College of Engineering
Computer Engineering Department
CMPE 275 – Enterprise Software Components

Project 1 Part B Report

April 03, 2012

Table of Contents

Introduction

❖ Goal of the project:

The following are the main goals of the project

- To develop a system to understand the Component Based architecture and working with container managed components.
- To manage the components using the Enterprise Java Beans, working with various kinds of beans like session beans, entity beans and message-driven beans.
- To Understand Inversion of Control through Spring Framework that helps reduce dependencies.
- Decoupling the functionalities in component based system aims at achieving system granularity with better performance.
- Familiarizing with the functionalities of Struts2 Framework which is an open-source implementation for MVC, for the flexible GUI development.

❖ Purpose of System:

The purpose of the system is to implement a Stock Trading Portal that provides the following functionalities that allow a user to:

- Get Stock temporal data i,e the user can search for a particular industrial sector that in turn returns the companies that fall under that sector and download the stock data for that company in XML,CSV and JSON formats
- Get the Spatial Data i,e the user can search for companies in a particular location which provides a list of companies in that location and allows the user to download the stock data in XML,CSV and JSON formats
- Besides these functionalities the user can directly enter the ticker value of a company and retrieve the companies'
  - Historical data – all the data pertaining to that particular company
  - Snapshot Data – all the data pertaining to that particular company in a given date range
- Other functionalities include:
  - Analyzed data: Returns stock data for top 5 companies for a particular date
  - Real Time Data: shows the percentage change in stocks for a particular company on a given date

❖ Working of the system:

The user can upload or download stock data. To upload the data, the user gives the name of the file. The file is read and the data is pushed on to the queue. This data is validated using a message driven bean and is finally sent to the database. To download stock data the user can either enter a ticker value directly and select the kind of data to be downloaded or look up a ticker value based on industrial sector or the location, choose a ticker and download the historical or snapshot data in XML,CSV or JSON formats.

The user selects functionality on a JSP page. The JSP forwards the request to an action file which in turn calls an interceptor to verify if the action requested can be performed. The request is then sent to a java file which calls the method on the server through a remote object for performing the requested action.

System Design

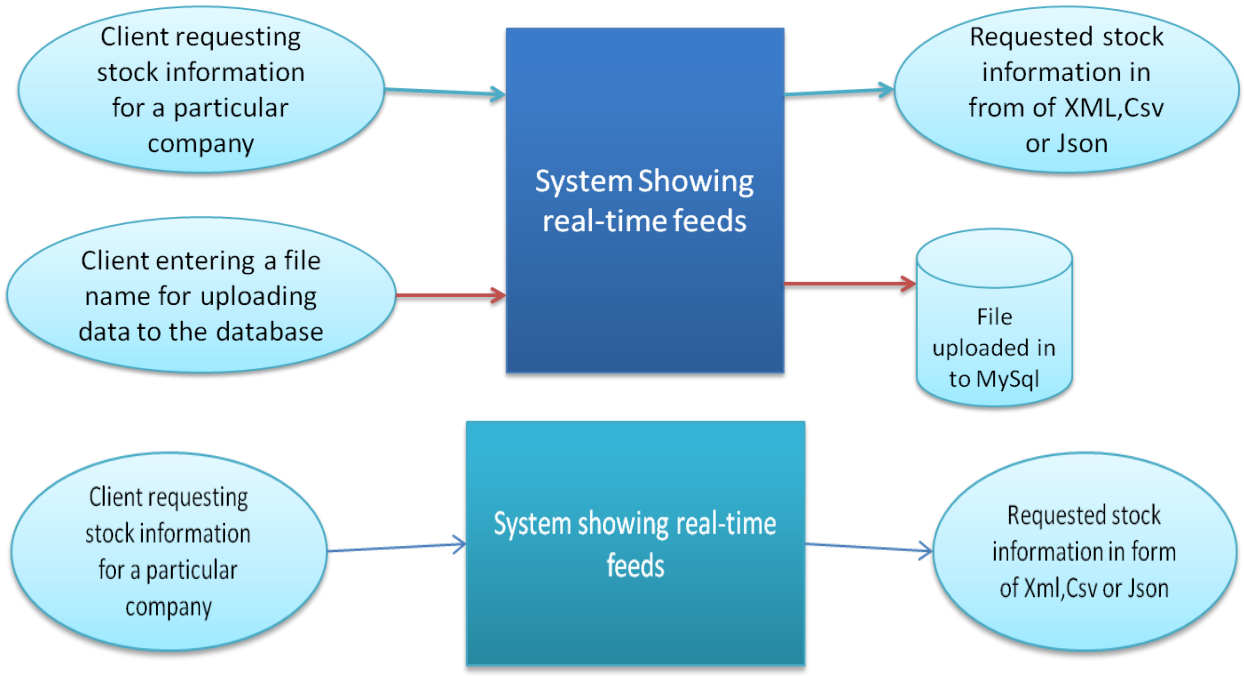


Figure 1. Logical Design

Figure 1. Shows the logical design of the Stock Trading System The client will request the stock information of a particular company to the system. The output will be the requested stock information in the format of XML, CSV or JSON.

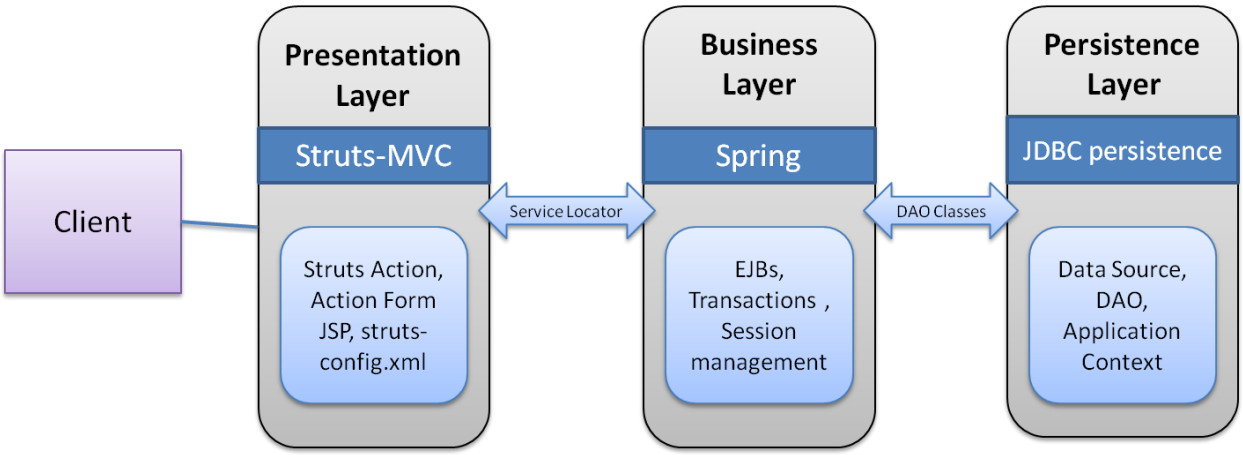


Figure 2. Physical Design

Figure 2. Shows the physical design of the Stock Trading System. The presentation layer consists of Struts2 framework which is a MVC framework. The business logic layer consists of Spring Framework which is an implementation of Inversion of Control pattern in EJB 3.0. Then the persistence layer is JDBC persistence which consists of the data access objects and application context for the persistence.

