The Florida State University
College of Arts and Sciences

The Real Estate Listing Web Service Using IBM Websphere and DB2

by
Susmitha Athota

September 2004

A project submitted to the
Department of Computer Science
In partial fulfillment of the requirements for the
Degree of Master of Science

Major Professor: **Dr. Daniel G. Schwartz**

Table of Contents

Acknowledgements 3

Abstract 4

1. Introduction 5
   1.1 Requirements Specifications 5
   1.2 Design Specifications 6
   1.3 Developer Environment used 6
   1.4 Database (Back end) used 7

2. Code Development and Modules 8
   2.1 Servlets used in Project 8
   2.2 JSPs (Java Server Pages) used in Project 9
   2.3 Uploading files from InputStream 10

3. Java Beans 12
   3.1 Enterprise Java Beans 12
   3.2 CMP Entity Beans 12
   3.3 JNDI (Java Naming Directory and Interface) 13
   3.4 Layered Architecture 14
   3.5 EJB-QL (EJB Query Language) 17

4. Deployment Descriptors 19
   4.1 Application Deployment Descriptor 19
   4.1 Web Deployment Descriptor 19
   4.3 EJB Deployment Descriptor 21

5. Database and Websphere Test Environment (WTE) 23
   5.1 Database Table used in Project 23
   5.2 Websphere Test Environment 24

6. Flowcharts 25

7. On Running the Server 28

8. Web Services and Future Extensions to Project 33
   8.1 Web Services 33
   8.2 Future Extensions to Project 33

9. References 35

Acknowledgments

I have quite a few people to thank regarding the development of this project. Most importantly, I would like to acknowledge my major professor and advisor, Dr. Daniel G Schwartz for his guidance and idea about taking up this project. It has been wonderful working under his guidance and I thank him for his inspiration and encouragement which made this project possible.

I also extend my thanks to other professors and staff in the Computer Science Department, Florida State University for providing me with all the facilities required for this project. I express gratitude to my parents for their constant motivation and support, which helped me a lot during the duration of my master's degree at Florida State University.

Abstract:

The Real Estate Listing Web Service is a J2EE coded web application developed using IBM Websphere Application Studio Developer. Users will be able to search for properties according to their choice of location, price and property features, including number of bedrooms and baths. IBM DB2 is used as the backend for this project and Servlets and JSP pages are used in the front end.

Enterprise Java Beans (EJBs) are used to access the DB2 database in a convenient and easy way, by mapping the CMP (Container Managed Persistence) Entity beans to the database. Request and session beans help in collecting data from the Servlet methods and transferring the results to the JSP pages for display in a pleasing manner. Deployment Descriptors exist for the Enterprise Application, EJB and the Web application, which have a detailed representation of how data is accessed, the Servlets' and JSPs' information, the enterprise Java beans used, and the queries used for accessing data from the database. EJB-QL (EJB Query Language), which has SQL-like queries, is the language used with EJBs.

1. Introduction

Real Estate listing services list various properties available in a particular area and provide users that ability to search through them according to their location, price, and other features. For example, one might want a house in the Tallahassee area, in the price range $100,000 to $150,000, having at least 3 bedrooms and 2 bathrooms, and a swimming pool.

Several websites have been developed to achieve this goal and do a good job helping users to find the kind of house or property they are looking for. In this project, I developed a web service using IBM Websphere that serves this same purpose.

Websphere has convenient tools for developing Servlets and JSP pages. Both can be created using a wizard that allows you to choose the methods you wish to use. For example, if you need a doGet() or doPost() method in a Servlet, it automatically generates basic parts of the code, as well as the necessary package statement at the top of the code, as soon as you click finish on the wizard

1.1 Requirements Specifications

The requirements for the Real Estate Web Service are that:

1) The project should allow a user to search for real estate properties in any location in the US by referring to the city and state it would be in.
2) The program should allow a user to search by having extra options to choose from, like some particular features of the property.
3) The project should allow a user to enter a new property's details into the database through an input form, and this data should be available for search by future users.
4) The project should be able to display images of the available property.
5) The program should display more detailed property features when a property's hyperlink is clicked.

1.2 Design Specifications

1) The design decision of this project was to develop Servlets and JSPs in the IBM Websphere environment. IBM Websphere is quite convenient to use in developing a web application.
2) In order to access the database, entity beans were considered so that EJB-QL could be used to write SQL-like functions and thus be able to select (search) the database for property record sets that satisfy the user's query.
3) IBM DB2 was selected as the database, as it would be easier to use with the other IBM product.

For this project, there are three applications: Enterprise Application Project (RealEstate1.EAR), Web Application Project (RealEstateWeb.WAR) and Enterprise Java Beans Project (RealEsate1EJB.JAR). All these three projects coordinate to help in the successful working of the system.

Technologies and terms used in the project:

1.3 Developer Environment used

1) IBM Websphere Studio Application Developer version 5.1.2 (WSAD) was chosen, as it provides an integrated development environment (IDE) for designing and deploying J2EE applications and web services. While J2EE web services can be created without the use of an IDE, the editors, tools and wizards in Websphere make this work much easier. For example, EJB components that are used in this project were easily mapped from the database by using a tool in the developer. Debugging is also made easier in WSAD, by providing a tool to open the debugging environment. The only drawback is that it takes a considerable amount of time to learn how to use the various tools.

2) IBM Websphere Application Server (WAS) delivers an open service infrastructure functions as an engine to execute many applications ranging from J2EE to web services in a secure and reliable way.

3) Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing component-base multi-tier enterprise applications [7)7]. After all the components or applications are individually developed, they can be combined and operated in coordination with the help of J2EE.

4) Servlets: "A Servlet provides web developers with a functionality that extends a web server."[1] While applets are used on the client side, servlets can be thought of as those applets used on the server-side. A servlet has the main code, which does

[1] http://Java.sun.com/products/servlet/index.jsp

the processing of all requests by JSP and HTML pages. Websphere has an integrated test environment, WTE, which can be used to test servlets. This is more efficient to use than the Java 2 SDK Enterprise Edition (J2SDKEE) provided by Sun Microsystems. Servlets can also provide session management, user authentication and authorization. The doGet() and doPost() methods are the basic methods that are usually used in servlets. Either a Get or Post request is supplied to a servlet that processes the request according to the code in the corresponding method. An init() method may also be present for making any initializations.

5) JSPs: "Java Server Pages Technology provides a simplified, fast way to create dynamic content. JSP technology enables rapid development of web-based applications that are server- and platform-independent."[2] Tag files and an expression API facilitate the development of JSP pages. Using servlets with html code is more complicated and confusing. Also, it is unpleasant to read servlet code with HTML embedded in Java print statements. JSP pages are more convenient in that they allow Java code to be enclosed in opening <% and closing tags %> thereby clearly distinguishing between the HTML and Java code. JSPs also allow code to be reusable. The Websphere Page designer allows for three views of a JSP page: Design, Source and Preview. This gives us a glimpse of how the finished JSP page will look when it is run on the server.

6) JDBC (Java Database Connectivity) API (Application Program Interface) is an interface between Java platforms and databases. It allows Java applications to execute SQL statements. Since most databases support SQL, it is possible for Java to interact with almost any kind of DBMS.

7) JNDI (Java Naming and Directory Interface) is a way to give unique names to local home interfaces of enterprise beans so that they can be referred to by a remote client program. The Naming exception is to be thrown if JNDI is used.

1.4 Database (Back end) used

IBM DB2 version 8.1 is the database used in this project. It is claimed by IBM to be robust, scalable and quite easy to use. There are versions of DB2 available for Windows, Unix, and Linux. This project used Windows.

.

[2] http://Java.sun.com/products/jsp/

2. Code Development and Modules

2.1 Servlets used in Project:

1) CMPServlet: This Servlet allows the display of all the properties available in the database. It is invoked by the first index.html page, and after processing the request and retrieving all information from the database, it transfers control to the DisplayTable1.jsp page. It has the doGet() method of the servlet to get all the required information.

2) CreateServlet: This servlet is basically used to create a new property entry in the database. It invokes the ejbCreate method via a reference created using initialcontext() in the doGet() method. It is invoked by the webForm.jsp page, which is the page where the user enters all the details about the property he wants to advertise and sell. Once the recordset is inserted into the database, it transfers control to the index.html welcome page so that the user can choose his next option.

3) SearchServlet: This Servlet processes the request which it gets from the searchForm.jsp page. It invokes the findbyLocation(…) query via a reference in the doGet() method. On getting the matching recordsets, it transfers control to the DisplayTable1.jsp page which is the final page to display all results for any request.

4) ExtraSearchServlet: This servlet is invoked by the extrasearchForm.jsp page which allows the user to choose from more options rather than just the usual city, state, price and bed, bath options. It has the method findbyFeatures(….) which it invokes via a reference by taking all the input parameters supplied to by the user through the extrasearchForm.jsp page input form. Once findbyFeatures(…) retrieves all the matching recordsets as a Collection data type, it sends the control to the DisplayTable1.jsp page as done by the previous servlets.

5) IndivServlet: This servlet invokes the query findbyId(..) query indirectly via a reference it creates using initialcontext() for naming. The input for this query is obtained from the hyperlink on the DisplayTable1.jsp page. If observed closely on the DisplayTable1.jsp page, each property address displayed is a hyperlink, which if clicked starts up the IndivServlet which calls the query findbyId(..), which finds the particular recordset with the primary key specified. IndivServlet dispatches control to the individual.jsp page, which displays only the specified recordset details in a pleasing manner.

2.2 JSP (Java Server Pages) used in project

1) webForm.jsp : This is the input form that enables a user to enter details regarding the property he wants to advertise, e.g., location, price, number of bedrooms, etc.

2) searchForm.jsp: This is a form where the user can enter details describing features of the kind of property he is searching for, e.g., location, price, etc.

