefronts for many mobile devices. Those stores are fully featured eCommerce applications, which can be used with smart phones, tablets and web browsers. Mobile stores are designed to be user friendly for the end users of portable devices. Access to the business logic is handled by RESTful services.

Mobile store can be developed in three different ways:

- Mobile Web applications
- Native mobile applications
- Hybrid mobile applications.

**Mobile web applications**

Storefronts are available by web browser installed on the portable device (smart phone or tablet). Store pages are particularly designed as simple and tap-friendly for limited screen space.

**Native mobile applications**

Storefronts are available by application installed on smart phones or tablets. This application provides improved functionality greater than the mobile web storefront by using device-particular features such as a smart phone's contact list. In a native application, all elements are coded by using the mobile operating system's Software Development Kit (SDK). Native starter store is available for Android Operating System only (2.2 and newer versions with Google APIs).

**Hybrid mobile applications**

The hybrid application delivers a native experience by wrapping the mobile web storefront with a native application. The native elements are coded using the mobile operating system's SDK, while the storefront is accessed using the mobile web interface. Hybrid starter store is available for Android Operating System (2.2 and newer versions with Google APIs) and iOS (version 4 and 5).

## 2.6 Barcode specification

Barcode is an optical representation of data that can be read by optical scanners (e.g. laser scanners or camera-based readers). Those codes contain data that describes objects to which they are attached. There are two main types of barcodes: one-dimensional and two-dimensional (see FIGURE 7).

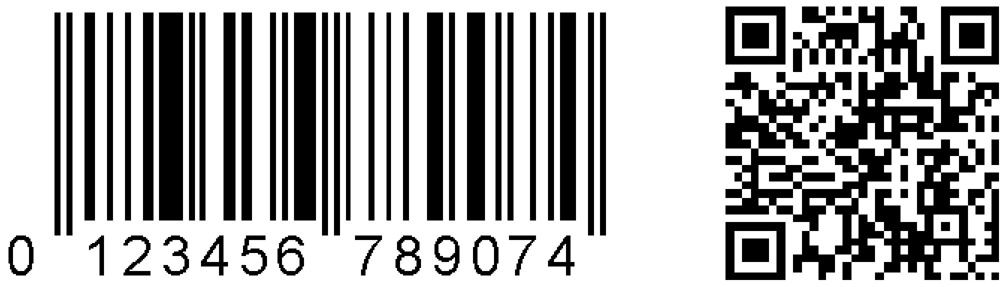


FIGURE 7. Barcode examples

One-dimensional barcodes are a combination of light and dark elements that are specified by symbolic of quantified code. Most of one-dimensional barcodes can contain only numerical data.

QR barcodes are built from squares that can be light or dark. Those codes can contain alphanumerical data.

The objective of this thesis is to implement a barcode scanner feature to the mobile store. To archive that goal the application uses a camera- based code scanner. First, the device takes a picture of a barcode, and then performs image-processing to decode the data that the barcode contains. The application can handle both types of barcodes. Depending on which code is used, the application performs a different kind of operation. The task of this thesis was not to implement an application to scan barcodes but to customize store presentation and business logic to use the information obtained from these barcodes.

## 2.7 Auxiliary libraries

### 2.7.1 ZXing

ZXing is an open source library for barcode image processing written in Java. This library is included in hybrid mobile store application. ZXing supports many barcode formats, both one-dimensional and two-dimensional. It offers a wide functionality, from scanning barcodes to generating them. To start using this library in Java application, it must be built first with *ant* command line tool (ZXing, 2011).

ZXing originally was implemented in Java, however, there are ports to other languages such as C++, Objective C, Ruby and partial C# and ActionScript.

### 2.7.2 Apache POI

Apache POI is a library, which provides API for many Microsoft documents. POI is able to open and handle Excel, Word, PowerPoint, OpenXML4J, OLE2, Outlook, Visio and Publisher documents. This library offers developers simple API to read and write Microsoft documents in Java (The Apache POI Project, 2011).

### 2.7.3 Simple JSON

Simple JSON is lightweight library, which s implements JSON format for Java. It provides a wide spectrum of functions such as encoding, decoding and escaping JSON text. Simple JSON is independent of any external libraries. It is fully compatible with JSON specification (JSON Simple, 2011).

# 3 DEVELOPING ECOMMERCE APPLICATION

The main objective of this thesis was to develop a hybrid mobile application for Android Operating System with search engine based on barcodes and QR codes. Despite of providing by IBM extensive sample store, there are still many efforts to be taken by developer to create high-end, customized product for customer.

QR codes are easy-to-use two-dimensional barcodes. Those codes can be scanned from a photo taken with a camera built in a smart phone or tablet.

## 3.1 Deploy starter mobile store

WebSphere Commerce with Feature Pack 4 provides support for mobile applications. To use mobile starter store provided by IBM WebSphere Commerce it is obligatory to publish working store first and share its data assets by publishing a mobile starter store. Both native and hybrid types of mobile applications require installed *Madison's Enhancements* and *Mobile Madison's Enhancements* to work correctly. The Figure 8 shows the proper installation order of enhancements.