❖ Presentation Layer : Struts - MVC

Apache Struts2 is an extensible framework for creating enterprise-ready Java web applications. The framework is designed to streamline the full development cycle, from building, to deploying; to maintaining applications over time.Struts2 is a pull-MVC framework. i.e. the data that is to be displayed to user has to be pulled from the Action. Struts2 supports annotation based configurations which are easy to create and more intuitive. Action class in Struts 2 acts as the model in the web application. Struts2 also comes with power APIs to configure Interceptors that reduce greatly the coupling in application. The view part of Struts 2 is highly configurable and it supports different result-types such as Velocity, FreeMarker, JSP, etc.

The MVC Architecture:

Struts is an open source framework used for developing J2EE web applications using Model View Controller (MVC) design pattern. It uses and extends the Java Servlet API to encourage developers to adopt an MVC architecture. Struts framework provides three key components:

1. A **request** handler provided by the application developer that is used to mapped to a particular URI.
2. A **response** handler which is used to transfer the control to another resource which will be responsible for completing the response.

3. A **tag library** which helps developers to create the interactive form based applications with server pages.

Struts provide you the basic infrastructure for implementing MVC allowing the developers to concentrate on the business logic.

The Struts Model components:
The model components provide a model of the business logic behind a Struts program. It provides interfaces to databases or back- ends systems. Model components are generally a java class. There is not any such defined format for a Model component, so it is possible for us to reuse Java code which is written for other projects. We should choose the model according to our client requirement.

The Struts Controller Components:
Whenever a user request for something, then the request is handled by the Struts Action Servlet. When the ActionServlet receives the request, it intercepts the URL and based on the Struts Configuration files, it gives the handling of the request to the Action class. Action class is a part of the controller and is responsible for communicating with the model layer.

The Struts View Components:
The view components are responsible for presenting information to the users and accepting the input from them. They are responsible for displaying the information provided by the model components. Mostly we use the Java Server Pages (JSP) for the view presentation. To extend the capability of the view we can use the Custom tags, java script etc.

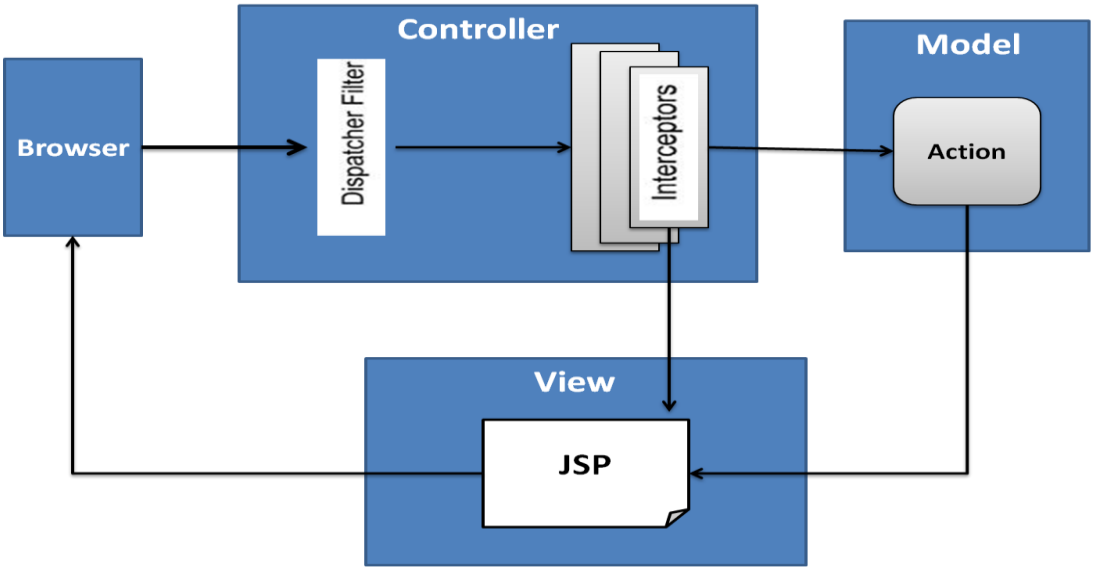


Figure 3. Struts MVC

1. **User Sends request:** User sends a request to the server for some resource.
2. **Filter Dispatcher determines the appropriate action:** The Filter Dispatcher looks at the request and then determines the appropriate Action.
3. **Interceptors are applied:** Interceptors configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.
4. **Execution of Action:** Then the action method is executed to perform the database related operations like storing or retrieving data from the database.
5. **Output rendering:** Then the Result renders the output.
6. **Return of Request:** Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.
7. **Display the result to user:** Finally the control is returned to the servlet container, which sends the output to the user browser.

❖ Business Layer : Spring Inversion Of Control

The middle component of a typical web application is the business or service layer. One of the most popular frameworks in this space is **Spring.** This is referred to as micro container that has a very small footprint and determines how to wire the objects together. This framework works on a simple concept of dependency injection (also known as Inversion of Control). Spring also allows a sophisticated form of constructor injection as an alternative to setter injection as well. The objects are wired together by a simple XML file that contains references to objects such as the transaction management handler, object factories, service objects that contain business logic, and data access objects (DAO).

The business layer should be responsible for the following:

- Handling application business logic and business validation
- Managing transactions
- Allowing interfaces for interaction with other layers
- Managing dependencies between business level objects
- Adding flexibility between the presentation and the persistence layer so they do not directly communicate with each other
- Exposing a context to the business layer from the presentation layer to obtain business services
- Managing implementations from the business logic to the persistence layer

Problems with JNDI lookup

With plain JNDI, lookup can be done in classes' static initializer, in the constructor or any method including the finalizer. Thus there is no control. With JNDI used under EJB control, and concerning only components looked up from that bean's sisters (implicitly under the same container's control), the specification indicates that the JNDI lookup should only happen at a certain moment in the startup of an EJB application, and only from a set of beans declared in ejb-jar.xml. Hence, for EJB containers, the control element should be back.

That means many bean containers have no clue as to when lookups are actually being done, and applications work by accident of deployment. Allowing it for static is truly evil. It means that a container could merely be looking at classes with reflection in some early setup state, and the bean could be going off and availing of remote and local services and components.

Design pattern: Dependency Injection

Dependency Injection previously known as Inversion of Control (IoC) pattern is a software design pattern and set of associated programming techniques in which the flow of control of a system is inverted in comparison to the traditional interaction mode, which means instead of the application calling the framework for services (as is the case with traditional patterns) it's the framework which calls the components as specified by the application.