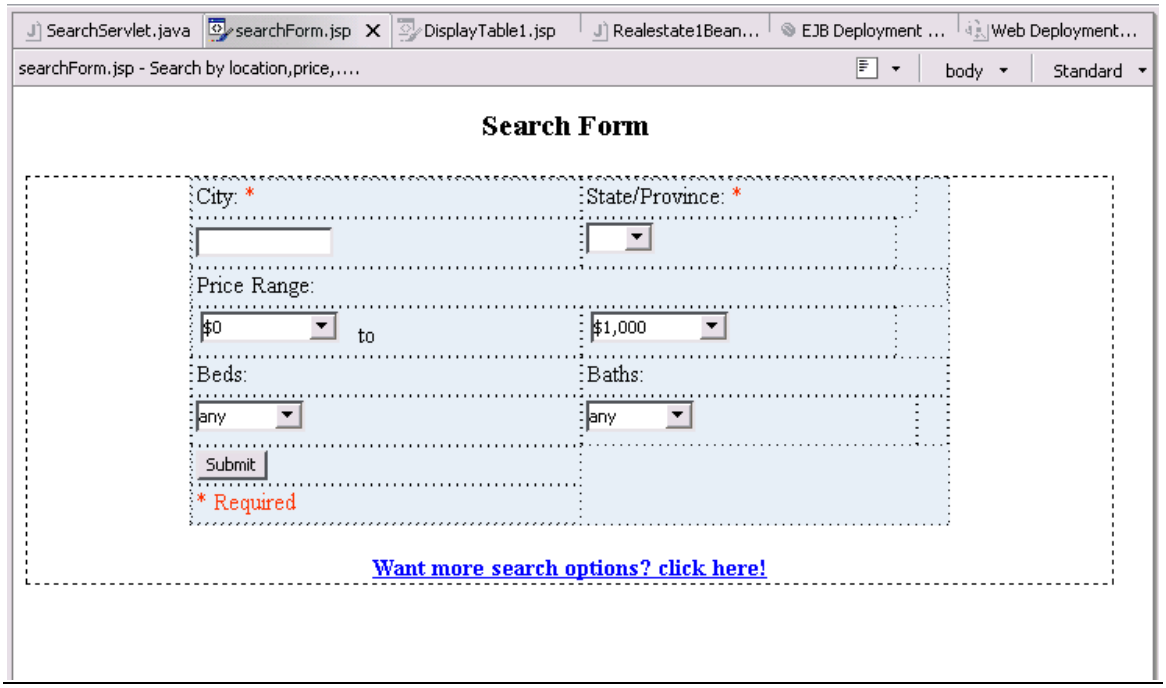
3) extrasearchForm.jsp: This is also a search form, but it gives more options for the user to choose from. For example it also lets the user choose if he would want certain features to be on the property like a swimming pool, etc.

4) DisplayTable1.jsp: This JSP page takes on the responsibility of displaying the results of any request, which could be passed on by any of several Servlets.

5) Individual.jsp: This JSP page displays the details related to only one property, which is identified by it's id in the database. It is invoked by the IndivServlet.Java file.

JSP pages can be easily developed using convenient features, which are supplied through the Websphere development environment. JSP files generally contain HTML-like tags, which allow dynamic content including Java code to be embedded in them. It is easier to edit JSP code than the HTML code inside servlets and also redeploy them if necessary. Tag libraries help in reuse of code as they perform some common functionality, which can be used in any web application. It is called the Java Server Pages Standard Tag Library (JSTL). Below in Figure 1 is a screenshot of how the JSP page designer view looks.

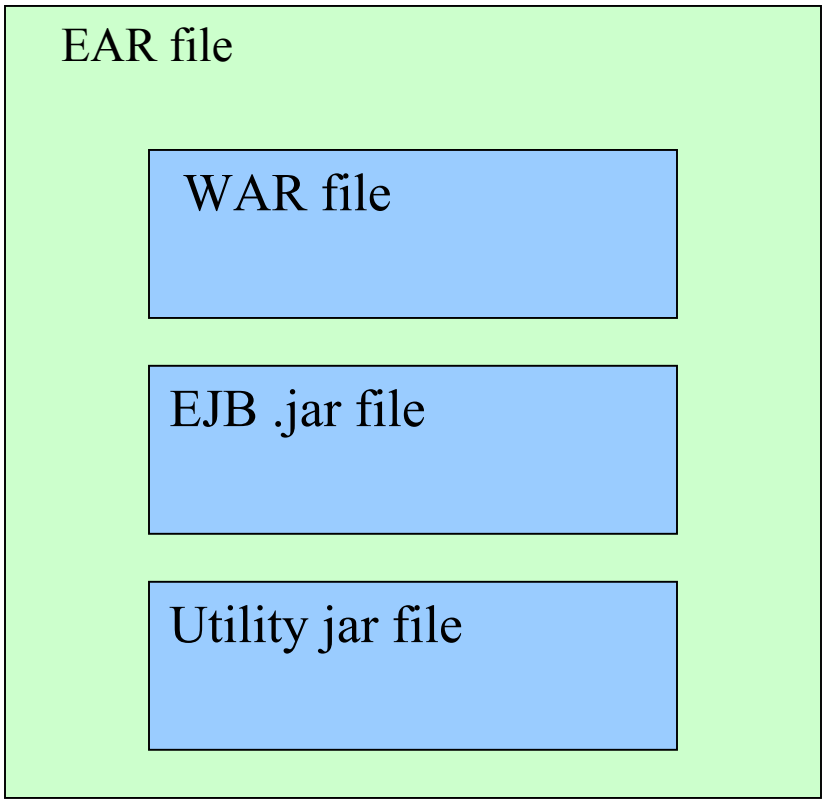
Figure 1: Page Designer view of Search Form JSP page



We can see that the page designer view is more or less like the preview page but it also displays how the table actually looks like in design.

There are three main applications used in this project: Enterprise Application Project (EAR file) which is named RealEstate1EAR.ear, Web Application Project (WAR file) which is named as RealEstate1Web.war, and the EJB project (JAR file) which is named as RealEstate1EJB.jar. These three components coordinate and help in the working of the project. The EAR file is the main file which packages together a number of .jar, .war, etc files in order to form one whole application and run it. Websphere only allows deployment of applications through the standard EAR format unlike other software. Figure 2 shows how all other files are packaged together into an EAR file (Enterprise Application.

Figure 2: Embedded Applications in EAR file



2.3 Uploading Files from the Input Stream

Files can be uploaded from forms online, which is very useful in daily applications, for example uploading of resumes for job search, uploading of pictures, etc. This section describes how uploading of files is done following details from the book 'Java Servlet

Programming' by Jason, Hunt, O'Reilly publishers. RFC 1867 at the website *http://www.ietf.org/rfc/rfc1867.txt* gives details about the file upload specification.

The client side module of a file upload is fairly easy, with a simple HTML form that asks for a file to be uploaded. A file dialog box similar to the operating system type dialog, pops up giving the option to browse for a file from the file system. The user could also directly enter the file name in the textbox provided. The File input type in the HTML script makes this dialog box appear on clicking the browse button on the form. The sample code for the client side is as follows:

```
<FORM ACTION="/servlet/UploadTest" ENCTYPE="multipart/form-
data" METHOD=POST>
What is your name? <INPUT TYPE=TEXT NAME=submitter> <BR>
Which file do you want to upload? <INPUT TYPE=FILE NAME=file> <BR>
<INPUT TYPE=SUBMIT>
</FORM>
```

Next, coming to the server side details, a utility class named MultipartRequest in the book does the job of parsing the data uploaded as servlet API unfortunately doesn't have any class for doing this. The class constructors takes in parameters of filename, name of directory to be saved in, and the size of the file if necessary. An IOException is thrown if any problem occurs during parsing. Other methods in the class are getParameterName which retrieves the name of the parameters, getParameter which gets the value of a parameter and getFileNames to get the list of files uploaded.

A servlet is used to handle the Multipart Request object, which just displays the statistics for what files were uploaded. MutipartRequest class deals with the parameters including the directory root (where the files are to be saved) sent by the servlet request.

3. Java Beans

In order to make reusability of components in Java feasible, Sun Microsystems introduced Java beans which allow reusability of controls in the Java application development. The big advantage of Java beans over all other normal controls and components is that they are independent of operating systems and development environments. Thus flexibility and reusability of the Java bean component is possible for developers who share components. The advantage of a bean over a Java class is that the bean can be manipulated at design time, by analyzing and determining its properties and methods, than at runtime. Screenshots of the code and deployment descriptor are taken to show a better view of how actually the Real Estate Listing project works.

3.1 Enterprise Java Beans

An EJB is a component which contains the business logic and business data of an application. There are three types of EJBs defined in the EJB 2.0 specification:

1. Session EJBs – They are non-persistent objects, can be either stateful or stateless and are generally used for depicting the flow of the business process.
2. Entity EJBs – These are persistent objects and they are of two types – BMP entity beans which manage their own persistence and mapping relations and CMP entity beans which have their mapping and persistence relations managed by the EJB container. We can see that therefore, entity beans represent some data in a database and can participate in transactions by users.
3. Message Driven EJBs – This is similar to a session bean in some ways but is activated only when it receives an asynchronous message from a JMS queue or topic.

3.2 CMP Entity Beans

Coming to discussing about the project implementation, CMP entity beans have been used to map the bean to the database DB2. When we first create a CMP bean using the wizard, it automatically creates the basic classes, Local, LocalHome, Bean class and Primary Key class. The [project name] Bean class has all the main getter and setter methods for access from the Local and LocalHome classes which list out the declarations for all classes defined in the Bean class. This behaves like a remote interface declaring externally accessible methods.

When a client program needs to access entity bean, it's not possible to create it using the new operator as other objects. One would have to give it a unique JNDI (Java Naming and Directory Interface) in order to identify it and use it. The naming service helps it searching for the particular bean home interface using the JNDI name provided. JNDI will be discussed in the next section. A session bean generally has a remote interface which declares the methods that can be accessed from an external object. The sample code for the RealEstate1Bean has been provided below. As we can see, there are four steps in how the EJB can be used: first, an initial naming context is obtained. Secondly,

using the initial context, a lookup is generated and stored as an object reference. Once the reference to the home interface is created, any of the method operations declared in the bean can be used, for example, to find, create, remove, etc. The few basic operations of create; find and remove are automatically created in the generated code. Any other methods required can be added by the programmer.
The code below shows how an initial context is obtained and used to call an EJB-QL find query, i.e., findbyLocation.