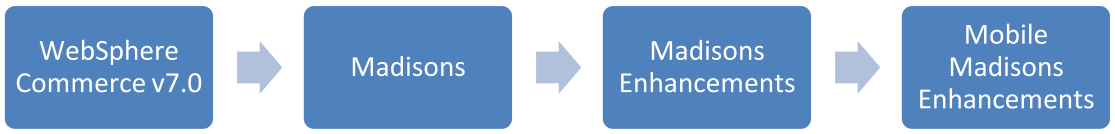


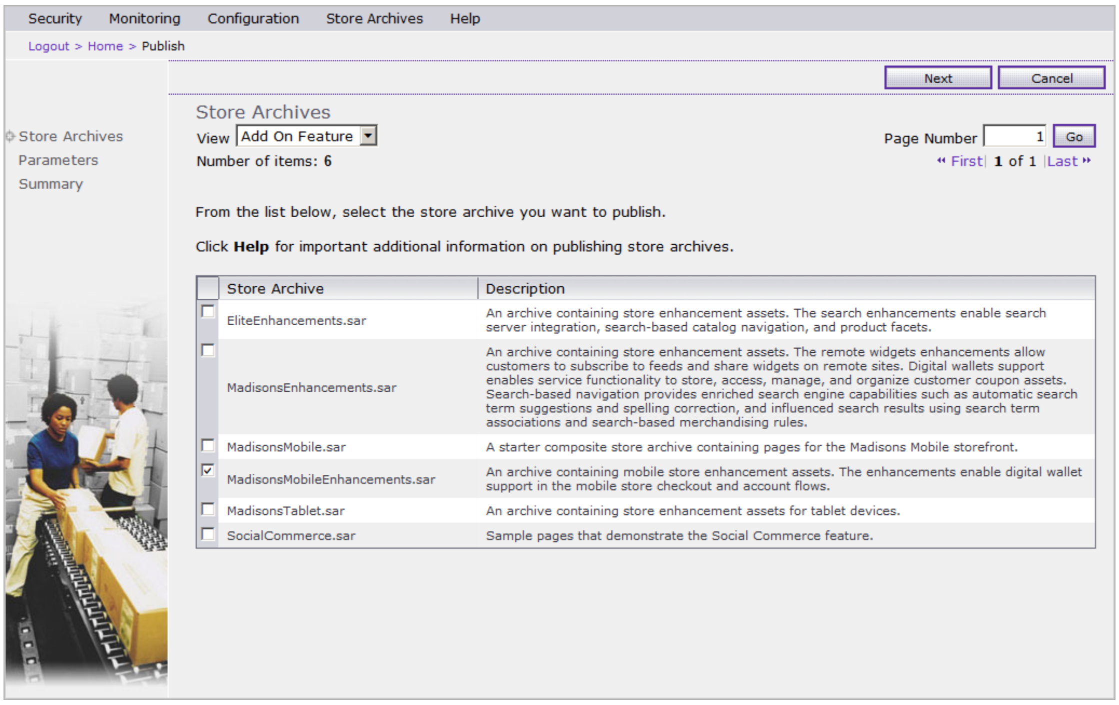
FIGURE 8. Dependency of enhancements

It is highly recommended to check dependency of enhancements before publishing them. To determine dependencies between packages it is necessary to check *IBM Info Center* page and enhancements documentation files.

IBM provides for store administrators WebSphere Administration Console, which allows administrators to control and configure store.

WebSphere Administration Console has a special wizard tool for installing enhancements. The publish wizard tool is used to install various store enhancements (including organization structures, user roles, access control policies, store assets). Enhancements are made as packages. Those packages are compressed to *Servlet ARchive* file.

To install new enhancements simply publish SAR file with WebSphere wizard tool (see FIGURE 9). After successful publishing archiving, it is necessary to restart WebSphere Application Server.



Store Archives

View: Add On Feature

Number of items: 6

Page Number: 1

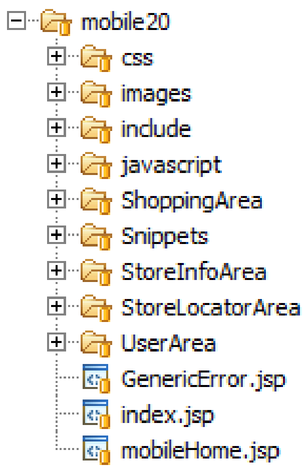
From the list below, select the store archive you want to publish.

Click **Help** for important additional information on publishing store archives.

| | Store Archive | Description |
|---|---|---|
| ☐ | EliteEnhancements.sar | An archive containing store enhancement assets. The search enhancements enable search server integration, search-based catalog navigation, and product facets. |
| ☐ | MadisonsEnhancements.sar | An archive containing store enhancement assets. The remote widgets enhancements allow customers to subscribe to feeds and share widgets on remote sites. Digital wallets support enables service functionality to store, access, manage, and organize customer coupon assets. Search-based navigation provides enriched search engine capabilities such as automatic search term suggestions and spelling correction, and influenced search results using search term associations and search-based merchandising rules. |
| ☐ | MadisonsMobile.sar | A starter composite store archive containing pages for the Madisons Mobile storefront. |
| ✓ | MadisonsMobileEnhancements.sar | An archive containing mobile store enhancement assets. The enhancements enable digital wallet support in the mobile store checkout and account flows. |
| ☐ | MadisonsTablet.sar | An archive containing store enhancement assets for tablet devices. |
| ☐ | SocialCommerce.sar | Sample pages that demonstrate the Social Commerce feature. |

FIGURE 9. WebSphere Administration Console - Store Archives

The mobile store shares database and business logic with an already working store. The publish process installs the dedicated JSP files into */mobile20* directory (see FIGURE 10). The catalogue structure is the same as a catalogue structure of a web store. The mobile store should not use any presentation files of web store.



```
mobile20
  css
  images
  include
  javascript
  ShoppingArea
  Snippets
  StoreInfoArea
  StoreLocatorArea
  UserArea
  GenericError.jsp
  index.jsp
  mobileHome.jsp
```

**FIGURE 10. Catalogue structure**

It is highly recommended to optimize JSP pages for mobile devices (e.g. reduce size of images, compress files). One of the methods to optimize the size of JSP pages is enabling the *useCDataTrim* JSP compiler flag. This compiler flag is used to reduce whitespaces in JSP files. It is also possible to compress files with *mod_deflate* module of Apache HTTP Server.

The mobile store files include a ***mobile*** prefix to separate them from the starter store files of the same name. Configuration ids contain ***m20*** prefix. This naming convention should be used in the whole process of developing a store application.

Installation updates also *struts-config-ext.xml* configuration file and accesses control policies. Below is shown a code snippet from Struts configuration file.

```
<forward className="com.ibm.commerce.struts.ECActionForward"
name="m20StoreView/10701" path="/mobile20/mobileHome.jsp">forward>

<forward className="com.ibm.commerce.struts.ECActionForward" name="m20Index/10701"
path="/mobile20/mobileHome.jsp"></forward>

<forward className="com.ibm.commerce.struts.ECActionForward"
name="m20LocationCheckIn/10701"
path="/mobile20/UserArea/AccountSection/ServiceSection/LocationCheckInSubsection/C
heckInLocation.jsp"></forward>

<action path="/m20StoreView" type="com.ibm.commerce.struts.BaseAction">
      <set-property property="credentialsAccepted" value="0:P,0:P"/>
</action>

<action path="/m20Index" type="com.ibm.commerce.struts.BaseAction">
      <set-property property="credentialsAccepted" value="0:P,0:P"/>
</action>

<action path="/m20LocationCheckIn" type="com.ibm.commerce.struts.BaseAction">
      <set-property property="https" value="0:1,0:1,0:1"/>
      <set-property property="credentialsAccepted" value="0:P,0:P,0:P"/>
</action>
```

WebSphere provides starter native and hybrid mobile store applications for Android platform. Those applications are provide as Android projects packed to zip files. To start developing a mobile application simply imports a project to the integrated development environment, which supports Android SDK. The author of this thesis used Eclipse SDK version 3.7.2.

## 3.2 Setting test environment

Native and Hybrid applications use features that are built in Android Operating System. Android SDK provides an emulator platform for testing purposes; however, this emulator has many limitations.

The emulator does not support:

- Phone calls
- USB connections
- Camera/video capture
- Handling headphones
- Controlling battery charge level
- Handling SD card
- Bluetooth

Those limitations can be resolved by using a dedicated mobile device with Android operating system. However, to access *localhost* server through mobile device it is necessary to connect with server through VPN. This thesis assumes that the reader has a working VPN server and can connect to it with a smart phone or tablet.