Java components / classes should be as independent as possible of other Java classes. This increases the possibility to reuse these classes and to test them independently of other classes (Unit Testing). To decouple Java components from other Java components the dependency to a certain other class should get injected into them rather that the class itself creates / finds this object.

A class A has a dependency to class B if class uses class B as a variable. If dependency injection is used then the class B is given to class A via

- the constructor of the class A - this is then called construction injection
- a setter - this is then called setter injection

The general concept between dependency injection is called Inversion of Control. A class should not configure itself but should be configured from outside. A design based on independent classes / components increases the re-usability and possibility to test the software. For example if a class A expects a Dao (Data Access object) for receiving the data from a database you can easily create another test object which mocks the database connection and inject this object into A to test A without having an actual database connection.

Spring framework has two packages supporting this core functionality. These packages are:

- org.springframework.beans
- org.springframework.context

Two interfaces which are the building blocks of Spring IoC are:

- **BeanFactory** - The top level interface, BeanFactory takes care of supporting an advanced configuration mechanism to manage objects of any type.
- **ApplicationContext**- ApplicationContext interface is actually a sub-interface of the BeanFactoryinterface. ApplicationContext interface on the other hand is responsible for functionalities like integration to Spring AOP, message resource handling, event prorogation, and other enterprise-centric functionalities.

DI generally favors loose coupling between components. Loose coupling in turn favors:

- More reusable classes and classes that are easier to test
- Systems that are easier to assemble and configure

❖ Persistence Layer : JDBC data access

Persisting and accessing data forms one of the most routine yet core functionalities of any application. The beauty of the Spring Framework is the flexibility it provides. Using the Spring Framework, one can integrate low-level APIs such as JDBC or high-level frameworks such as Hibernate into an application using the same technique - Dependency Injection. In this discussion, the focus will be on using the Spring Framework's data access functionality with JDBC, which forms the basis of all other frameworks.

The first section will focus on the Data Access module (or DAO module) and the services it provides for using JDBC. The second and third sections will detail the steps required to use the DAO module of the Spring Framework. In the last section, an application will be developed that will make use of the steps detailed in the second section to access and persist data.

Using JDBC with Spring

JDBC support in the Spring Framework is part of the persistence (also known as DAO) support provided by the framework. Hence, it is better to know about DAO support before going into the JDBC aspects. The most important feature of DAO support is consistency. Spring supports JDBC through the JdbcTemplate class. The three main subclasses of JdbcTemplate are as follows:

- JdbcTemplate is the most widely used of all the types or flavors. It is the 'lowest-level' type. In other words, all other flavors make use of JdbcTemplate as their basis.
- NamedParameterJdbcTemplate provides a wrapper for Named Parameter queries of JDBC. It wraps the JdbcTemplate to provide a more convenient way of handling parameterized queries instead of the conventional "?" placeholder provided by JDBC, thus making it easier to use multiple parameters.
- SimpleJdbcTemplate combines the most commonly used features of JdbcTemplate and NamedParameterJdbcTemplate. It also makes use of new features of Java 5 such as varargs, generics etc.

Work flow of the System

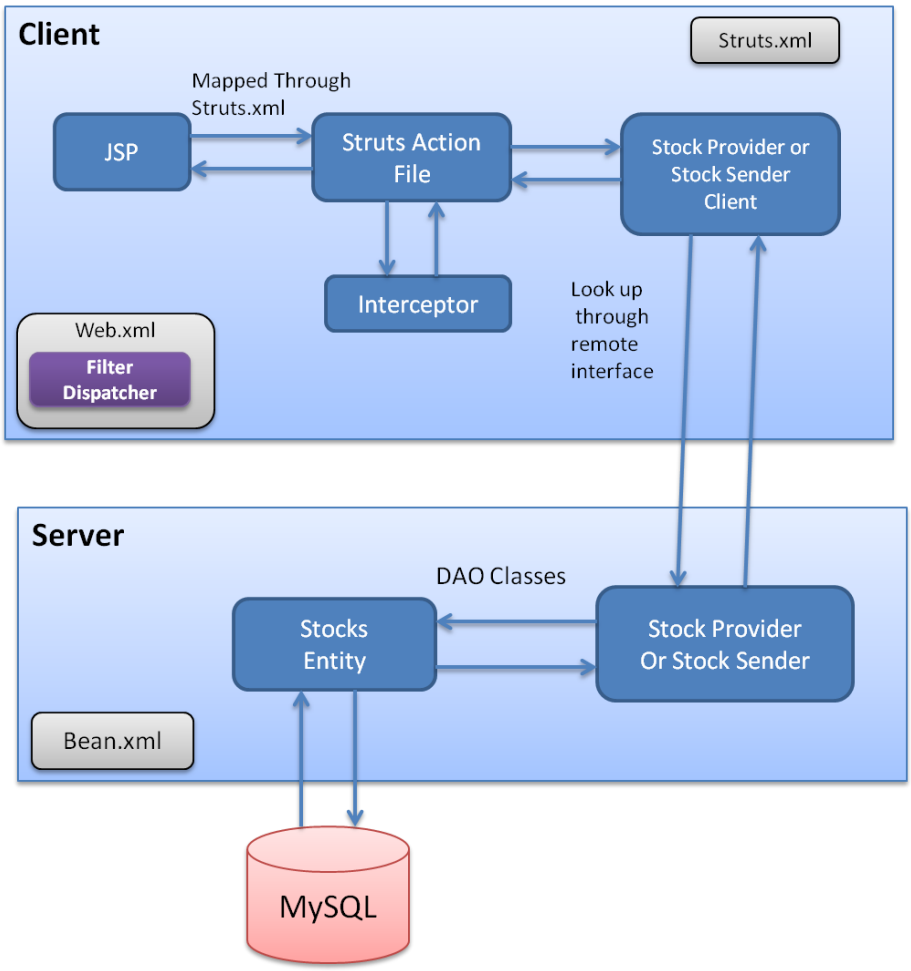
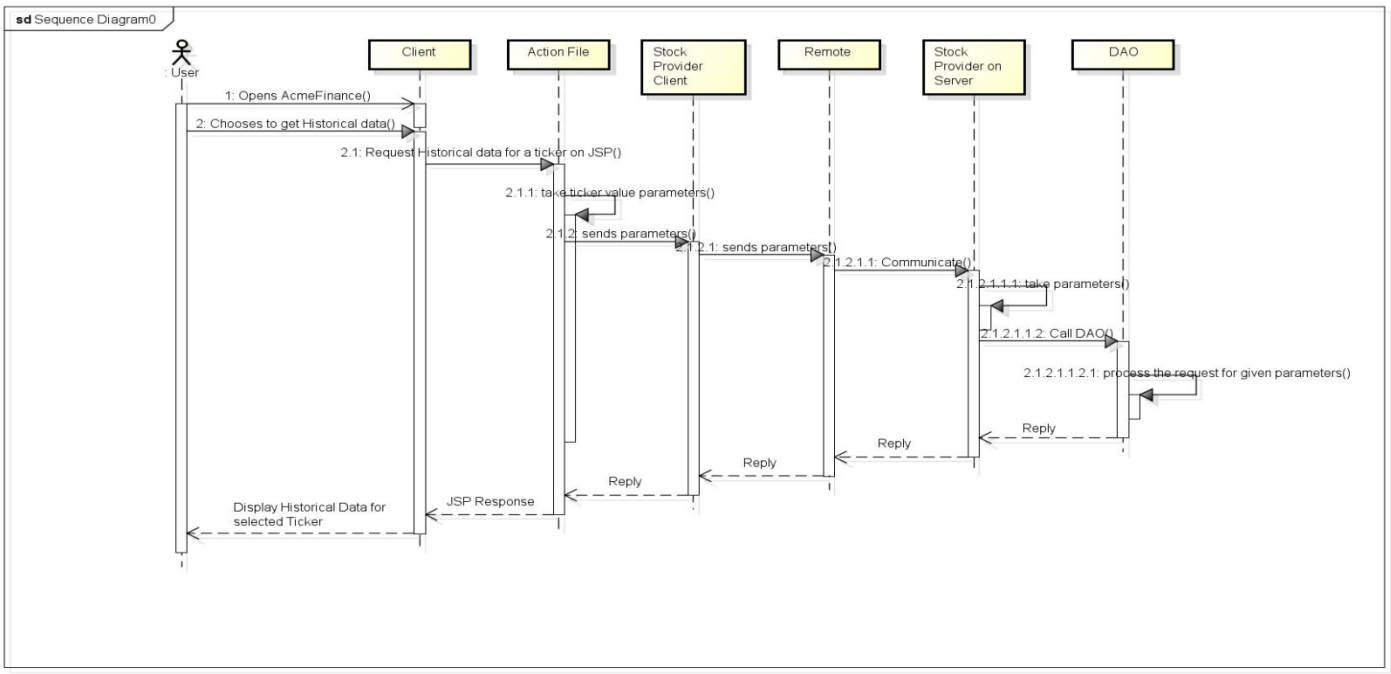