Figure 3: Code snippet of SearchServlet

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
                  try {
                  InitialContext context = new InitialContext();
                  Realestate1LocalHome emh = (Realestate1LocalHome)
            context.lookup("local:ejb/property/Realestate1LocalHome");
String action = req.getParameter("submit");
Collection emps = emh.findbyLocation(
req.getParameter("city"), req.getParameter("state"),

new Double(Double.parseDouble(req.getParameter("maxprice"))),
new Integer(Integer.parseInt(req.getParameter("mnbed"))),
new Integer(Integer.parseInt(req.getParameter("mnbath"))));
req.setAttribute("emps", emps);
RequestDispatcher rd =
getServletContext().getRequestDispatcher("/DisplayTable1.jsp");
rd.forward(req,resp);
```

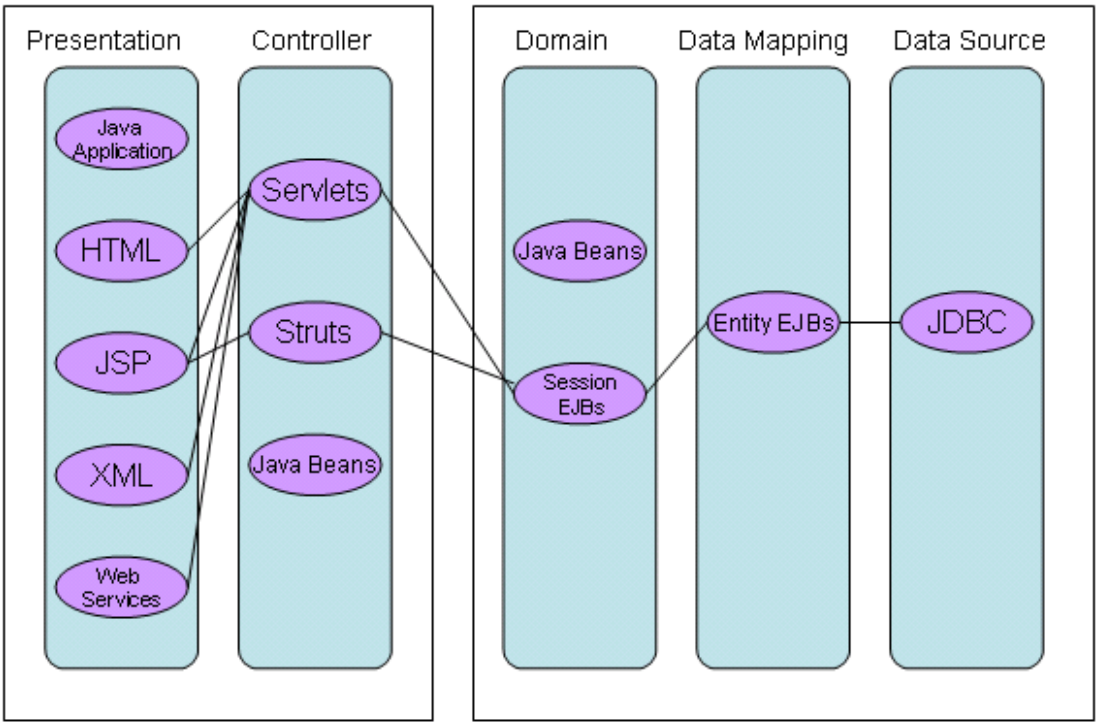
3.3 JNDI (Java Naming Directory Interface)

The Javax.naming package defines all the interfaces for the naming and directory services like JNDI. There are other naming services like RMI, COS and LDAP. A naming exception is thrown in case the name was not found. Usually, the name is given in the Java context as “Java:ejb/RealEstate1”, etc. More information about JNDI specification can be found at http://Java.sun.com/jndi.

3.4 Layered Architecture

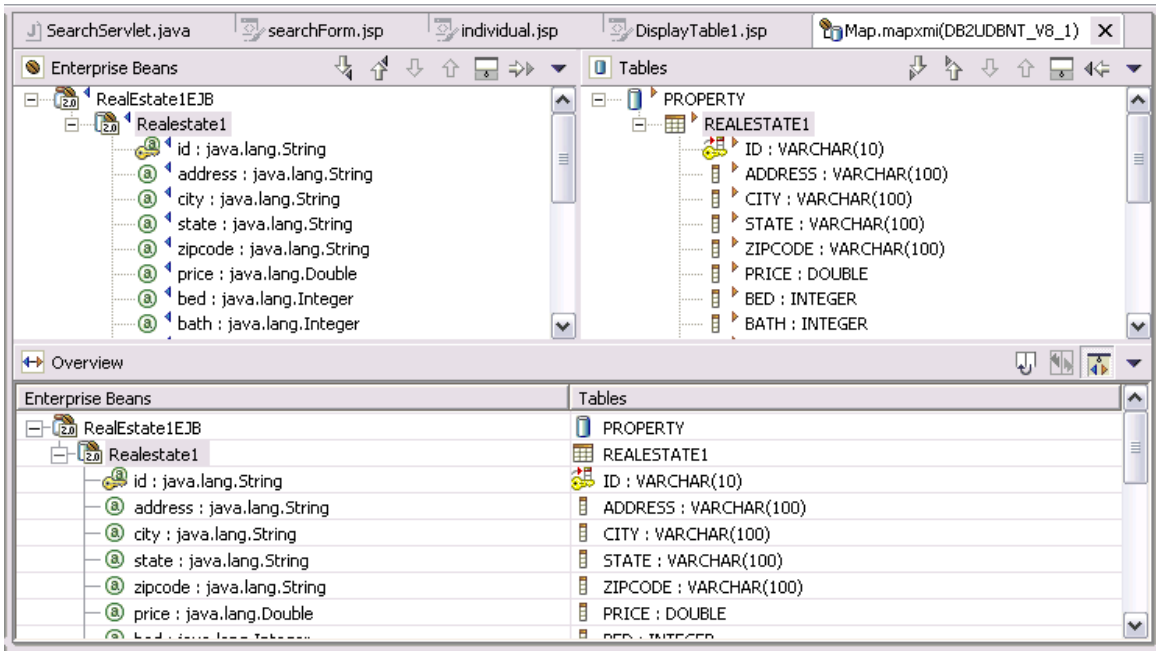
The working of the components in this project can be best depicted using a Layered architecture map as in Figure 6. The presentation layer is at the top with the possible presentation components: Java application, HTML, JSP, XML using XSL and Web services. In this project I use HTML and JSP as the presentation pages to the user. This can also be turned into a Web Service by following a step-by-step procedure in Websphere and then invoking the web service using a proxy in order to test it. This procedure is explained toward the end of the document.

Figure 4: Layered architecture map [2]



The next level is the Controller level and we can observe that Servlets, struts and Java beans make up the level. Servlets are basically used in the controller layer of the project. **Struts** could also be used for validation purposes of a form, etc. The domain layer has session beans employed mainly which is also employed in this project. These are like an intermediate between the Servlet and the entity bean invocation. Request beans used in the JSP page are also some kind of Java beans which are used in the project. Entity EJBs are used in the Data Mapping layer, and in the case of this project, more specifically CMP (Container Managed Persistent) entity beans are used which directly access the database (via the local interface) through the use of some automatically generated code. Finally, the last layer is the Data Source layer which lists JDBC (Java Data Base Connectivity) as the one tool used in connecting the code to the underlying database. Figure 5 shows a screenshot of Map.mapxmi file (visible in the J2EE perspective view) which represents the mapping from the enterprise bean RealEstate1EJB to the database PROPERTY.

Figure 5: Screenshot of Map.mapxmi file



From the above screen shot, we can observe that each field of the database table RealEstate1 is mapped exactly to the same name attribute (as that of the field) of the bean correspondingly. This is done automatically when the wizard is used to create a bottom-up mapping. There are three types of mappings possible using the wizard in Websphere, namely top-down mapping, meet-in-the-middle mapping and bottom-up mapping. Below, I display the default code generated by the wizard for the ejbCreate(int) and ejbPostCreate(int) methods. I had to manually create the ejbCreate(String, String,……..) and its corresponding ejbPostCreate methods as required for the project which follow the default code below.

Figure 6: Code snippet of Realestate1Bean class

```
/**
	 * ejbCreate
	 */
	public property.Realestate1Key ejbCreate(Java.lang.String id)
		throws Javax.ejb.CreateException {
		setId(id);
		return null;
	}
	public property.Realestate1Key ejbCreate(Java.lang.String id,
Java.lang.String address, Java.lang.String city, Java.lang.String
state, Java.lang.String zipcode, Java.lang.Double price,
Java.lang.Integer bed, Java.lang.Integer bath, Java.lang.Double
totalarea, Java.lang.String kitchenarea, Java.lang.String livingarea,
Java.lang.Integer yearbuilt, Java.lang.Integer washer,
Java.lang.Integer dryer, Java.lang.Integer carpet, Java.lang.Integer
centralac, Java.lang.Integer centralheat, Java.lang.Integer dishwasher,
Java.lang.Integer swimmingpool, Java.lang.Integer diningroom)
```