QR codes scanner requires access to the device's camera. The emulator does not support capturing images; therefore, it is obligatory to use a real device.

*Madison* mobile starter store contains its preferences in *systems_settings.xml* file. To run a store on device it is necessary to set up those preferences. Below is a code snippet of *system_settings.xml* file with defined preferences.

```
<string name="storeName">Madisons</string>
<string name="hostName">192.168.32.50</string>
<string name="storeId">10051</string>
<string name="storeIdentifier"></string>
<string name="catalogId">10051</string>
<string name="map_api_key"></string>
<string name="languageId">-1</string>
```

## 3.3 Android hybrid mobile application

### 3.3.1 Implementation of barcode activity

IBM mobile starter store implements a basic implementation of a barcode scanner. Code scanning is handled by a third-part application installed on a mobile device by the end-user. First, the user chooses the barcode scanner from the menu. After scanning is finished the intent is sent to the barcode activity. The intent contains data from a scanned barcode. Based on this data the application performs certain actions and generates a specified request for the web application. The business logic handles this request and prepares a JSP page. When the page is ready, it is sent back to the mobile device. The Figure 11 illustrates the whole process.

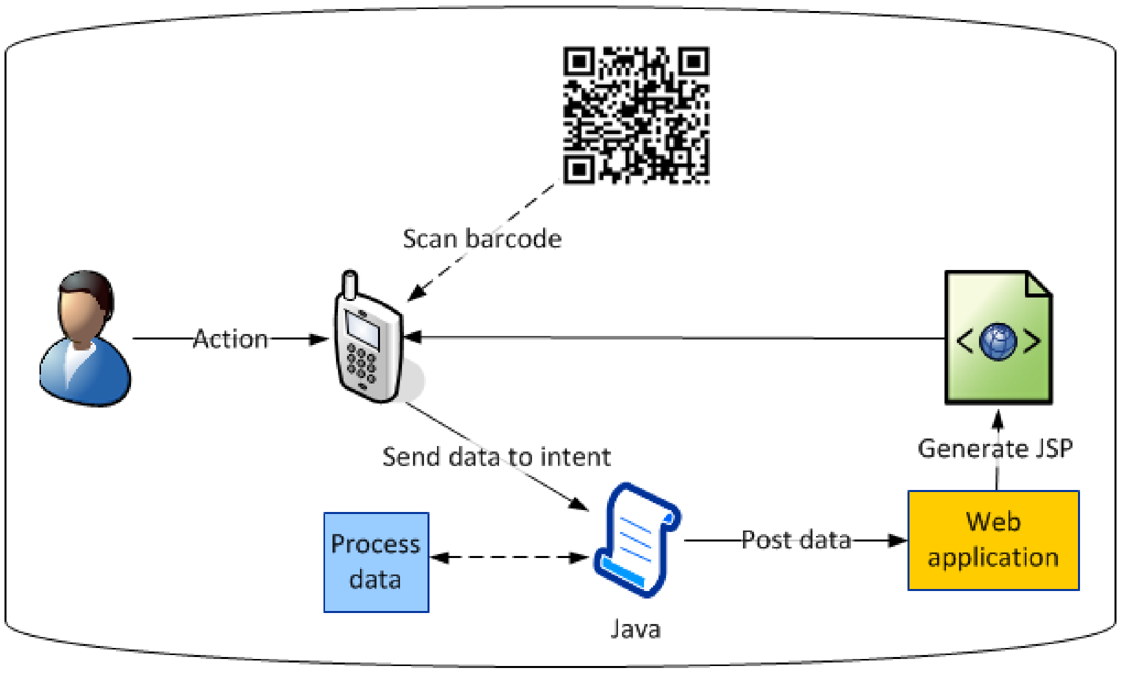


FIGURE 11. Workflow of a barcode scanning

The main purpose of the barcode activity is to show results as a JSP page. Hybrid application provides a special view called *StoreWebViewFlipper*. The snippet below shows a code of Android *StoreWebViewFlipper* view.

```
<com.ibm.commerce.android.hybrid.mobile.web.StoreWebViewFlipper
         android:id="@+id/barcodeSearchWebViewFlipper"
         android:layout_height="wrap_content"
         android:layout_width="match_parent"
         android:layout_weight="1.0" />
```

*StoreWebViewFlipper* is basically an extended Android's *ViewFlipper*, which is designed for displaying web pages. To take advantage of using *StoreWebViewFlipper* view the activity class has to inherit the following classes and implement one interface:

- Abstract class AbstractStoreWebViewActivity
- Interface IStoreWebViewActivity
- Abstract class AbstractHybridMobileActivity.

Because Java does not support multi-inheriting, developers have to inherit only from *AbstractStoreWebViewActivity.* This abstract class implements *IStoreWebViewActivity* interface and inherent from *AbstractHybridMobileActivity.*

*IStoreWebViewActivity* interface contains following abstract methods:

- `public StoreWebViewController getStoreWebViewController();`
- `public String getDefaultUrl();`
- `public int getContentViewId();`
- `public int getStoreWebViewFlipperId();`

Method *getStoreWebViewController()* is implemented in *AbstractStoreWebViewActivity* and there is no need to override this implementation in *BarcodeActivity.* The remaining methods are implemented in *BarcodeActivity*.

The snippet below shows the implementation of those methods.

```
	@Override
	public String getDefaultUrl() {
		return
((AndroidHybridMobileApplication)getApplication()).getHomePageUrl().toString();
	}

	@Override
	public int getContentViewId() {
		return R.layout.barcode;
	}

	@Override
	public int getStoreWebViewFlipperId() {
		return R.id.barcodeSearchWebViewFlipper;
	}

	@Override
	public StoreWebViewController getStoreWebViewController() {
		return iWebViewController;
	}
```

*AbstractHybridMobileActivity* has one very important abstract method called *initialize()*. The implementation of this method is in *AbstractStoreWebViewActivity*. The main purpose of this method is to load web content to StoreWebViewFlipper by executing *loadWebContent()* method.

*BarcodeActivity* also implements *initialize()* method expanding its function by preparing the layout views, setting action listeners and handling gestures. It is also executing *initialize()* method from superior class.

To complete the process of adding a new activity it is necessary to add an annotation about a new activity in *AndroidManifest.xml* file.

### 3.3.2 Processing data from barcode

When a barcode is scanned, the intent with data returns to the activity (see FIGURE 11). *AbstractHybridMobileActivity* contains *onActivityResult()* method. This method handles all intents coming back to the activities. The method checks if the intent action name is equal to SCAN_INTENT constant. Then it checks the result code of the intent. If the action name indicates that the returned intent is SCAN_INTENT and the result code is equal -1 (RESULT_OK constant), the method processes the data and sends it the *BarcodeActivity*.