Figure 4. Architectural Diagram of System

The Client has web.xml and struts.xml for configurations. JSP pages are mapped with the struts action file through struts.xml. Web.xml has a filter dispatcher which looks at the request and determines the action. Interceptors are configured for common functionalities such as validations. The action is sent to the client file looks up to server file through remote interface. The server file then invokes the DAO classes and communicates with entity bean and thus with the database. The response is sent to the client for display.

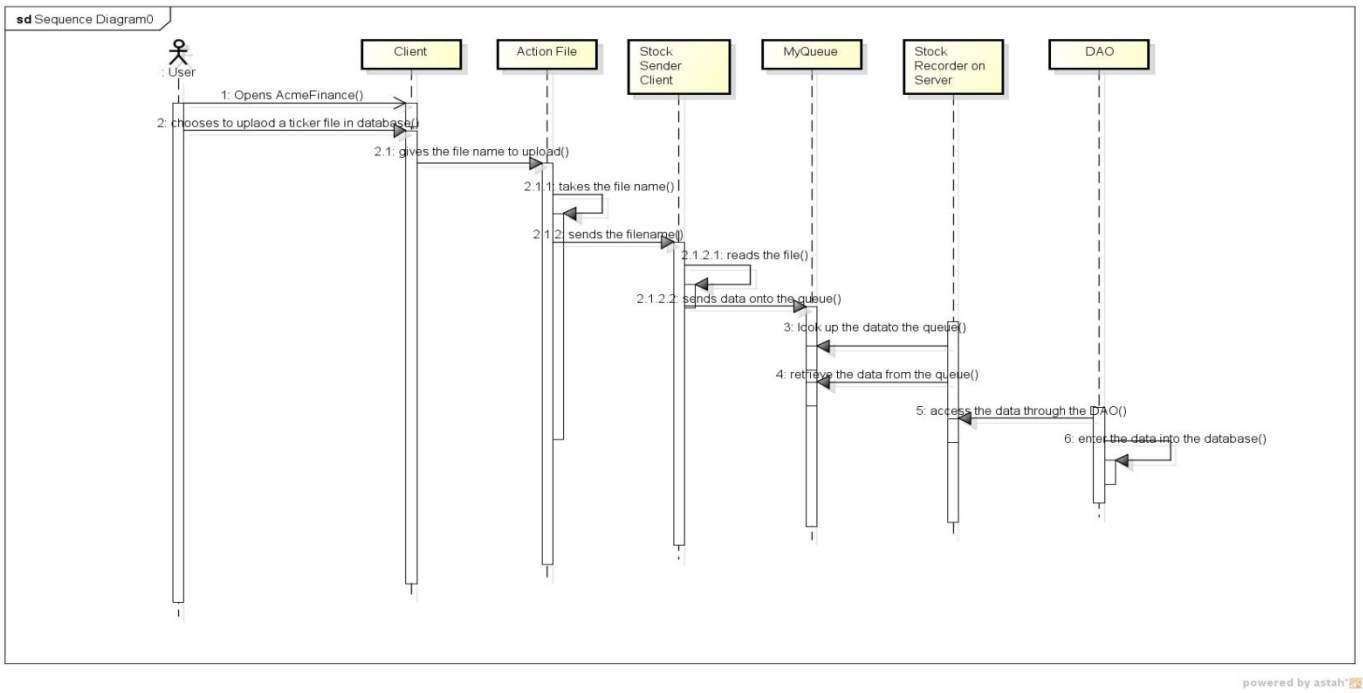
❖ **Sequence Diagram 1 : Stock Provider**

This is the process for data retrieval



❖ **Sequence Diagram 2: Stock Sender**

This is the process for uploading data



Reasons Why Choosing This Design

1. Spring is explicitly designed to resolve some of the real-world problems of JEE, such as complexity, poor performance, testability, and much more.
2. Spring framework lets you apply AOP (aspect-oriented programming technique) rather than object-oriented code.
3. Struts enable clean separation of code between functionality and presentation, allowing developer to concentrate on the Business Logic.
4. Struts promotes modularity i.e. allows modular development and easy integration with new components.
5. Being modular Struts is also maintainable, which yields reduced code complexity and less duplication.
6. The Spring architecture allows you to connect Struts as your Web framework to Spring-based business and persistence layers.
7. Spring provides more control on struts actions. That may depend on the method of integration you choose.

Performance and Scalability

The performance problems can be tracked down to the persistence layer. Good performance here is often a function of sound design choices.