```
                        throws Javax.ejb.CreateException {
                        setId(id);
                        setAddress(address);
                        setCity(city);
                        setState(state);
                        setZipcode(zipcode);
                        setPrice(price);
                        setBed(bed);
                        setBath(bath);
                        setTotalarea(totalarea);
                        setKitchenarea(kitchenarea);
                        setLivingarea(livingarea);
                        setYearbuilt(yearbuilt);
                        setWasher(washer);
                        setDryer(dryer);
                        setCarpet(carpet);
                        setCentralac(centralac);
                        setCentralheat(centralheat);
                        setDishwasher(dishwasher);
                        setSwimmingpool(swimmingpool);
                        setDiningroom(diningroom);
                        return null;
                }
        /**
         * ejbPostCreate
         */
        public void ejbPostCreate(Java.lang.String id)
                throws Javax.ejb.CreateException {
        }
        public void ejbPostCreate(Java.lang.String id, Java.lang.String
address, Java.lang.String city, Java.lang.String state,
Java.lang.String zipcode, Java.lang.Double price, Java.lang.Integer
bed, Java.lang.Integer bath, Java.lang.Double totalarea,
Java.lang.String kitchenarea, Java.lang.String livingarea,
Java.lang.Integer yearbuilt, Java.lang.Integer washer,
Java.lang.Integer dryer, Java.lang.Integer carpet, Java.lang.Integer
centralac, Java.lang.Integer centralheat, Java.lang.Integer dishwasher,
Java.lang.Integer swimmingpool, Java.lang.Integer diningroom)
                        throws Javax.ejb.CreateException {
        }
```

The next step to be done is to promote the ejbCreate method to the LocalHome Interface so that it can be visible to the outside client during lookup. In the process of doing this, a corresponding method is created in the LocalHome interface class for the bean. In order for the created bean to be used in the application, a mapping needs to be generated by selecting Generate > EJB to RDB mapping from the context menu of the EJB module. Backend folders and mapping files Map.mapxmi as depicted in the figure above are automatically generated.

In the top-down approach, each attribute of the EJB is mapped to a column of the same name in the database table. In the bottom-up approach, each column of the database table is mapped to an attribute of the same name in the EJB (which is the opposite of how the top-down approach works). Another important thing to note is that the JNDI name of the EJB needs to be correctly typed in the EJB deployment descriptor. Naming exceptions

can result if not done carefully. Now the time comes to deploy the RMIC code finally, we do this by selecting the RealEstate1 module and select 'Deploy and RMIC code' from the pop up context menu. A number of classes are created corresponding to the classes created during mapping and programmer modification of those classes. Each time any change is made to the basic four classes, i.e., the Local, LocalHome, Bean or Key classes, the code has to deployed all over again otherwise it may point out some errors in the tasks column.

A convenient mechanism for testing the newly created bean is provided by the UTC (Universal Test Client). This allows a friendly interface to enter information and invoke the methods in the bean to insert, update, remove records in the database via the CMP entity bean.

3.5 EJB-QL (Enterprise Java Beans Query Language)

EJB-QL is a query language that defines select and finder methods for CMP (Container-managed persistence) entity beans. A typical EJB-QL query consists of the main clauses SELECT, FROM and WHERE. EJB-QL 2.0 is based on SQL-92 and is used with CMP entity beans. EJB-QL 2.0, unlike EJB 1.1, offers the capability to form relationships too. The queries are defined in the EJB deployment descriptor. By using EJB-QL, CMP beans can be independent of the underlying database.
There are two types of queries possible – finder queries and select queries and this is exactly what the wizard asks you to select from. It also asks us to select a sample query which we can modify according to our requirement. Input parameters can also be added if required and it is necessary to specify the data type of input parameter if added. There is also another way to classify the types of queries, i.e. as simple, with input parameters and with relationships.
The simplest query possible is the simple findAll() query which displays all the available properties with no other criteria to restrict the selection.
All the queries used in this project are of the type 'queries with input parameters'. These can be described as those EJB-QL queries which have at least one input parameter represented by '?n' where n can be 1,2,…. In the EJB-QL query employed in this project, the obvious input parameters would be all the search parameters as those required to be displayed on the search web input form. The Real Estate Search form includes input parameters to be entered as city, state, minimum and maximum price, and the number of bedrooms and bathrooms. All these parameters are as corresponding input parameters in the finder query. The query findbyLocation(…..) is as follows:

select object(o) from Realestate1 o where o.city=?1 and o.state=?2 and o.price>?3 and o.price<=?4 and o.bed>=?5 and o.bath>=?6

In the above query, o represents the object Realestate1 which is the table name. o.city represents the city field in the recordset, o.state the state field in the recordset, and so on. We can see that o.price (which is the corresponding field in the table) is checked if it is within the range specified by the input parameters thus satisfying the user's request that the price should be bounded between the values input in the search form. Similarly, the

number of bedrooms and bathrooms are compared to the input parameters 5 and 6, to see if the corresponding field in the recordset is greater than or equal to the number specified as input. The return type for the query is Java.util.Collection. The resultset of this query is a set of records which have all the input parameters satisfied; in this case it is if the city, state, price range, and number of beds and baths are as specified by the user. Each separate recordset from the collection is obtained by using the iterator and then passed to the reference which is invoked by the method in the SearchServlet.Java class. Then the Servlet passes the results to a JSP page which displays it.

The next query used is findbyFeatures(……) which has all the input parameters just like the previous query, but it also has additional parameters indicating the flags representing the presence of washer, dryer, and other property features. The return type of this query is Java.util.Collection. Below is the findbyFeatures(……) query:

select object(o) from Realestate1 o where o.city = ?1 and o.state=?2 and o.price>=?3 and o.price<=?4 and o.bed>=?5 and o.bath>=?6 and o.washer>=?7 and o.dryer>=?8 and o.carpet>=?9 and o.centralac>=?10 and o.centralheat>=?11 and o.dishwasher>=?12 and o.swimmingpool>=?13 and o.diningroom>=?14

This is also invoked in the same way as the previous by using an initialcontext() and reference. The property features, i.e. washer, dryer, carpet and so forth are represented by checkboxes on the JSP page so that they can be checked by the user which passes its results to this query.

4. Deployment Descriptor Files

4.1 Application Deployment Descriptor

The Application Deployment Descriptor is the file application.xml present in the META-INF folder. It has a section called Modules where all the web applications war files, EJB application jar files and utility jar files are listed one by one. Each module has URI to the WAR or JAR file which has the module implementation. The context root specifies the URL prefix which is used when the application is run to get the path of the pages.
Below is a code listing of the application deployment descriptor for the RealEstate1 EAR application.

Figure 7: Code listing of Application Deployment Descriptor (application.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.3//EN" "http://Java.sun.com/dtd/application_1_3.dtd">
<application id="Application_ID">
      <display-name>RealEstate1EAR</display-name>
      <module id="WebModule_1090120827205">
            <web>
                  <web-uri>RealEstate1Web.war</web-uri>
                  <context-root>RealEstate1Web</context-root>
            </web>
      </module>
      <module id="EjbModule_1090431386503">
            <ejb>RealEstate1EJB1.jar</ejb>
      </module>
</application>
```

4.2 Web Deployment Descriptor

The Web Deployment Descriptor is the web.xml file present in the WEB-INF folder of a Web application. This basically describes all the modules, and resource files it uses in the application and information about how they can be invoked, any .jar references, database references, etc. The first part of the XML file contains information about the servlets, and then all the JSP files used in the project are mentioned. The second part displays the welcome index pages which could be used and then finally the servlet-mapping which associates a URL pattern with the servlet name. Other information like login information, security roles and constraints, environment entries and JSP tag library descriptors are also listed if used. A resource reference can be added in the resource tab of the descriptor by clicking add and typing in JDBC/database name.
Below is a screenshot of the web deployment descriptor for the RealEstate1 Web application part.

Figure 8: Code listing of Web Deployment Descriptor (web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://Java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app id="WebApp">
	<display-name>RealEstate1Web</display-name>
	<servlet>
		<servlet-name>CMPServlet</servlet-name>
		<display-name>CMPServlet</display-name>
		<servlet-class>com.source.CMPServlet</servlet-class>
	</servlet>
	<servlet>
		<servlet-name>DisplayTable</servlet-name>
		<display-name>DisplayTable</display-name>
		<jsp-file>/DisplayTable.jsp</jsp-file>
	</servlet>
	<servlet>
		<servlet-name>CreateServlet</servlet-name>
		<display-name>CreateServlet</display-name>
		<servlet-class>com.source.CreateServlet</servlet-class>
	</servlet>
	<servlet>
		<servlet-name>SearchServlet</servlet-name>
		<display-name>SearchServlet</display-name>
		<servlet-class>com.source.SearchServlet</servlet-class>
	</servlet>
	<servlet>
		<servlet-name>searchForm</servlet-name>
		<display-name>searchForm</display-name>
		<jsp-file>/searchForm.jsp</jsp-file>
	</servlet>
	<servlet>
		<servlet-name>webForm</servlet-name>
		<display-name>webForm</display-name>
		<jsp-file>/webForm.jsp</jsp-file>
	</servlet>
	<servlet>
		<servlet-name>ExtraSearchServlet</servlet-name>
		<display-name>ExtraSearchServlet</display-name>
		<servlet-class>com.source.ExtraSearchServlet</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>IndivServlet</servlet-name>
		<display-name>IndivServlet</display-name>
		<servlet-class>com.source.IndivServlet</servlet-class>
	</servlet>
	<servlet>
		<servlet-name>DisplayTable1</servlet-name>
		<display-name>DisplayTable1</display-name>
		<jsp-file>/DisplayTable1.jsp</jsp-file>
	</servlet>
	<servlet>
		<servlet-name>extraSearchForm</servlet-name>
		<display-name>extraSearchForm</display-name>
		<jsp-file>/extraSearchForm.jsp</jsp-file>
	</servlet>
	<servlet>
		<servlet-name>individual</servlet-name>
		<display-name>individual</display-name>
		<jsp-file>/individual.jsp</jsp-file>
```