Depending on the scanned barcode format, different actions are taken. Basic one-dimensional barcodes contain only a product number. Based on this number a web address is built and passed to the *BarcodeActivity*. Map with URL's parameters is populated with store, catalogue and language identifiers. At the end content is put to the map. When all data is placed, the map is passed to the static method *StoreWebUtils.buildUrl()*. This method based on passed parameters returns the URL to the search JSP page. Finally, a new activity with additional data is launched. The main purpose of this activity is to show user search results. The snippet below shows the process of handling one-dimensional barcode.

```
Intent outGoingIntent = new Intent();

Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(StoreWebConstants.STORE_ID, getString(R.string.storeId));
queryParameters.put(StoreWebConstants.CATALOG_ID, getString(R.string.catalogId));
queryParameters.put(StoreWebConstants.LANGUAGE_ID,
                                       getString(R.string.languageId));
queryParameters.put("sku", contents);

String path  = getString(R.string.search_barcode_url_path);
Uri httpUrl = StoreWebUtils.buildUrl(lastHostName, path, queryParameters, true);

outGoingIntent.setClass(this, BarcodeActivity.class);
outGoingIntent.setData(httpUrl);
startActivity(outGoingIntent)
```

QR codes contain simple JSON object. The object represented by JSON has at least two fields: action and content. The action field contains the identifier of the action type.

An action field may take following values:

- URL,
- SEARCH,
- PROMOBUY.

The content field contains main data retrieved from barcode (e.g. web address for URL action type). It is possible that an object has more fields. It depends on the action type field (e.g. for PROMOBUY action type, object will contain an additional field called *productId*).

When a two-dimensional barcode is scanned, the content is converted to JSON object. Following actions are dependent on the content of *action* field. The snippet below shows an implementation of this process.

```
if (format.equals("QR_CODE")) {
	try {
		JSONObject jObject = new JSONObject(contents);

		String action = jObject.getString("action");
		String content = jObject.getString("content");

		// Handling data code was omitted


	} catch (JSONException e) {
		Toast.makeText(this, R.string.ERROR_READ_QR_CODE,
Toast.LENGTH_LONG).show();
		e.printStackTrace();
	}

}
```

URL action redirects the mobile device user to the web address contained in the *content* field. If the address is in the same domain as a mobile store, a web page opens in the store application. However, if web address points to a web outside the shop domain, the page is opened in Android web browser. The snippet below shows the implementation of handling URL action.

```
if("url".equalsIgnoreCase(action)){
	if (content != null && StoreWebUtils.isHttpUrl(content)) {
		Intent outGoingIntent = new Intent();
		Uri httpUrl = Uri.parse(content);

		if (StoreWebUtils.isUnderDomain(content)) {
			outGoingIntent.setClass(this, SearchResultActivity.class);
			outGoingIntent.setData(httpUrl);
		} else {
			outGoingIntent.setAction(Intent.ACTION_VIEW);
			outGoingIntent.setData(httpUrl);
		}

		startActivity(outGoingIntent);
	}
}
```

SEARCH action performs searching for provided search term in the *content* field of JSON object. The snippet below shows the implementation of handling SEARCH action.

```
else if("search".equalsIgnoreCase(action)){

	Intent outGoingIntent = new Intent();

	Map<String, String> queryParameters = new HashMap<String, String>();
	queryParameters.put("searchTerm", content);

	String path= getString(R.string.search_category_url_path);
	Uri httpUrl = StoreWebUtils.buildUrl(lastHostName, path, queryParameters,
true);

	outGoingIntent.setClass(this, BarcodeActivity.class);
	outGoingIntent.setData(httpUrl);
	startActivity(outGoingIntent);

}
```

PROMOBUY action is responsible for preparing a special web address for ordering a product with special promotion. A product is unique identified with the *productId,* which is contained in the *content* field of JSON object. An additional field is passed through JSON object - *customPromotionId.* This identifier represents the promotion code added with the *Management Center*. A snippet code is responsible for handling PROMOBUY action is shown below as follows:

```
else if("promoBuy".equalsIgnoreCase(action)){

        String promoId = jObject.getString("promo_id");

        Intent outGoingIntent = new Intent();

        Map<String, String> queryParameters = new HashMap<String, String>();
        queryParameters.put("productId", content);
        queryParameters.put("customPromotionId", promoId);
        queryParameters.put("scan2buy", "true");

        String path= getString(R.string.product_url_path);
        Uri httpUrl = StoreWebUtils.buildUrl(lastHostName, path, queryParameters,
true);

        outGoingIntent.setClass(this, BarcodeActivity.class);
        outGoingIntent.setData(httpUrl);
        startActivity(outGoingIntent);

}
```

## 3.4 Web application

### 3.4.1 Struts configuration

IBM WebSphere Commerce uses Struts framework to help developers with creating web applications. Struts configuration file is located in *Stores/WebContent/WEB-INF/struts-config-ext.xml* file.

To add a new JSP page it is necessary to configure *actions* and *forwards* for it in the Struts configuration file. The snippet code below shows the configuration for the *BarcodeSearchResultDisplay* JSP page.

```
<forward className="com.ibm.commerce.struts.ECActionForward"
      name="m20BarcodeSearchResultView/10051"
      path="/mobile20/ShoppingArea/BarcodeSearchResultDisplay.jsp">
</forward>

<action path="/m20BarcodeSearchResultView"
      type="com.ibm.commerce.struts.BaseAction">
      <set-property property="credentialsAccepted" value="0:P,0:P"/>
</action>
```

After adding new records to the Struts configuration file, it is essential to refresh the configuration registry through the Administration Console. A default WebSphere Commerce configuration allows only administrators to view a new JSP pages. In order to change this, a new access policy should be added. IBM provides a special tool for changing access policies called *acpload*. This is a console batch script, which takes XML file as a parameter. The XML code below represents new policies added to the configuration.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
<Policies>
      <Action Name="m20BarcodeSearchResultView"
             CommandName="m20BarcodeSearchResultView">
      </Action>
      <ActionGroup Name="AllSiteUsersViews" OwnerID="RootOrganization">
            <ActionGroupAction Name="m20BarcodeSearchResultView"/>
      </ActionGroup>
</Policies>
```

Before executing this command, it is highly recommended to stop the server. If *acpload* does not return any exceptions, the server can be started. If any error occurs, utility will generate a log file in *logs/acpload.log.*

### 3.4.2 Customizing JSP search page

A one-dimensional barcode contains only a product number. Based on this number the search engine tries to find a product. First, it seeks only for an equal number. If there are no results, it searches one more time, this time less strictly. During the second search, the searching is performed with *LIKE* operator instead of *EQUAL*.

At default settings, the searching data bean looks for products, items, packages and bundles. Setting the corresponding values in the URL address can customize those options.

To search a database for matching products *CatEntrySearchListDataBean* is used. This bean is provided by IBM WebSphere and is used to retrieve information about a product from a database based on provided searching parameters.

WebSphere Commerce provides two searching options for different usage cases. Those searching formats are based on the catalogue search data bean. Catalogue search offers basic and advanced search functionality. A comparison of those two methods is shown in the Table 2.

TABLE 2. Comparison of search methods

| Simple search | Advanced search |
|---|---|
| **A lightweight search feature** | General searching feature |
| **Can be used on any JSP page** | Can be used on any JSP page |
| **Search based on single term** | Search based on multiple terms |
| **Searching executed on base tables** | Searching executed on base tables |
| **Support Boolean values** | Support Boolean values |
| | Advanced sorting criteria available |

*CatEntrySearchListDataBean* is a data bean, which is a Java bean the main role of which is to provide access to data from JSP pages. This data bean is an implementation of abstract class *SearchBaseDataBean*. The code below shows a part of JSP page implementation for one-dimensional barcode search with *CatEntrySearchListDataBean* data bean.