- In our application we had to read a (very large) Excel spreadsheet, perform a simple transformation, and then insert rows into a SQL Server table, the operation was taking three hours, despite best attempts at tuning. By taking advantage of SQL Server linked queries to treat the spreadsheet as a database accessed via JDBC, and moving the logic into a stored procedure, the whole operation was comparatively fast.
- Batch applications bring additional consideration since memory usage becomes very important – we have avoided consuming large amounts of memory and causing costly garbage collection pauses. Stream-based algorithms are the best approach here. We have used iterators rather than collections for example.

Performance and Scalability of our system is based on, how fast the server is able to process the client request and send the result back to client. It also depends on the ability to manage data persistence and relationship between data elements between application and database. Our system is using EJB3 and JDBC which eased the persistence of data. Java annotations and/or XML deployment descriptor is used for the mapping between Java objects and a relational database.

Our system is scalable because it can handle growing amounts of request. Scalability and performance are interdependent. Because of the improved performance as mentioned below, our system is more scalable in terms of data management.

Classes Explained

StocksAllDemo	Description
com.foo.Stocks	
StockProvider.java	A session bean that uses application context and communicates with the DAO
StockProviderLocal.java	Local interface for Stock Provider bean
StockProviderRemote.java	Remote interface for Stock Provider bean
StockRecorder.java	A message Driven bean to receive the data from the queue ,validate and insert it into the database
StockProviderTest.java	Unit test cases for StockProvider.java
com.foo.xml	
ToXml.java	A java file used for converting retrieved data into xml format
ToXmlLocal.java	Local interface for ToXml bean
ToXmlRemote.java	Remote interface for ToXml beam
ToXmlTest.java	Unit test cases for ToXml.java
Similar package as com.foo.xml and similar classes for CSV and JSON	
DAO	
StockDAO.java	A DAO file that consists of queries to retrieve and insert data from and to the database
StocksDAOImpl.java	An Interface to the StocksDAO file
Domain	
Stocks.java	An Entity Bean that consists of setter getter methods and helps in insertion and retrieval of data
com.foo.Interceptors	
LoggingInterceptor.java	Implements the Spring AOP `MethodBeforeAdvice` for general logging functionality which involves tracing entry into a Stock class method. It runs before any Stocks target method is fired, and logs the name of the class method in the log info.

StocksAllClient	Description
com.foo.struts2	
MyClientAction.java	

StocksAllClient	Description
MyInterceptor.java	
com.foo.Client	
StockProviderClient2.java	A java file that connects the front end to the server
StockSenderClient.java	A java fie that takes actually reads a requested file and pushes the data onto the queue
StockProviderClient2.test	Unit test cases for StockProviderClient2.java
StockSenderClient.test	Unit test cases for StockSenderClient.java

Assumptions

The following are the assumptions we have already made:

1. The nature of the client is not known, it can be a simple web client or another consumer interested in analysis and research of data.
2. Nothing is ***hardcoded*** in the system, and the system is easily scalable for any future changed formats of data.
3. The data can increase heavily over a period of time; so to keep the historical data we can use archives and files structure in its future implementations.
4. Data is already available in the database.

Limitations

- If client sends any request other than supported by our system, exceptions are not thrown.
- Our system depends on operating system for read/write of historical data which is stored in file system. If operating system crashes the data would be lost or temporarily unavailable.

Future Work

i. We can make a trading portal in which the client is able to buy and sell stocks along with mutual funds of his choice.
ii. a portal handling the accounts of all the customers and reporting him or her if his/her share or stock is bought or not getting good or a matching deal for his orders.
iii. Notify the user about recent activities in the stock markets, new IPO's launching in the market.
iv. Using the concurrency locks, to ensure that transactions are not made while some transaction is already taking place.

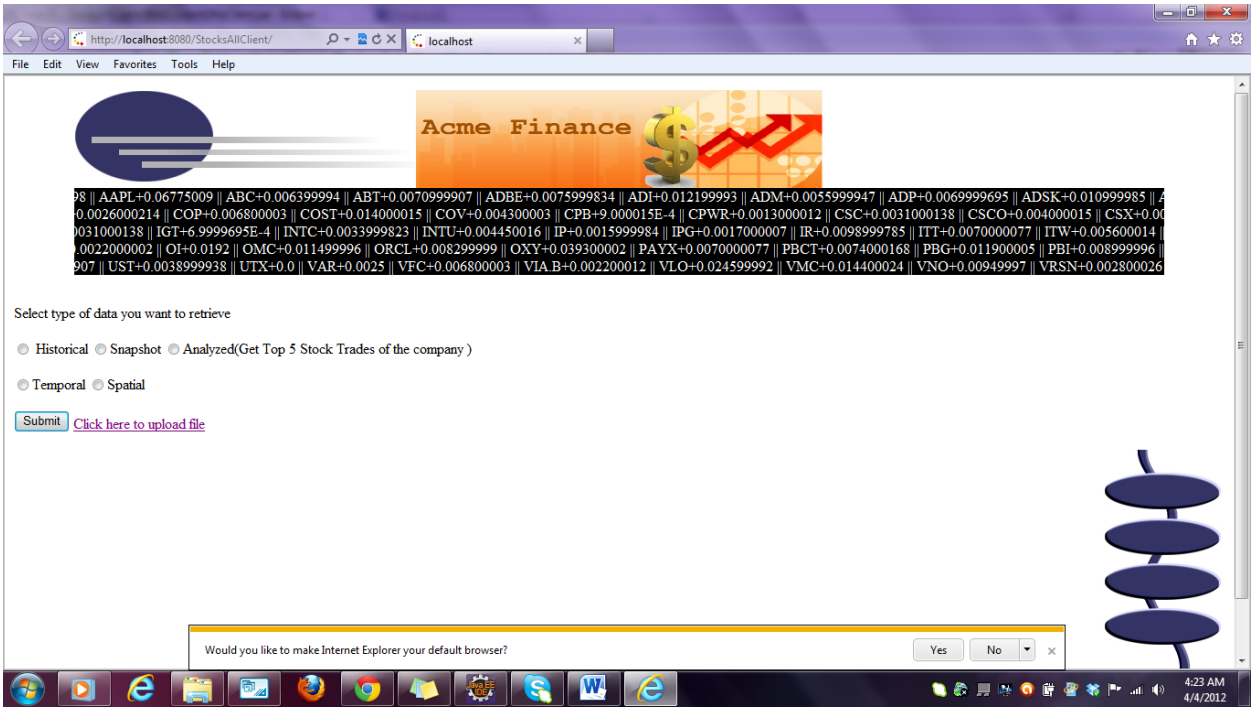
v. We can expand the Spring AOP interceptor class to provide more detailed logging and profiling information. For example we can implement the after advice and throw advice.