```
	</servlet>
	<servlet-mapping>
		<servlet-name>CreateServlet</servlet-name>
		<url-pattern>/CreateServlet</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>SearchServlet</servlet-name>
		<url-pattern>/SearchServlet</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>ExtraSearchServlet</servlet-name>
		<url-pattern>/ExtraSearchServlet</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>IndivServlet</servlet-name>
		<url-pattern>/IndivServlet</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
		<welcome-file>index.htm</welcome-file>
		<welcome-file>index.jsp</welcome-file>
		<welcome-file>default.html</welcome-file>
		<welcome-file>default.htm</welcome-file>
		<welcome-file>default.jsp</welcome-file>
	</welcome-file-list>
	<ejb-ref id="EjbRef_1090972447276">
		<ejb-ref-name>source/RealEstateSession</ejb-ref-name>
		<ejb-ref-type>Session</ejb-ref-type>
		<home>source.RealEstateSessionHome</home>
		<remote>source.RealEstateSession</remote>
		<ejb-link>RealEstate1EJB1.jar#RealEstateSession</ejb-link>
	</ejb-ref>
</web-app>
```

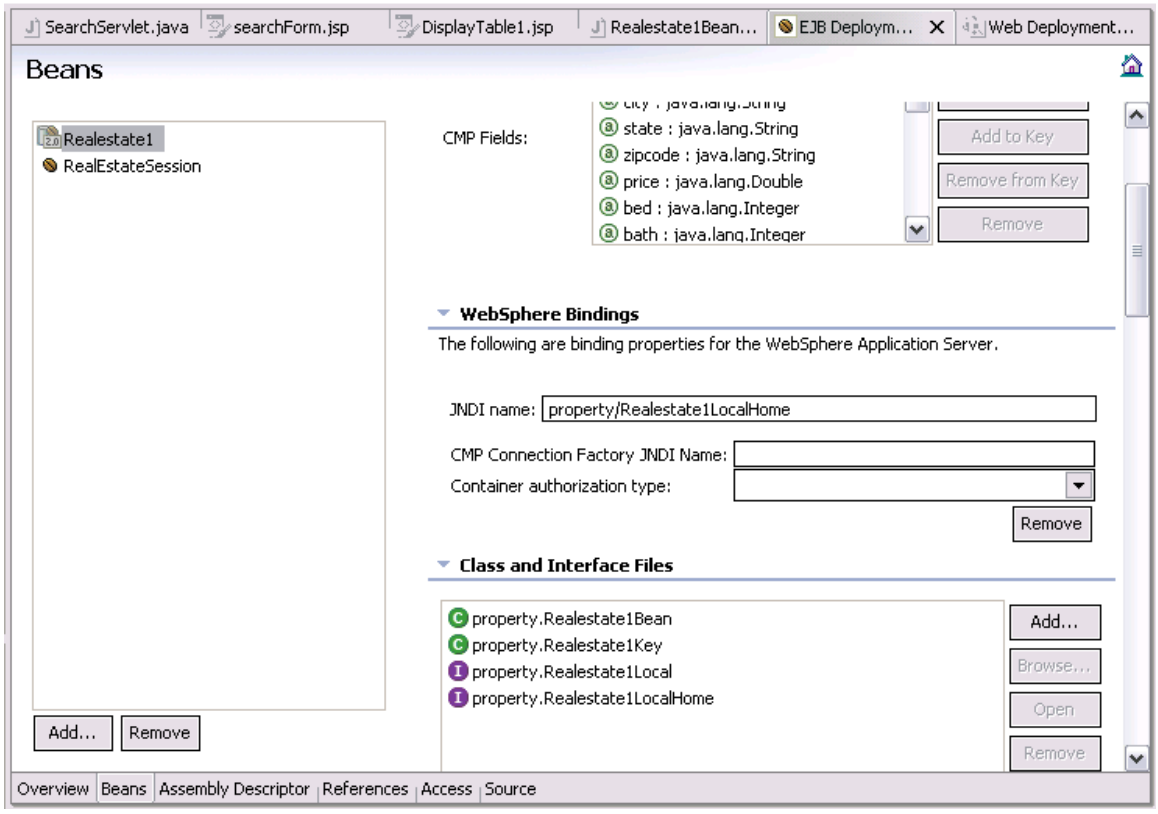
4.3 EJB Deployment Descriptor

The EJB deployment descriptor is an important XML file which describes any beans used in the EJB project. The file's name is ejb-jar.xml which is in the META-INF subdirectory. At runtime, the J2EE reads the deployment descriptor and acts accordingly on the application. Below is a screenshot of the friendly interface of the EJB deployment descriptor.

The above EJB deployment descriptor screen shows the Beans tab. Two beans can be seen defined, one a CMP (Container Managed Persistence) entity bean Realestate1 and the other a session bean, RealEstateSession. The right hand side shows various sections of the selected bean Realestate1. The top most is the CMP fields declared in the bean, next we can see the title 'Websphere Bindings, under which the unique JNDI name of the bean is given. This JNDI name is used in initialcontext when trying to access the LocalHome interface from some other method. The next section we can observe is the 'Class and Interface Files' section which lists the four important files, from which other bean files are deployed, i.e. Realestate1Bean, Realestate1Key, Realestate1Local and

Realestate1LocalHome. All these files are within the package ‘property’. There are other sections too including the Queries section. The queries section of this bean defines all the queries used in this project, i.e. findAll(), findbyLocation(…), findbyFeatures(…), and findbyId(..). The References tab which can be see at the bottom of the screenshot, allows one to add references to a bean. For example, a session bean can refer to another CMP entity bean.

Figure 9: EJB Deployment Descriptor Beans tab



5. Database and Websphere Test Environment (WTE)

The backend used in this project is IBM DB2 version 8.1. It has the database PROPERTY with schema ACME, which has a table called RealEstate1. Details about its field names and types are discussed below.

5.1 Database Table used in project

1) ID – Each house property has a unique id associated with it which is also the primary key of this table. Data type is varchar(60).
2) CITY – This is the location of the property. Data type is varchar(60).
3) STATE -- This is the state where the property is located. Data type is varchar(60).
4) PRICE – This is the price of the property. Data type is double.
5) BED – Represents the number of bedrooms in the property. Data type is integer.
6) BATH – Represents the number of bathrooms in the property. Data type is integer.
7) TOTALAREA – The total area of the property in square feet. Data type is double.
8) KITCHENAREA – The total kitchen are of the property in square feet. Data type is varchar(60).
9) LIVINGAREA – The living room area of the property in square feet. Data type is varchar(60).
10) YEARBUILT – This is the year in which the property was constructed. It thus represents the age of the home. Data type is integer.
11) WASHER – A flag to represent if a laundry washer is present on the property. Data type is varchar(yes/no).
12) DRYER – A flag to represent if a laundry dryer is present on the property. Data type is varchar(60).
13) CARPET – A flag to represent if the property is carpeted. Data type is varchar(60).
14) CENTRALAC – A flag to represent if central A/C is present on the property. Data type is varchar(60).
15) CENTRALHEAT – A flag to represent if central heat is present on the property. Data type is varchar(60).
16) DISHWASHER – A flag to represent if a dishwasher is present on the property. Data type is varchar(60).
17) SWIMMINGPOOL – A flag to represent if a swimming pool comes with property. Data type is varchar(60).
18) DININGROOM – A flag to represent the presence of a dining room on the property. Data type is varchar(60).

This table maps to the RealEstate1 bean with the exact same names as the fields. The corresponding datatypes mapped are as follows: integer mapped to Java.lang.Integer object, varchar(60) mapped to Java.lang.String and double mapped to Java.lang.Double.

5.2 Websphere Test Environment (WTE)

Websphere Test Environment (WTE) works as a test environment to test the J2EE application before it can actually be tried on the Websphere Application Server. We can create any number of servers for testing (like one for every project) and then we have to configure the server before running the enterprise application on it. The databases used in the project have to be added to the Data Sources tab of the configuration editor. In entering the database information, we have to make sure that we enter the same JNDI name that we referred to in the web deployment descriptor which would otherwise cause runtime errors. There are several other tabs in WTE like the Applications tab which lists out all the applications being run on this server and other options. Another tab is the Ports tab which lists all the port numbers required for the server to run, for example HTTP port, SOAP port, etc. Then there are other sections like security tab, configuration and web which can be set as required. Once these settings are updated, the web application can be right-clicked for the context menu to pop up and select 'Run on Server' option to get the server running. There is an option to also debug on the server in Websphere which is very useful for programmers as it opens up a Debug perspective. This perspective has a step-by-step mode for debugging servlet code.

Figure 10: Data Sources tab of Websphere Test Environment (WTE)

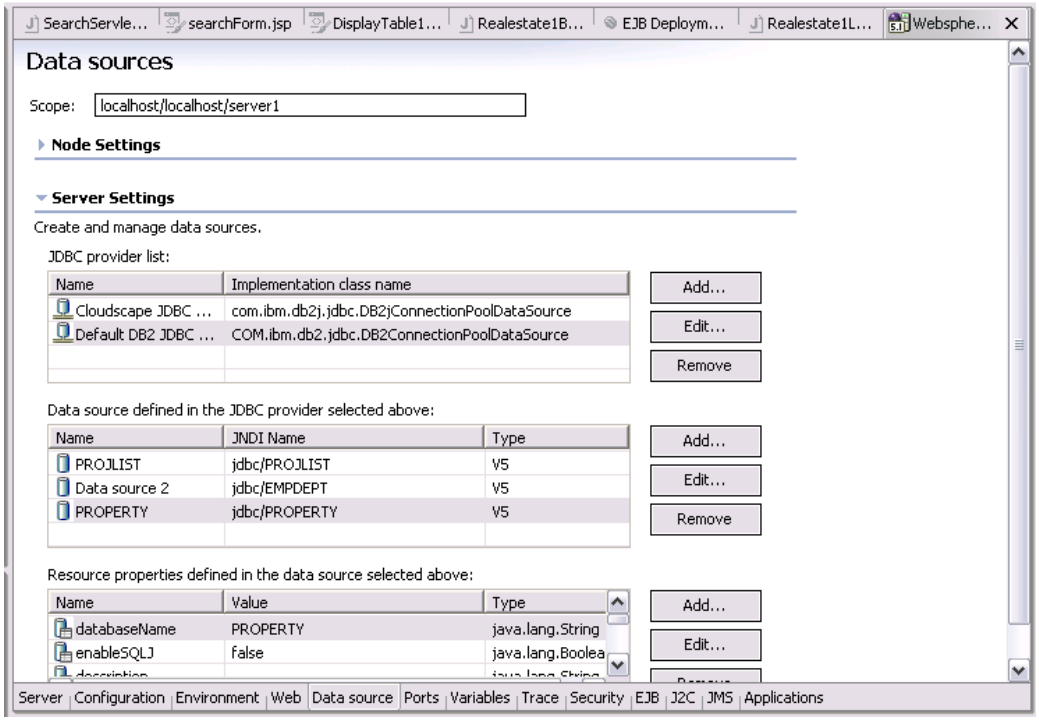


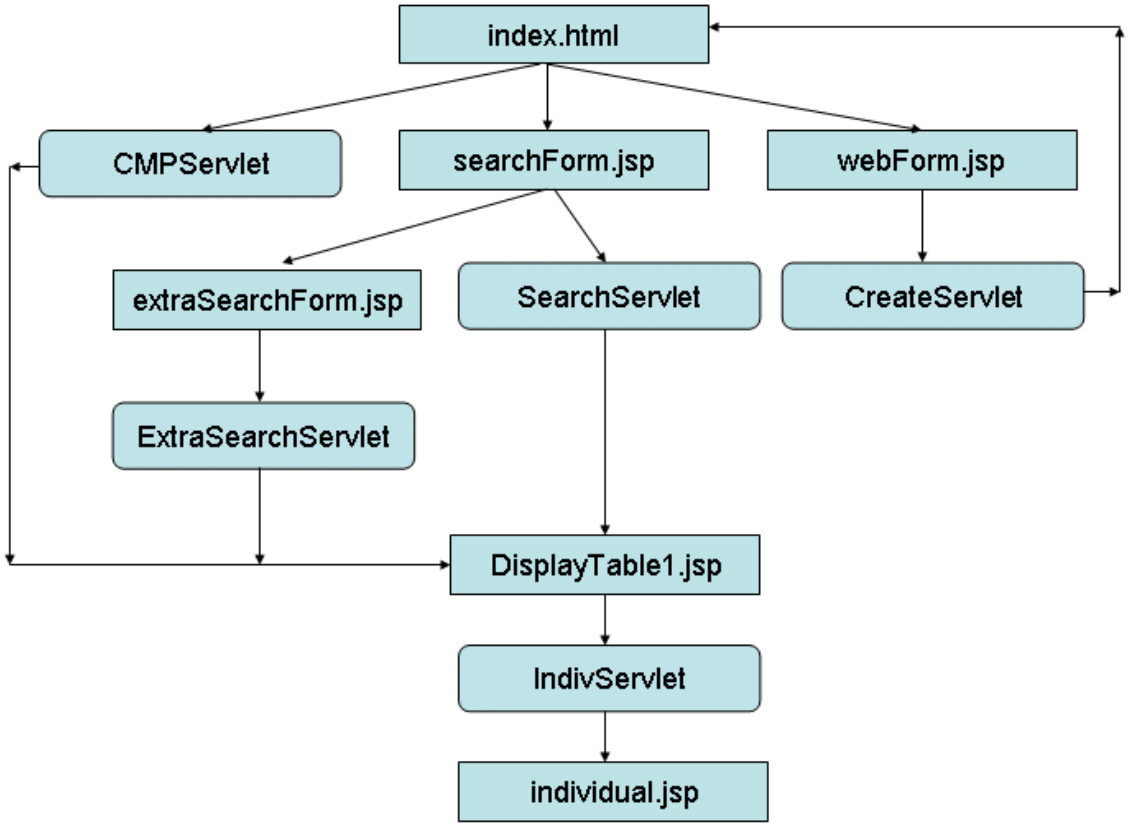
Figure 10 shows a screenshot of how the Data Sources tab of WTE looks like. We can observe that all the databases used in the applications are listed with the JNDI name, type and value. JDBC/PROPERTY is the JNDI name of the database used in this project. All these have to be declared correctly in the WTE before they can be used in the application otherwise a runtime error occurs.