```
<wcbase:useBean id="catEntSearchListBean" scope="page"
classname="com.ibm.commerce.search.beans.CatEntrySearchListDataBean">
      <%-- Some less important code was omitted --%>

      <%-- Set the SKU number as a search term --%>
      <c:set property="sku" value="${WCParam.sku}"
             target="${catEntSearchListBean}" />
      <c:set property="skuCaseSensitive" value="no"
             target="${catEntSearchListBean}" />
      <c:set property="skuOperator" value="EQUAL"
             target="${catEntSearchListBean}" />
</wcbase:useBean>

<c:if test="${catEntSearchListBean.resultCount == 0}">

      <%-- remove old result bean. Need new one --%>
      <c:remove var="catEntSearchListBean" />

      <wcbase:useBean id="catEntSearchListBean" scope="page"
classname="com.ibm.commerce.search.beans.CatEntrySearchListDataBean">
            <%-- Some less important code was omitted --%>

            <%-- Set the SKU number as a search term --%>
            <c:set property="sku" value="${WCParam.sku}"
                   target="${catEntSearchListBean}" />
            <c:set property="skuCaseSensitive" value="no"
                   target="${catEntSearchListBean}" />
            <%-- We don't have any results, make query less strict --%>
            <c:set property="skuOperator" value="LIKE"
                   target="${catEntSearchListBean}" />
      </wcbase:useBean>
</c:if>
```

Data bean performs searching for matching rows in database. It can return zero, one or many rows. If it returns zero rows, it tries once again with a wider spectrum by searching for matching rows with LIKE operator instead of EQUAL.

When only one row is returned to the JSP page, automatic redirection to the product page is performed. First, catalogue entry is retrieved from the result list. Then it is checked what type it is (product, item, bundle or package). Finally, the URL address is generated and passed to JavaScript redirection script.

If the searching data bean returns many products, the JSP page lists all of them as a list of products.

### 3.4.3 Customizing JSP order page

A two-dimensional barcode contains an identifier of action to perform and content. QR codes contain object converted to JSON format. This object contains the necessary information to execute custom tasks. One of those tasks can be ordering an item with promotion codes.

IBM WebSphere supports extend system of campaigns and promotions. The administrator can decide on what conditions customers get an order discount, shipping discount or a free gift. Promotions can concern every client or only clients specified by some criteria. It is possible to set public promotions for every customer or narrow it only for regular clients. Those promotions can be managed through *IBM Management Center.*

A mobile store user who had scanned QR code that contains order action receives additionally access to promotion code. This promotion code is only for customers who order product with mobile store application. The promotion applies automatically without user knowledge. Clients can remove this promotion if they want to.

The *ProductDisplay* JSP page contains a form with the action *OrderChangeServiceItemAdd* that is responsible for adding a product to the shopping cart. The Android application generates a special URL address, which contains *scan2buy* parameter.

This parameter indicates that an item should be added to the shopping cart. Based on this parameter JSP page adds to the form an extra input and after the page is loaded JavaScript script submits a form. The code below shows a customized form and JavaScript script.

```
<form id="AddToCartForm" method="post" action="OrderChangeServiceItemAdd">
     <input type="hidden" name="catEntryId" value="${product.productID}" />
     <input type="hidden" name="catalogId" value="${WCParam.catalogId}" />
     <input type="hidden" name="storeId" value="${WCParam.storeId}" />
     <input type="hidden" name="URL" value="${OrderItemDisplayURL}" />
     <input type="hidden" name="langId" value="${langId}" />
     <input type="hidden" name="quantity_1" value="1" />
     <input type="hidden" name="productId" value="${product.productID}" />
     <input type="hidden" name="errorViewName"
            value="m20ProductDisplayView" />

     <c:if test="${!empty(WCParam.scan2buy)}">
          <input type="hidden" name="customPromoCode"
                  value="${WCParam.customPromoCode}" />
     </c:if>

</form>

<script type="text/javascript">
//<![CDATA[
      <c:if test="${!empty(WCParam.customPromoCode)}">
            submitAddToCart();
      </c:if>
//]]>
</script>
```

The *OrderItemDisplay* JSP page handles the order process. If the user wants to order a product, it has to be added to the shopping cart first. Users can add products to the shopping cart from the product page - the *ProductDisplay* JSP page.

The action *OrderChangeServiceItemAdd* is a service defined in the Struts configuration file. This service is one of many order services available through Struts and can be executed like calling URL commands. Every Struts action name is mapped to a corresponding façade client method. Those methods are implemented in *OrderFacadeClient*.

After submitting the form, users are redirected to their shopping cart, where the promotion is applied to the order. The whole process is automatic and users only see the shopping cart page.

### 3.4.4 Customizing JSP shopping cart page

After submitting a form from the *ProductDisplay* JSP page users are automatically redirected to the *OrderItemDisplay* page. The promotion code is retrieved from the *WCParam.* It is a map that contains pair of the name of the request parameter and a corresponding single string value.

If a promotion code is mapped in the *WCParam* map, a checking process is performed. All active promotion codes are retrieved through *PromoCodeListDataBean*. That data bean retrieves a list of all codes associated to the current order. If in the promotion codes list there is already a code that was retrieved from *WCParam*, nothing happens. However, if it is a unique code, an extra input field is generated. This input contains the promotion code as a value. After the page is successful generated with the extra input, the shopping cart is updated.

*PromoCodeListDataBean* is a data bean that can populate itself. It is not necessary to provide another data bean for it. This bean returns a list of all promotion codes connected with the order passed as a parameter. It is obligatory to set the order identifier before populating the list. The figure below shows the essential code responsible for applying the promotion code to the order.