Individual Contribution

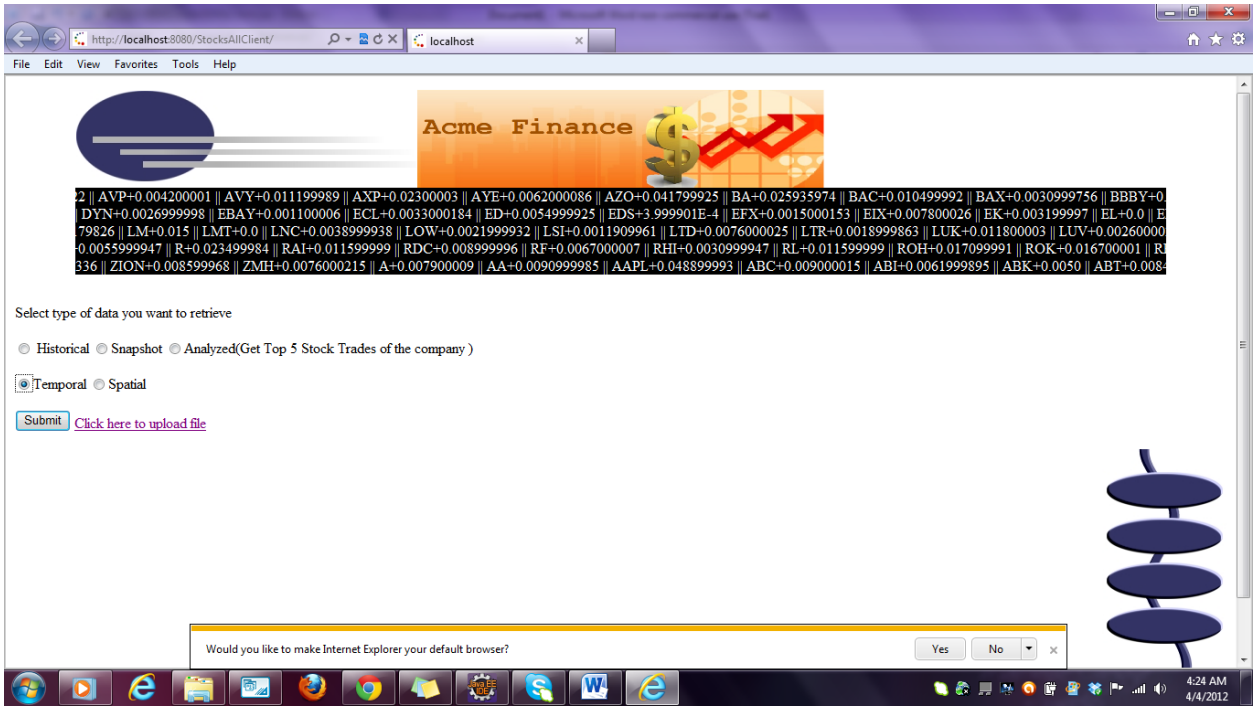
Fan Ieong	Jmeter
Harini Kasturi	Spring, DAO, Spring and Frontend Integration, Junit
Stuti Vora	Documentation
Sudershan Malpani	Spring, DAO, Struts, Spring and Frontend Integration, Ant Build
Vidushi Desai	Struts and Backend Integration, Jmeter

Snapshots

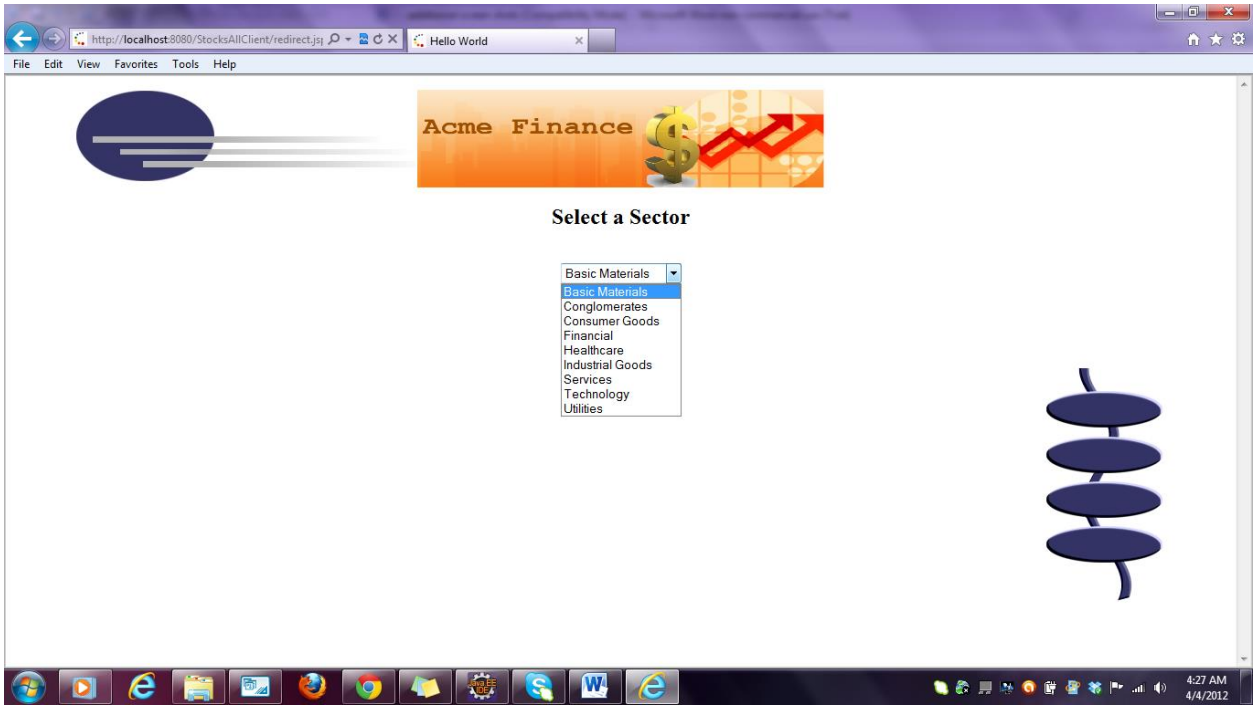
When the StockAllClient project is run it opens the home page:MyClient.jsp