6. Flowcharts

The following flowchart in figure 11 describes how all the servlets and jsps are invoked in the order required for the Real Estate Service to work. From the diagram, one can observe that the main idex.html is the welcome page for the project which is the first page to be displayed according to the web deployment descriptor. This has three hyperlinks:

1) 'search for properties' – If clicked, it takes one to the searchForm.jsp page which allows the user to select the location, price, beds, baths information to find properties.
2) 'want to sell your house, click here' – If clicked, it takes one to the webForm.jsp page which allows the user to input various parameters associated with the property he/she wants to sell.
3) 'display all available properties' – If clicked, it takes one to the DisplayTable1.jsp via the CMPServlet.Java code which processes the request to display all the available properties in the database.

Figure 11: Flowchart



The searchForm.jsp page has quite a few selections to be made by the user and once done with that, they click the submit button which transfers control to the SearchServlet which handles the request. After the request is processed, it passes control to the DisplayTable1.jsp page which has the responsibility of displaying the list of all the properties which are retrieved by the Servlet.

The webForm.jsp page is actually an input form which allows the user to enter all the property details and press the submit button which calls the CreateServlet in this case, to handle this request. The CreateServlet calls the ejbCreate(..) method in the RealEstate1Bean code of the CMP entity bean which creates a new recordset in the database with all the information that the user entered. Next time when a search is requested for, this new recordset is also included in the search. The CMPServlet is basically calling the findAll(..) EJB-QL query in the LocalHome interface of the bean. After processes the request, it hands over responsibility of displaying the results to the DisplayTable1.jsp page.

On the searchForm.jsp page, there is a hyperlink saying 'want more options?' which if clicked, leads to the extrasearchForm.jsp page which allows more options for selection by the user. The extrasearchForm.jsp page has several checkboxes which if checked indicates that the particular property feature is desired in the property the user is searching for. In order to process the request after the submit button is pressed, the ExtraSearchServlet is invoked immediately so that it can invoke the findbyFeatures(…) EJB-QL query to select the collection of matching recordsets. On retrieving the Collection, the displaying responsibility is dispatched to the DisplayTable1.jsp page as before. So, we can observe that the DisplayTable1.jsp page is the common jsp page called when anything is to be displayed.

The DisplayTable1.jsp page has a several hyperlinks referring to the address of the property. If that link is clicked, the IndivServlet is invoked which processes the request of finding the property with the particular id supplied through the hyperlink. This is successfully done by using the findbyId(..) EJB-QL query defined in the EJB deployment descriptor. The id is the input parameter for the above mentioned query and it retrieves only that recordset with that unique id as it is the primary key too. This time to display the results of this servlet request, individual.jsp page is called. It displays the property's location, price, beds, baths and all the property features associated with it.

In order to explain how the SearchServlet (or for that matter the ExtraSearchServlet) invokes the CMP bean's code, a diagram has been provided in Figure 12 which shows the method invocation order. As we can see, first a JNDI reference is obtained using intialicontext() facility, which allows the LocalHome interface of the bean to be accessed. The RealEstate1LocalHome bean code has a list of all methods out of which the findbyLocation (or for that matter the findbyFeatures(…)) method is invoked which retrieves all the recordsets as a Collection from the RealEstate1 Table of the PROPERTY Database.

Figure 12: Flowchart showing invocation of EJB-QL query

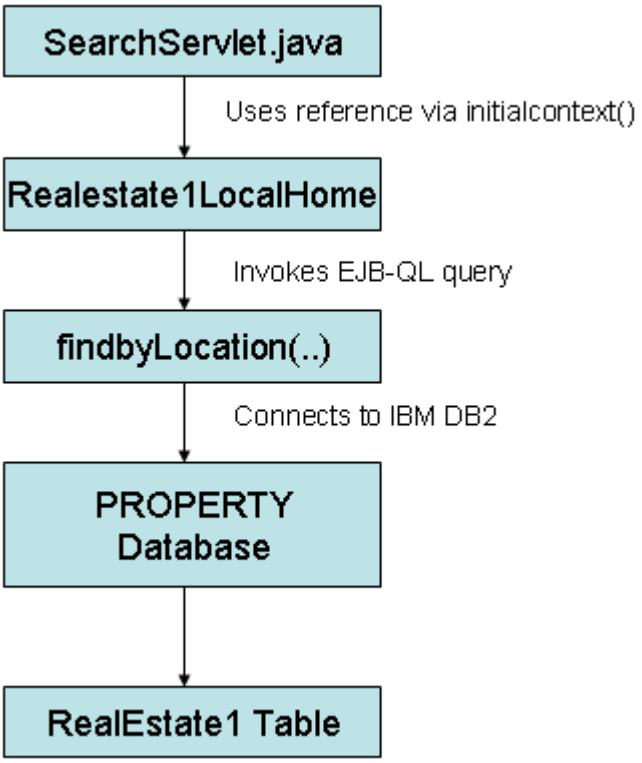
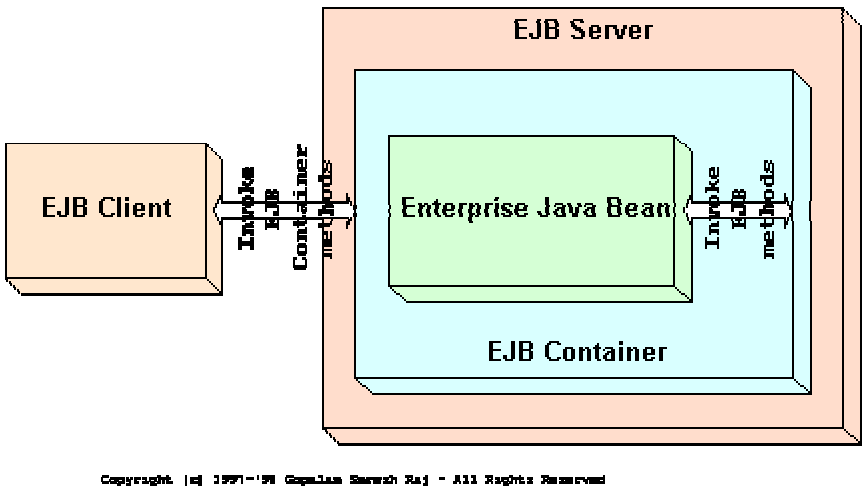


Figure 13: Invoking of Enterprise Java Beans [6]



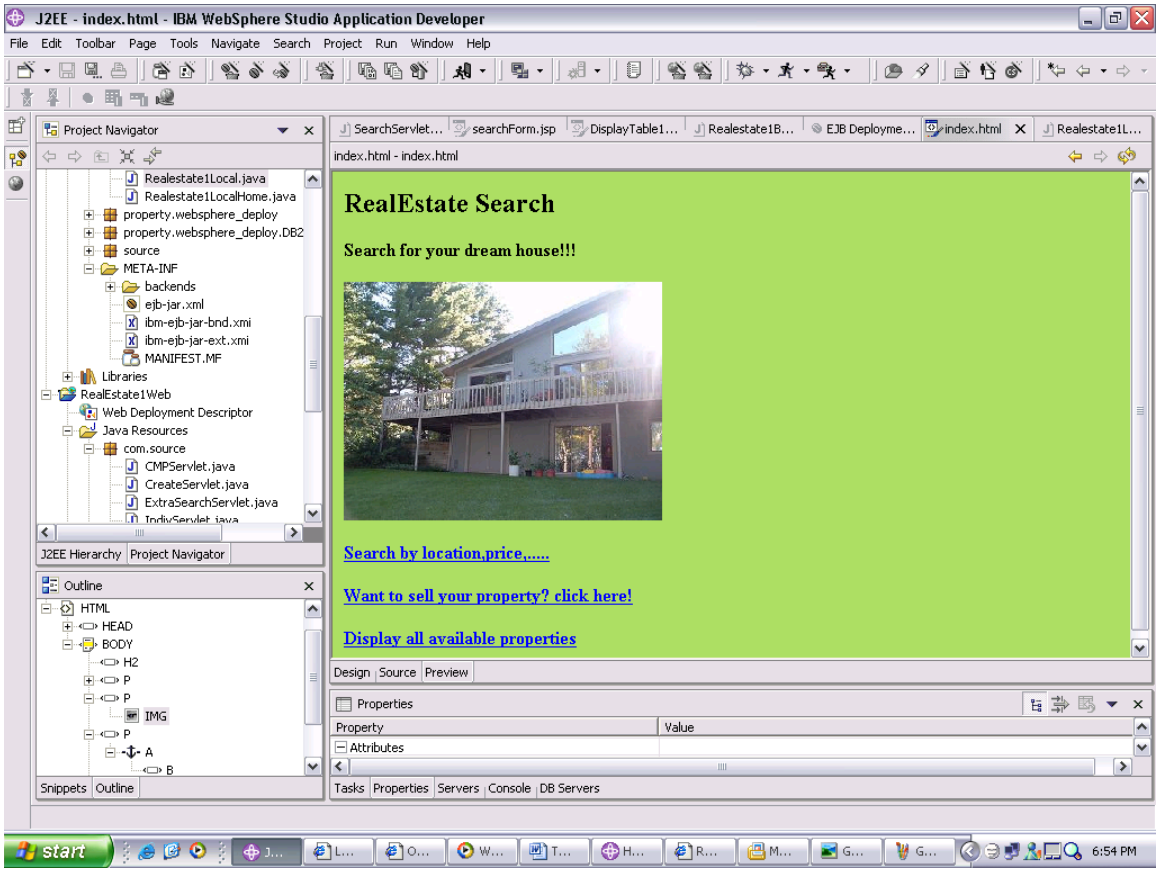
The above Figure 13 is a general way of describing how the EJB Client (in this case, it is one of the Servlets like SearchServlet.Java) invokes the EJB bean methods. As seen from the figure, the Enterprise is enclosed in an EJB Container, which is a runtime environment in the J2EE (or EJB) server. Similarly, the Servlets and JSP pages reside in

the Web container. The EJB client first invokes the EJB container (or the Web container) which then invokes the actual EJB (or the actual Servlet or JSP).

7. On Running the Server

On running the application on the server, following is the order of how the pages are displayed and presented to the user. The first page displayed on running is the index.html page which has its screenshot shown as below.
The first index.html page looks like the figure 14 below:

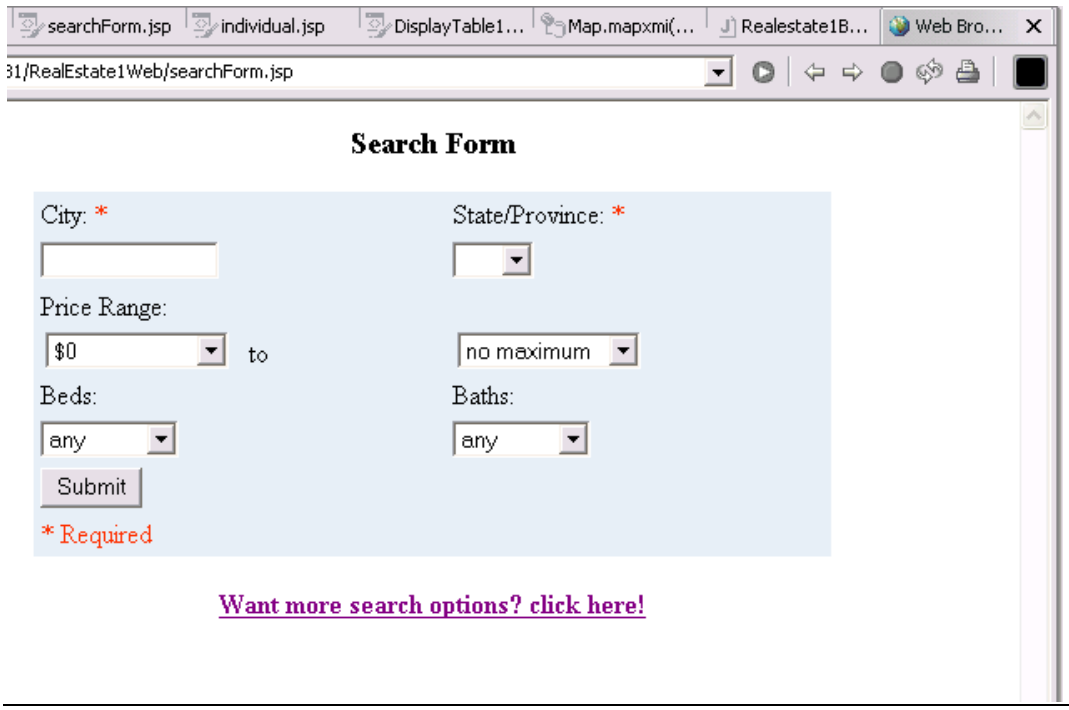
Figure 14: index.html welcome page for project



The screenshot in figure 15 gives us a glance at how the Search Form looks like. The SearchForm is a JSP page which is friendly user interface allowing users to enter the

search details to find the property among a huge list of real estates available. The city has to be typed in as it is normal textbox but the state can be chosen from a drop-down list. The minimum and maximum price range also can be chosen from a drop-down list as also the number of bedrooms and bathrooms. The default minimum is put to $0 and the default maximum to $9999999999 (a very big number). These values are sent as parameters to the action Servlet which in this project is the SearchServlet.Java Servlet.

Figure 15: searchForm.jsp page



The parameters are referred to by using 'req.getParameter(…)' and used in the findbyLocation(…) method of the RealEstate1 bean. The exceptions which have to be thrown by the doGet() method of the Servlet are FinderException() and NamingException(), which are respectively for the finder query and the initialcontext previously defined. The resultant collection of recordsets obtained by using this method is passed to the DisplayTable.jsp webpage which displays the results of the search query. This transfer of control from the servlet with the resultant recordsets to the display jsp page is successfully done by using a request bean. The request bean is defined as follows at the top of the DisplayTable.jsp page:

```
<jsp:useBean id="emps"
      type="Java.util.Collection" scope="request"></jsp:useBean>
```

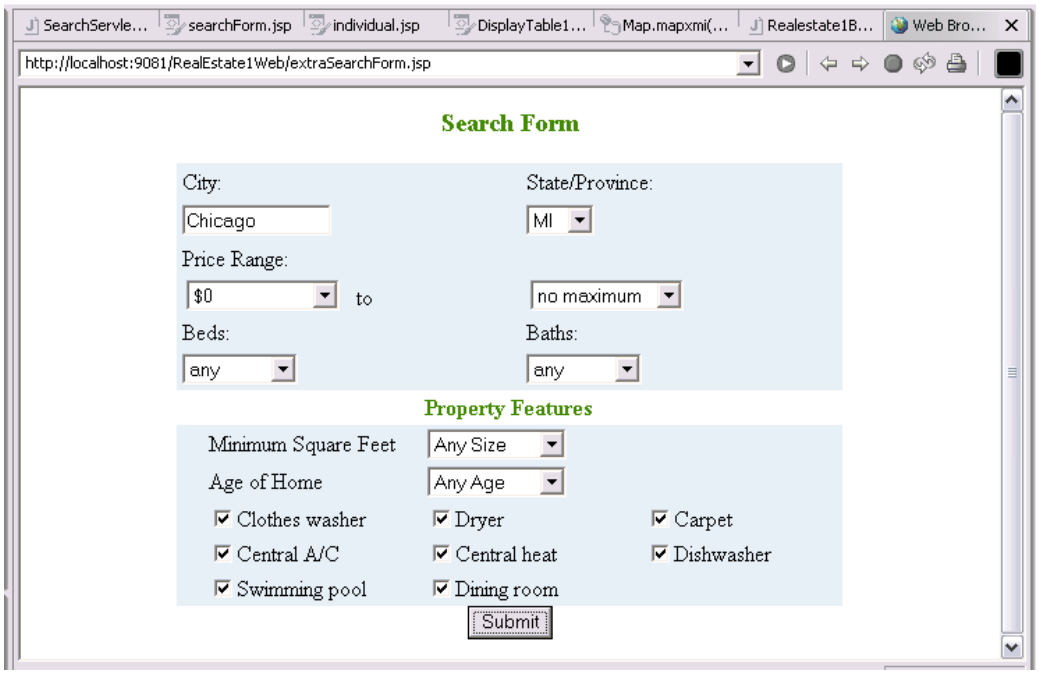
Once this request bean is set in the servlet, it can be used in the JSP page to which control is transferred to. The bean is used with the iterator method as follows which allows each recordset to be retrieved from the whole collection.