```
<wcbase:useBean id="promoCodeListBean"
classname="com.ibm.commerce.marketing.databeans.PromoCodeListDataBean"
scope="page">
      <c:set property="orderId"
             value="${order.orderIdentifier.uniqueID}"
             target="${promoCodeListBean}" />
</wcbase:useBean>

<c:forEach items="${promoCodeListBean.codes}"
           var="currentCode" varStatus="status">

      <c:if test="${currentCode.code == WCParam.customPromoCode}">
            <c:set var="duplicateCustomPromoCode" value="true" />
      </c:if>

</c:forEach>

<c:choose>
      <c:when test="${!empty(WCParam.customPromoCode)
                  && duplicateCustomPromoCode != true }">
            <input value="<c:out value="${WCParam.customPromoCode}" />"
type="hidden" name="promotion_code" />
      </c:when>
      <c:otherwise>
            <input type="text" name="promotion_code" />
      </c:otherwise>
</c:choose>


<c:if test="${!empty(WCParam.customPromoCode)
            && duplicateCustomPromoCode != true}">
      <script type="text/javascript">
            //<![CDATA[
                  updateShoppingCart(document.ShopCartForm);
            //]]>
      </script>
</c:if>
```

## 3.5 Barcode Generator

The application itself for reading barcodes is insufficient to provide a customer with a full-featured product. To make the barcode scanner feature useful it is important to deliver a barcode generator as well. This generator must be simple to use for a customer.

The client provides data for barcode generator as an Excel sheet. This sheet can be used in many others scripts, that is why it does not contain only data for the barcode generator. Therefore, the barcode generator must be flexible.

The generator is created using Java language and can be executed on any machine with installed Java Virtual Machine. The generator works as a console application. To generate barcodes simple execute *BarcodeGenerator* (see

```
root$ java -jar BarcodeGenerator.jar
Generate barcodes from XLS file

usage: BarcodeGenerator [ options ] file.xls
  --format=format: Barcode format. Code39 or QRCode. Default: QRcode.
  --column=number: Column of xls file, where content is. Start counting
from 0. Default: 3
  --actionColumnRows=number: Column of xls file, where action type is.
Start counting from 0. Default: --column + 1
  --width=pixels: Image width. Default: 400
  --height=pixels: Image height. Default: 300
```

FIGURE 12) in console command.

```
root$ java -jar BarcodeGenerator.jar
Generate barcodes from XLS file

usage: BarcodeGenerator [ options ] file.xls
  --format=format: Barcode format. Code39 or QRCode. Default: QRcode.
  --column=number: Column of xls file, where content is. Start counting
from 0. Default: 3
  --actionColumnRows=number: Column of xls file, where action type is.
Start counting from 0. Default: --column + 1
  --width=pixels: Image width. Default: 400
  --height=pixels: Image height. Default: 300
```

**FIGURE 12. Executing BarcodeGenerator in console**

Barcode generator is packed with necessary libraries as a JAR file. This archive includes the following libraries:

- ZXing 2.0 library,
- Apache POI 3.7 library,
- Simple JSON 1.1.1 library.

Barcode generator contains two classes: *BarcodeGenerator* and *DataHandler*. *DataHandler* implements *IDataHandler* interface.

BarcodeGenerator class contains the main method. In this method, parameters are set. The snippet below shows the process of setting parameters in *BarcodeGenerator* script.

```
for (String arg : args) {
      if (arg.startsWith("--format")) {
            if (arg.split("=")[1] == "code39") {
                  dataType = BarcodeFormat.CODE_39;
            } else {
                  dataType = BarcodeFormat.QR_CODE;
            }
      } else if (arg.startsWith("--column")) {
            rowNumber = Integer.parseInt(arg.split("=")[1]);
      } else if (arg.startsWith("--width")) {
            width = Integer.parseInt(arg.split("=")[1]);
      } else if (arg.startsWith("--height")) {
            height = Integer.parseInt(arg.split("=")[1]);
      } else if (arg.startsWith("--actionColumnNumber")) {
            actionRowNumber = Integer.parseInt(arg.split("=")[1]);
      }
}
```

When all parameters are set and no exception occurs, the application starts reading the Excel sheet. It is possible that this file contains a great deal of information irrelevant to the barcode generating task. However, when the application is executed it is possible to point columns that should be retrieved from sheet. The Table 3 represents sample source data file.

TABLE 3. Sample data source file for barcode generator

| | | | |
|---|---|---|---|
| **Electrolux ASM450-kulhovatkain** | 242158 | SC1203E | 10,5 |
| **Nikon D5100 -digitaalinen** | 142958 | SC1202E | 11,0 |
| **Peak Performance Takki** | 282957 | SC1201E | 27,00 |
| **Hugo Boss Black Solmio** | 341953 | SC1203D | 1,6 |

Every retrieved data row is passed to *DataHandler*, where it is handled. Depending on what type of barcode is generated, data is converted to String or JSON String. The snippet below shows sending data to *DataHandler* and converting implementation.

```
if (dataType == BarcodeFormat.QR_CODE) {
      Cell actionCell = row.getCell(actionRowNumber,
      Row.CREATE_NULL_AS_BLANK);
      dataHandler.setData(cell.toString(), actionCell.toString());
} else {
      dataHandler.setData(cell.toString());
}

data = dataHandler.getData();

@Override
public String getData() {
      return data;
}

@Override
public void setData(String data, String action) {
      params.put(DataHandler.CONTENT_FIELD, data);
      params.put(DataHandler.ACTION_FIELD, action);

      JSONObject jObject = new JSONObject(params);
      setData(jObject.toJSONString());
}
```

When data is parsed and ready to be converted to barcode, image processing is executed. This process is handled by ZXing library. The application creates *MultiFormatWriter* object and *BitMatrix* object.

*MultiFormatWriter* has *encode()* method which is responsible for converting data from parameter to the barcode of passed type. It also sets the dimensions of the barcode. This method returns object of *BitMatrix* type.

To save barcode as a file, simply pass returned *BitMatrix* object to the *MatrixToImageWriter.writeToFile()* static method. Code below shows described process.

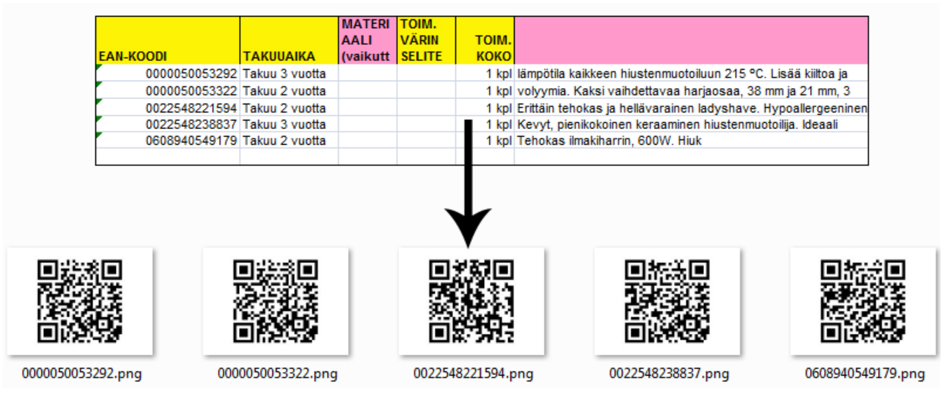
```
data = dataHandler.getData();