```
<%  Iterator projs = emps.iterator();
                while (projs.hasNext()) {
                  Realestate1Local     proj     =     (Realestate1Local)
projs.next();%>
```

The getter methods like getId, getAddress, etc are all defined in the RealEstate1 bean described earlier and also listed out in the local interface class. In this way, any other remote class is able to use those methods.

The next screenshot has more search options which are property features such as minimum square feet area, age of home, clothes washer, dryer, carpet, central a/c, central heat, dishwasher, swimming pool and dining room. Extra features could still be added to the present ones, but I have represented a few basic features in this project.

Figure 16: extrasearchForm.jsp page



The next screenshot shows us an input form which is displayed when the user clicks the link ‘want to sell your property’ on the main index html page. This allows the user to enter details of the property he would want to advertise on this website. The entered information is entered into the database using the ejbCreate methods which are automatically generated by the wizard when the bean was first created.

Figure 17: Realestate details Entry Form (webForm.jsp page)

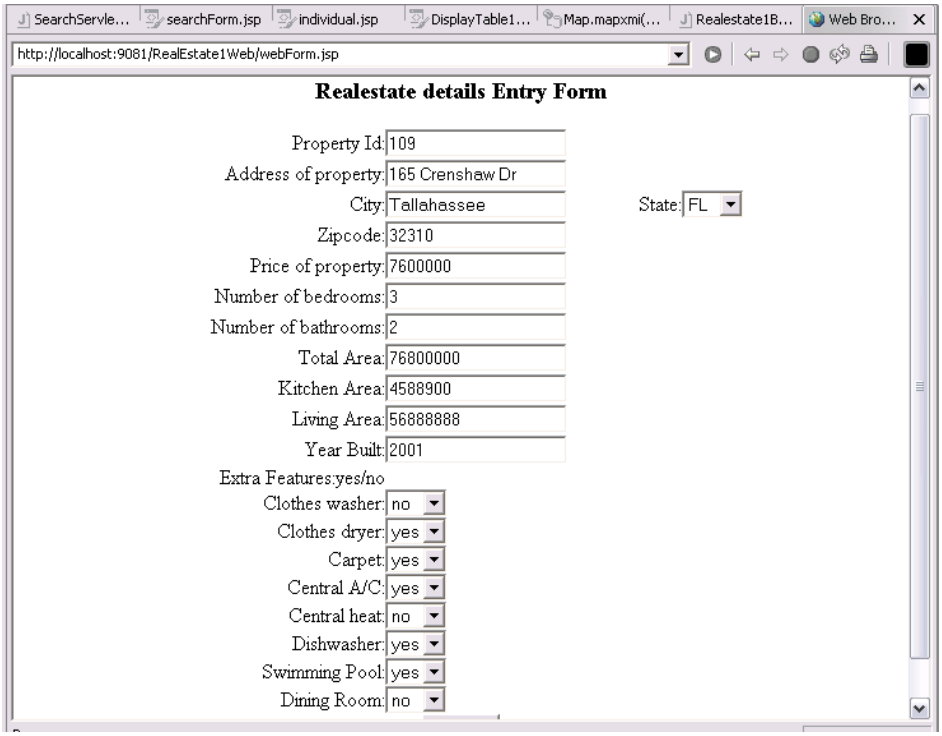
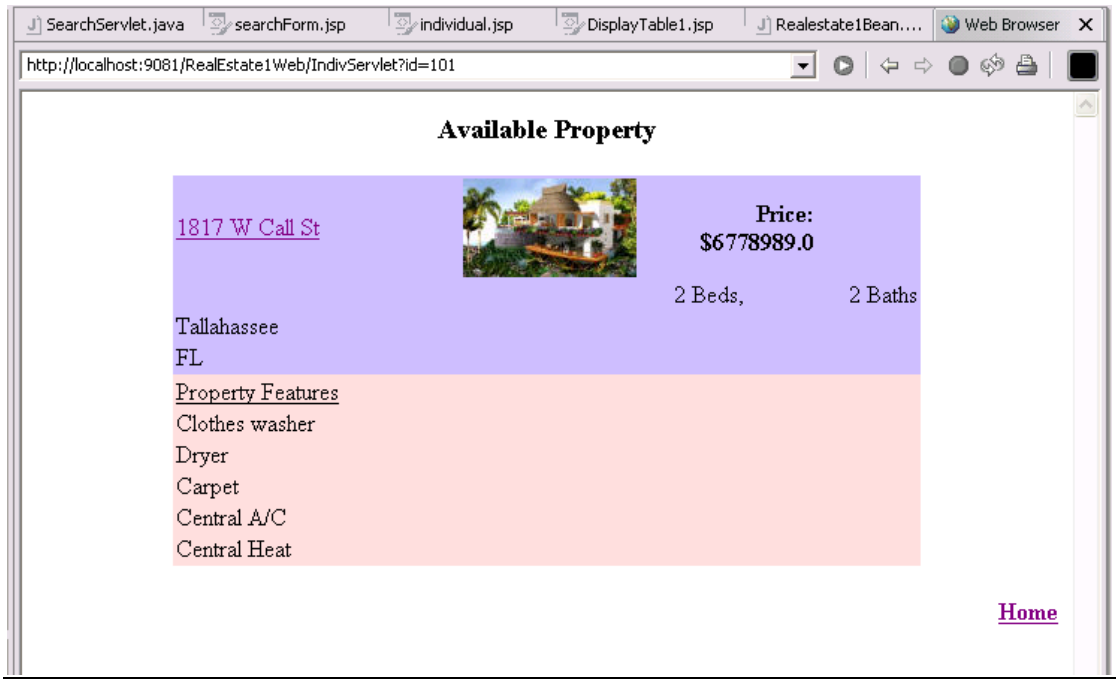


Figure 18: DisplayTable1.jsp page

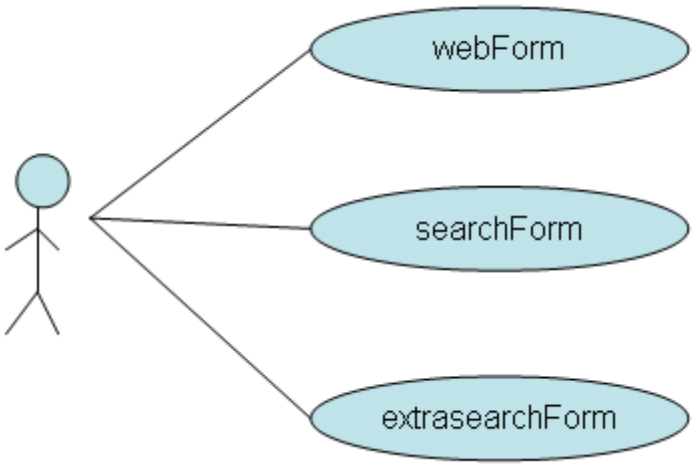
The screenshot in figure 18 shows the results of a search. We can observe that it displays the address, location, price, number of bedrooms, number of bathrooms and a picture of the property. On Clicking on the address link of the available properties, the specified property is displayed along with the features associated with it as shown in Figure 19 below.

Figure 19: individual.jsp page



Use Cases Diagram

Figure 20: Use Cases



8. Web Services and Future Extensions to Project

8.1 Generation of Web Service:

This project can be changed to a web service by following particular steps as required in Websphere. This will generate some WSDL documents and it can be tested using a proxy. A WSDL (Web Services Description Language) document describes where the web service can be deployed, the interface and the operations the service provides. There can be several WSDL documents which can be combined to form a single logical definition. Step-by-step procedure for creation of Web Service and WSDL documents[3]:

1) First, go into the J2EE hierarchy view, select the Realestate1 enterprise bean and go to File -> New -> Other to select Web Services.
2) A list of web services wizards are provided out of which we have to select Web Service wizard.
3) Go through the wizard steps and see that the checkboxes to generate a proxy, test a proxy are selected.
4) Also start the web services in web project and publish the web services to a UDDI registry.
5) One of the pages in the wizard is the Web Service Deployment Configuration page, which allows us to select a runtime protocol. IBM soap is selected form the list of available runtime environments.
6) The Web Service EJB configuration page allows us to type in information related to EJB deployment like the JNDI name of the bean, remote and home interface of the bean, etc
7) A URI is automatically created by the wizard for the bean selected. The default base URI is http://tempuri.org/
8) By clicking 'generate a client proxy', proxy code is created which includes a procedure to call the remote interface of the web service.
9) Once proxy is created, The application can be tested using the Universal Test Client (UTC).

Thus, by following the above steps, an web application can be deployed into a web service. The web service can be registered in the Public UDDI (Universal Description Discovery and Integration) Registry list.

8.2 Future Research (extensions to project):

1) A login and password database could be developed with seller or customer account information so that each have their right privilege or deleting, updating properties they have added to the property database.

[3] Websphere Studio Application Developer Help

2) The company's logo also could be added to each property to keep track of the seller.
3) There could be a way to keep track of required fields and giving an error message in case all fields are not entered, struts could maybe used for this purpose to validate input forms.
4) The project could be extended with a way to check if the city exists and in the selected state. I guess a new database with all the cities and states would have to be added so that the input city and state could be matched with that in the database for validity.
5) One can improve on the script to upload pictures into the database or at least into the file system. This could be done by future students working on this topic.

References

1. Websphere Studio Application Developer (WSAD) http://www-306.ibm.com/software/awdtools/studioappdev/

2. Kyle Brown, Gary Craig, Enterprise Java Programming with IBM Websphere, IBM Press

3. IBM DB2 reference, http://www-306.ibm.com/software/data/db2/

4. Websphere Information center, http://publib.boulder.ibm.com/infocenter/

5. Web Developer's Journal, Article by Ted Brockwood, Java Beans, http://www.webdevelopersjournal.com/articles/beans.htm

6. Enterprise Java Beans by Gopalan Suresh Raj http://my.execpc.com/~gopalan/Java/ejb.html

7. Java 2 Platform, Enterprise Edition, http://Java.sun.com/j2ee/

8. Java Server Pages technology, http://Java.sun.com/products/jsp/

9. Packaging Applications, http://Java.sun.com/j2ee/1.4/docs/tutorial/doc/Overview5.html

10. Enterprise Messaging using JMS and IBM Websphere, Yusuf, Kareem, 2004

11. J2EE Container, http://Java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts.html

12. http://www.webopedia.com/TERM/

13. Java Servlet Programming, Hunter, Jason, 1998, Electronic book accessible at http://www.netLibrary.com/urlapi.asp?action=summary&v=1&bookid=24210