matrix = barcodeWriter.encode(data, dataType, width, height);
MatrixToImageWriter.writeToFile(matrix, "png", new
File(cell.toString() + ".png"));
```

Barcodes are generated in a folder where barcode generator was executed. Filenames are determined according to the corresponding cell text.

## 3.6 Testing working application

The main objective of this thesis was to develop a hybrid mobile application for Android Operating System with search engine based on barcodes. However, the application itself for reading barcodes is insufficient to provide a client with a full-featured product. Before presenting ready product to a customer, it is necessary to test it.

Figure 13 shows generating QR codes based on data in XLS file.



| EAN-KOODI | TAKUUAIKA | MATERI AALI (vaikutt | TOIM. VÄRIN SELITE | TOIM. KOKO | |
|---|---|---|---|---|---|
| 0000050053292 | Takuu 3 vuotta | | | 1 kpl | lämpötila kaikkeen hiustenmuotoiluun 215 °C. Lisää kiiltoa ja |
| 0000050053322 | Takuu 2 vuotta | | | 1 kpl | volyymia. Kaksi vaihdettavaa harjaosaa, 38 mm ja 21 mm, 3 |
| 0022548221594 | Takuu 2 vuotta | | | 1 kpl | Erittäin tehokas ja hellävarainen ladyshave. Hypoallergeeninen |
| 0022548238837 | Takuu 3 vuotta | | | 1 kpl | Kevyt, pienikokoinen keraaminen hiustenmuotoilija. Ideaali |
| 0608940549179 | Takuu 2 vuotta | | | 1 kpl | Tehokas ilmakiharrin, 600W. Hiuk |

FIGURE 13. Generating barcodes from XLS file

The figures 14, 15, 16 and 17 show screenshots from a working mobile application.

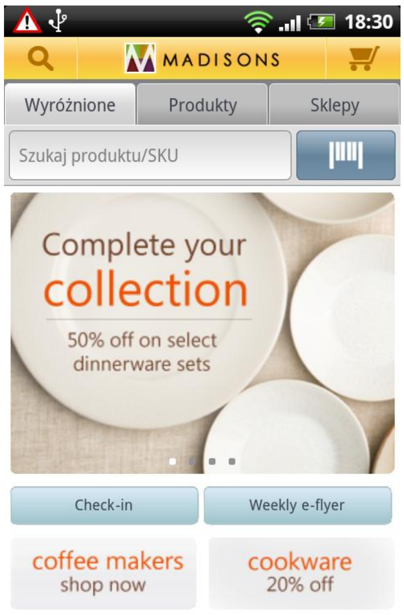
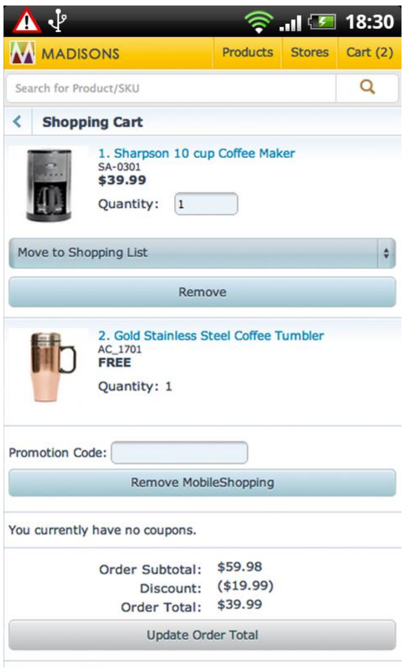


FIGURE 14. Main window of mobile application



Shopping Cart

1. Sharpson 10 cup Coffee Maker
SA-0301
**$39.99**
Quantity: 1

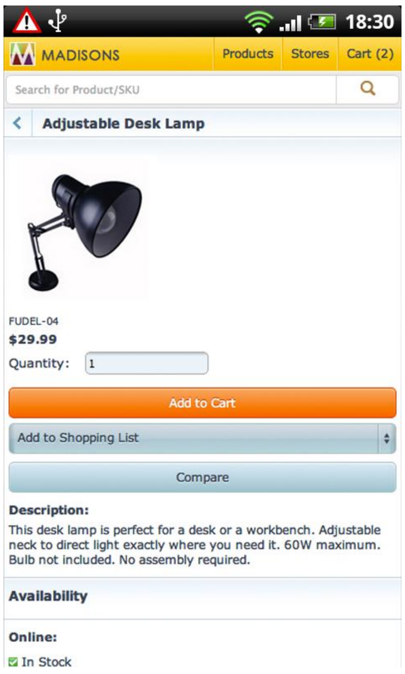
2. Gold Stainless Steel Coffee Tumbler
AC_1701
**FREE**
Quantity: 1

Promotion Code:

You currently have no coupons.

Order Subtotal: $59.98
Discount: ($19.99)
Order Total: $39.99

FIGURE 15. Shopping cart after scanning QR Code with promotion code



Adjustable Desk Lamp

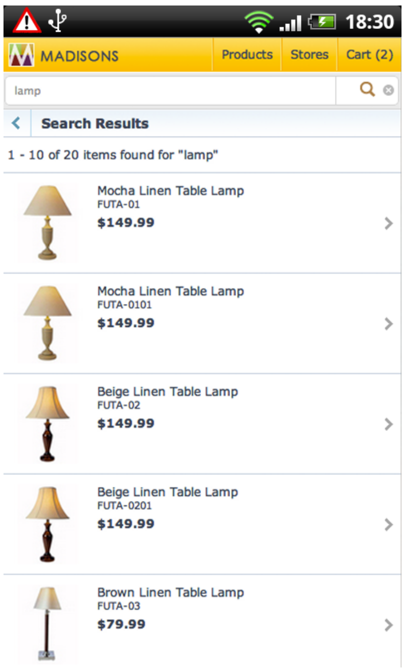
FUDEL-04
**$29.99**
Quantity: 1

**Description:**
This desk lamp is perfect for a desk or a workbench. Adjustable neck to direct light exactly where you need it. 60W maximum. Bulb not included. No assembly required.

**Availability**

**Online:**
✓ In Stock

FIGURE 16. Product page after scanning barcode



Search Results

1 - 10 of 20 items found for "lamp"

Mocha Linen Table Lamp
FUTA-01
**$149.99**

Mocha Linen Table Lamp
FUTA-0101
**$149.99**

Beige Linen Table Lamp
FUTA-02
**$149.99**

Beige Linen Table Lamp
FUTA-0201
**$149.99**

Brown Linen Table Lamp
FUTA-03
**$79.99**

FIGURE 17. Product list after scanning barcode

# 4 CONCLUSIONS

The main objective of this thesis was to develop a hybrid mobile store application for Android Operating System with search engine based on barcodes and QR codes. IBM provides an extensive starter store called *Madison*, which contains basic implementation of barcode scanner.  I planned to extend this feature with a full-featured and useful tool for both clients and store users to be an attractive marketing tool for improved sales and to store attractiveness for clients. For store users it would be a practical and easy to use application that would be helpful in every day shopping.

During the process of writing this thesis, I have managed to accomplish both goals. I extended the barcode scanner provided by IBM with new functionalities such as searching products based on barcode information, redirecting users to the page, performing custom search and ordering products with dedicated promotions.

Users can search for a product in online store by simple barcode scanning. It is possible for them to check for a product available in a web store, compare prices and take part in marketing campaigns.

Clients are able to perform interesting and modern marketing campaigns with the use of different kinds of barcodes. With multiple options to choose, they can plan and launch campaigns of a new type.

To start developing a mobile application it was necessary to deploy Feature Pack 4 for WebSphere Commerce, which provides support for mobile applications and installs the *Madison* mobile starter store. At the beginning, installing both *Madison's Enhancements* and *Mobile Madison's Enhancements* was straightforward and did not make any problems. However, the final step of installation those enhancements caused a lot of trouble. Unfortunately, in my humble option, IBM documentation about publishing enhancements does not contain a lot of information. It only provides basic knowledge about the whole process. The main problem was to determine the dependency between different packages. Not all dependencies are documented. Server console of Rational Application Developer became very useful in the search for solution of incompatible enhancements.

IBM provides an extensive mobile starter store, which does not require a great deal of configuration to run it. Improving this starter store is really straightforward and after getting basic knowledge about class structure, developing new features is relatively simple. On the other hand, I had to get that knowledge by reading code provided by IBM. There was no many useful information on the IBM Info Center website (e.g. tutorials about basic development). However, *Madison's* source code is written very well and it is also well documented. Therefore, it was not very complicated to create new activities on my own. What is very interesting, while I was developing my code under Android, I found a bug in an IBM class. It was not a critical error. Because of this error, I spent a few hours trying to get my activity to work. Even such a mature and advanced product as WebSphere Commerce has bugs. WebSphere Commerce API contains over 10000 interfaces and classes. The list of all classes with documentation can be found on IBM Info Center website. Despite the fact that IBM introduces name convention in they own documentation I found many confusing discrepancies. In my opinion, documentation of such a huge product should be very carefully created and checked for errors and misleading information.

IBM has implemented new views based on Android's views. They are specifically designed for displaying and handling eCommerce data. There are also many classes and interfaces ready to use in developing a mobile application. Those elements simplify the creation of an application.

IBM also provides a mobile version of *Madison* starter store. This web application uses the same business logic as a full version. Therefore, the only difference between the mobile application and its full edition is the presentation layer. By developing the mobile store, I familiarized myself with techniques of creating a useful mobile application and its limitations. I had an opportunity to test my application on a real devices. It was a very valuable experience that helps me to improve my application to be more user-friendly.

Customizing JSP pages responsible for searching products and making orders was straightforward. WebSphere Commerce has a good documentation of business logic. It is also very complex, and it was not necessary for me to expand its functionality. IBM already implemented everything I needed. Thanks to that, developing new applications is incredibly fast and simple.

When I was working on my thesis, I noticed that keeping the same name convention is really important. The naming convention that is suggested by IBM is clear and helpful. It increases the readability of the code. Another advantage is visible when we work in a team and we want to merge our work with others' work. If we keep our naming convention, merging of files is simple and in most cases proceeds without any conflicts.

I have now a much better understanding of WebSphere Commerce than before. I have learnt how to take advantage of provided beans, database and code. I have better knowledge about life cycle of a product. An extensive library of classes is very important in enterprise development. Enterprise applications have to be stable and fast. One of the ways to archive that goal is to use expanded and mature code. I think that, IBM WebSphere Commerce has very good, ready to use code. It also has a full-featured starter store, which can be used as a template to build a custom shop. It is not necessary to build everything from scratch.

Using WebSphere Commerce to built eCommerce applications reduces development time significantly. Creating new pages using JSP technology is straightforward. However, it is difficult to debug those pages. The debug tool provided by *Rational Application Developer* does not support debugging of JSP pages. It is strongly recommended to implement only presentation layer, not business layer in JSP pages. Therefore, debugging of those pages should not be necessary, because all implementation code should be in Java files.

Using open source libraries like *ZXing*, *Simple JSON* or *Apache POI* was also very helpful. Instead of developing a barcode scanner from scratch, I just used a ready and well-tested library. It not only reduces time spent on developing but also provides better security and performance. Many developers use those libraries, so probability of existing bugs in those codes is much less than in code written by one programmer. I could not afford to spend so much time on one task, which would be creating, testing and optimization of barcode scanner library.

The application is fully-featured and ready to use. However, it could be still improved with new features. The application could be connected with social media like *Facebook* or *Twitter* to add a possibility for sharing barcodes with friends. WebSphere Commerce provides a possibility for cooperative shopping. With the sharing barcodes feature, it would provide  great functionality for clients of a web store. Another feature to implement could be a management system for scanned barcodes. Users would have a possibility to use scanned barcodes in future, because not always is it possible to make order an when the barcode is scanned. WebSphere Commerce provides also wish lists. A very useful feature would be adding a product to wish lists by scanning barcodes.

# REFERENCES

Crawford, W., & Kaplan, J. (2003). *J2EE Design Patterns.* O'Reilly.

Fang, Y. (2011). *JSON Simple*. Referenced 2012-03-18 from JSON Simple: http://code.google.com/json-simple/

*IBM - WebSphere Commerce*. (2011). Referenced 2012-03-25 from IBM - WebSphere Commerce: http://www.ibm.com/software/genservers/commerceproductline

*IBM - WebSphere Commerce*. (2012). Referenced 2012-04-07 from IBM - WebSphere Commerce: http://www-01.ibm.com/software/genservers/commerceproductline/compare.html

*IBM Info Center*. (2011). Referenced 2012-04-08 from IBM Info Center - WebSphere Commerce Version 7.0.0.5 http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdapplication.htm

Siggelkow, B. (2005). *Jakarta Struts Cookbook.* O'Reilly.

*Struts 1*. (2008). Referenced 2012-02-19 from The Apache Software Foundation: http://struts.apache.org/

*The Apache POI Project*. (2011). Referenced 2011-03-17 from The Apache POI Project: http://poi.apache.org

*Three-tier*. (1998). Referenced 2012-02-19 from Three-tier definition from FOLDOC: http://foldoc.org/three-tier/

Ticknor, M., Corcoran, A., Csepregi-Horvath, B., Goering, A., Hernandez, J. P., Limodin, J., et al. (2011). *IBM WebSphere Application Server V8 Concepts, Planning, and Design Guide.*

Wahli, U., Cui, H., Fleming, C., Mehta, M., Rohr, M., Ugurlu, P., et al. (2007). *Rational Application Developer V7 Programming Guide.* IBM.

*ZXing*. (2011). Referenced 2011-03-18 from ZXing: http://code.google.com/zxing/