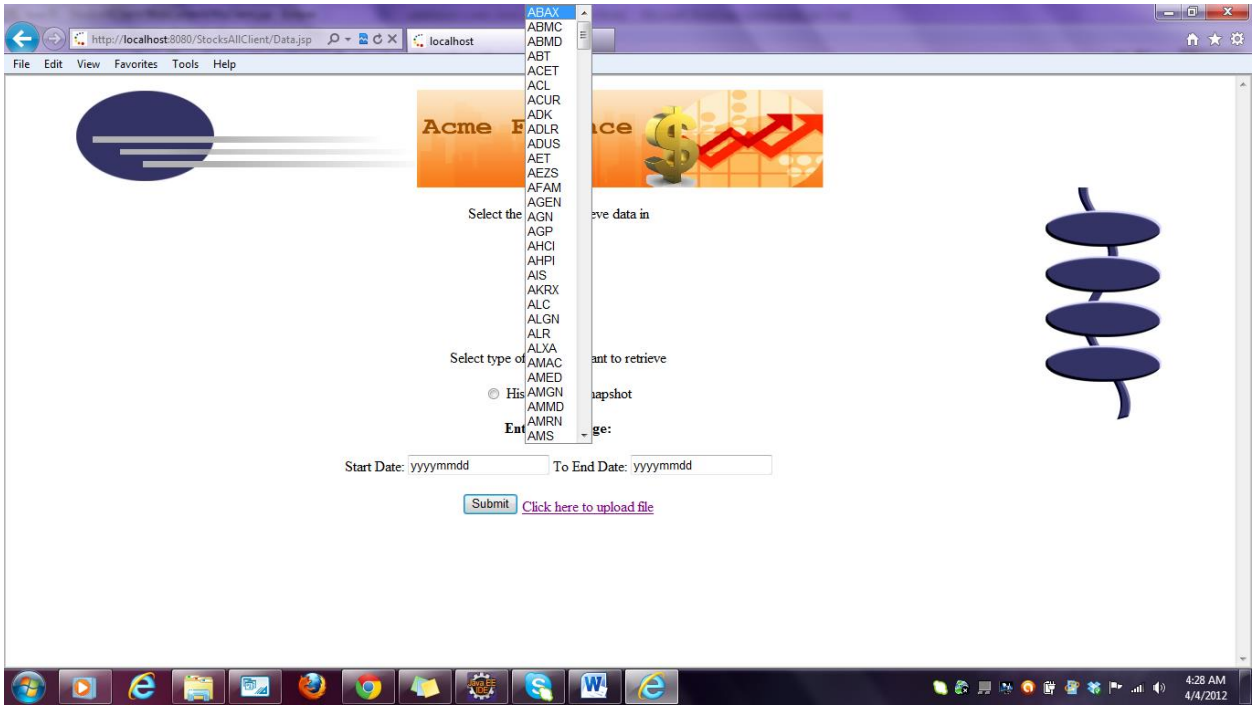
on selecting the temporal data option:



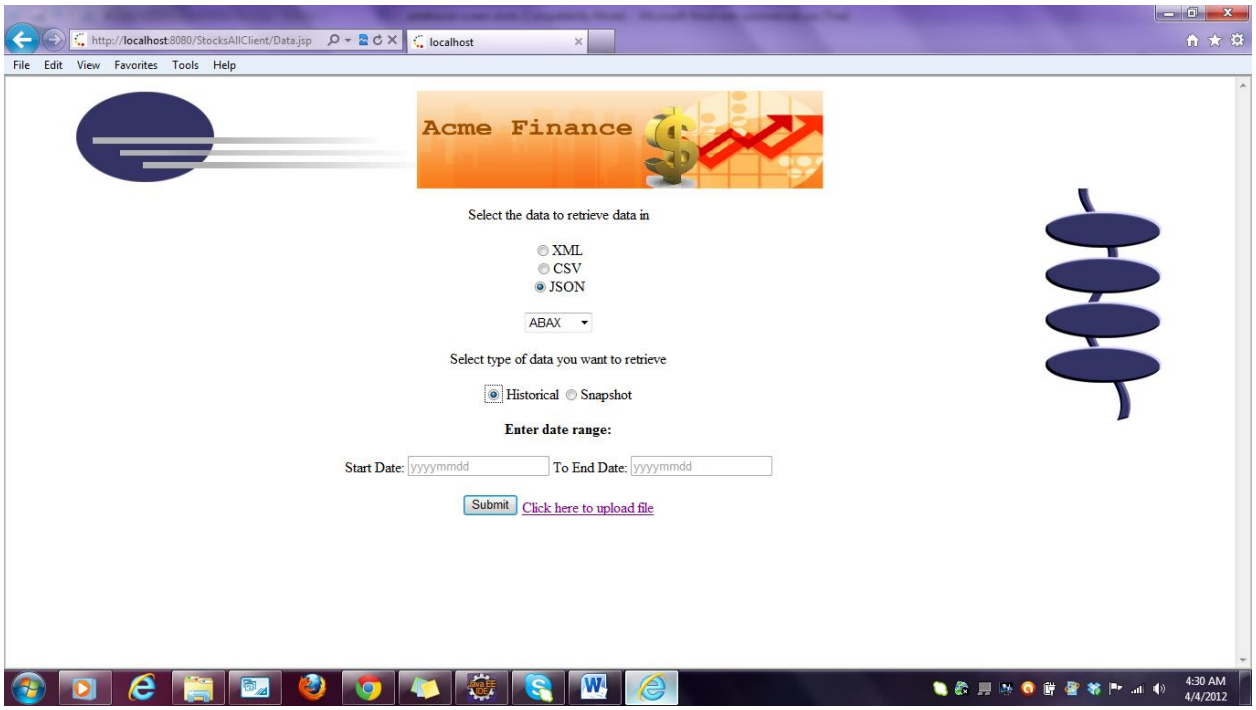
Displays a page which shows a dynamic content of the sectors available in the database:



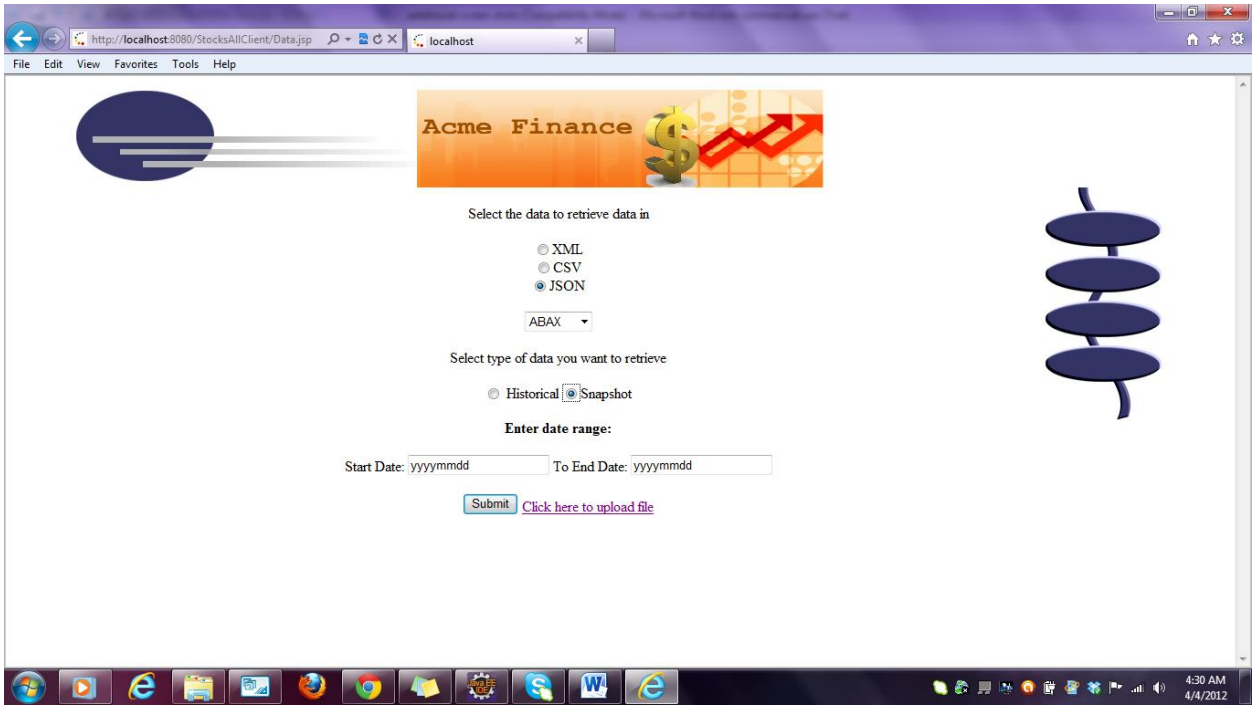
Upon selection of a sector a page with format and type is generated:



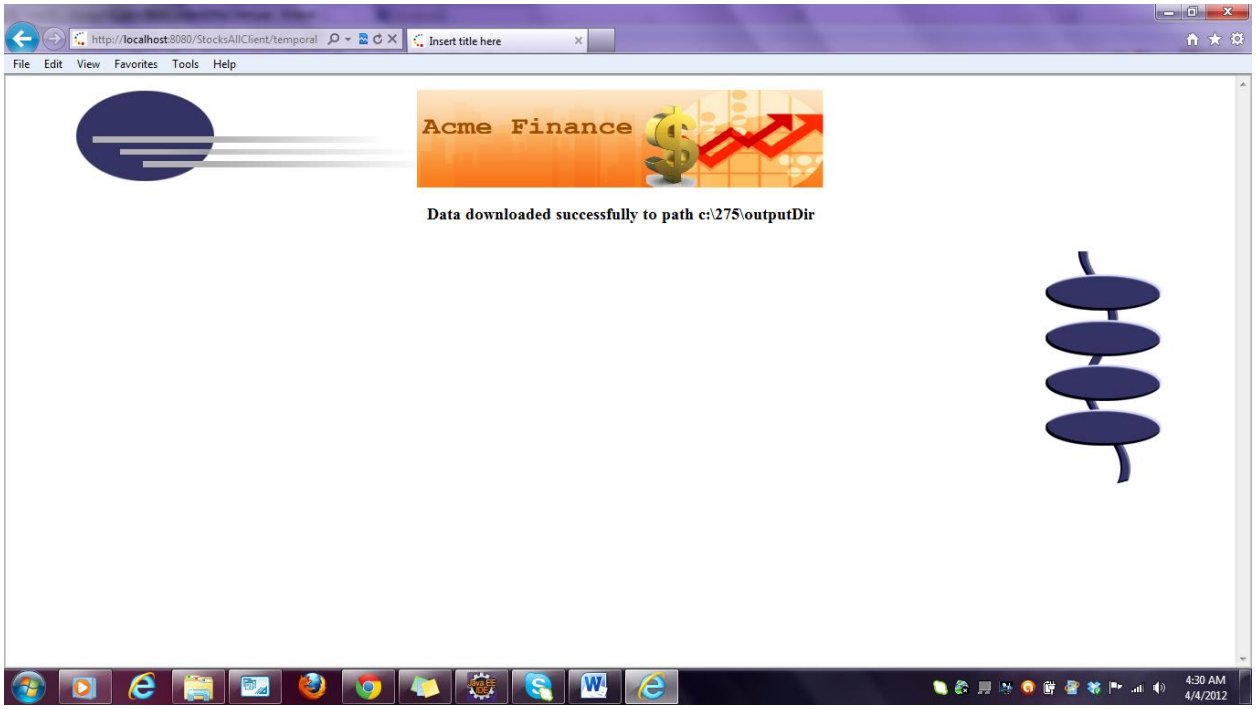
If historical data option is chosen then the text boxes where the date range can be given is disabled as follows:

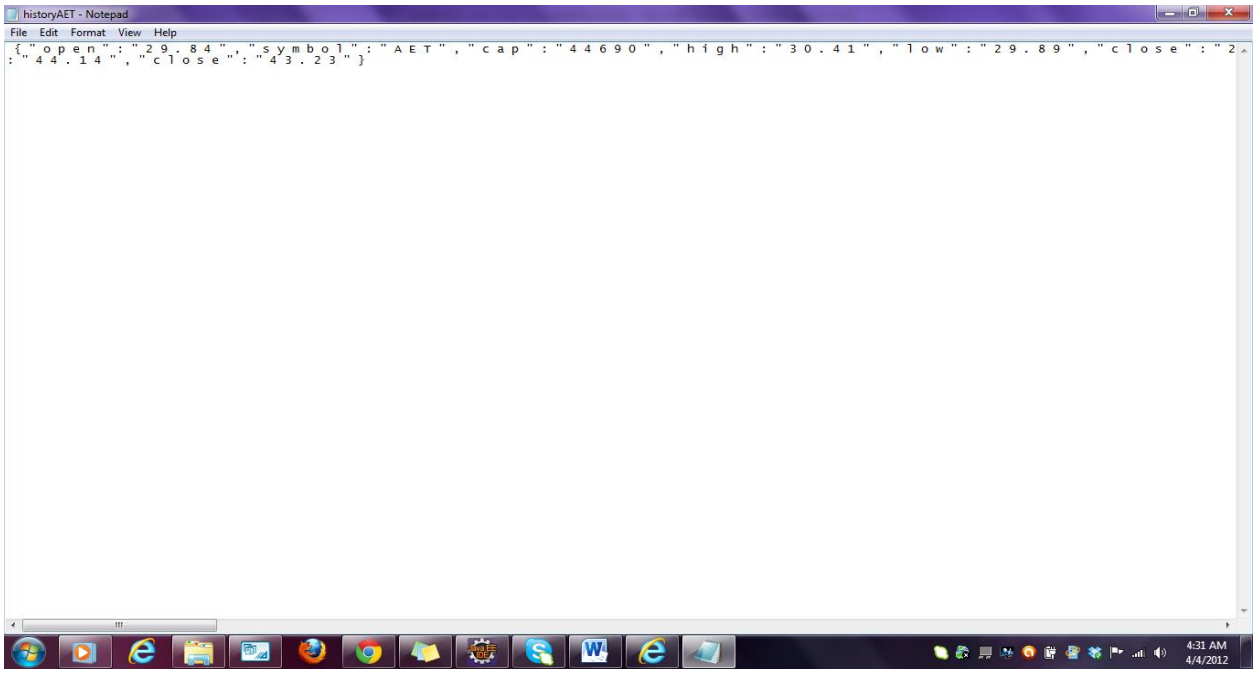


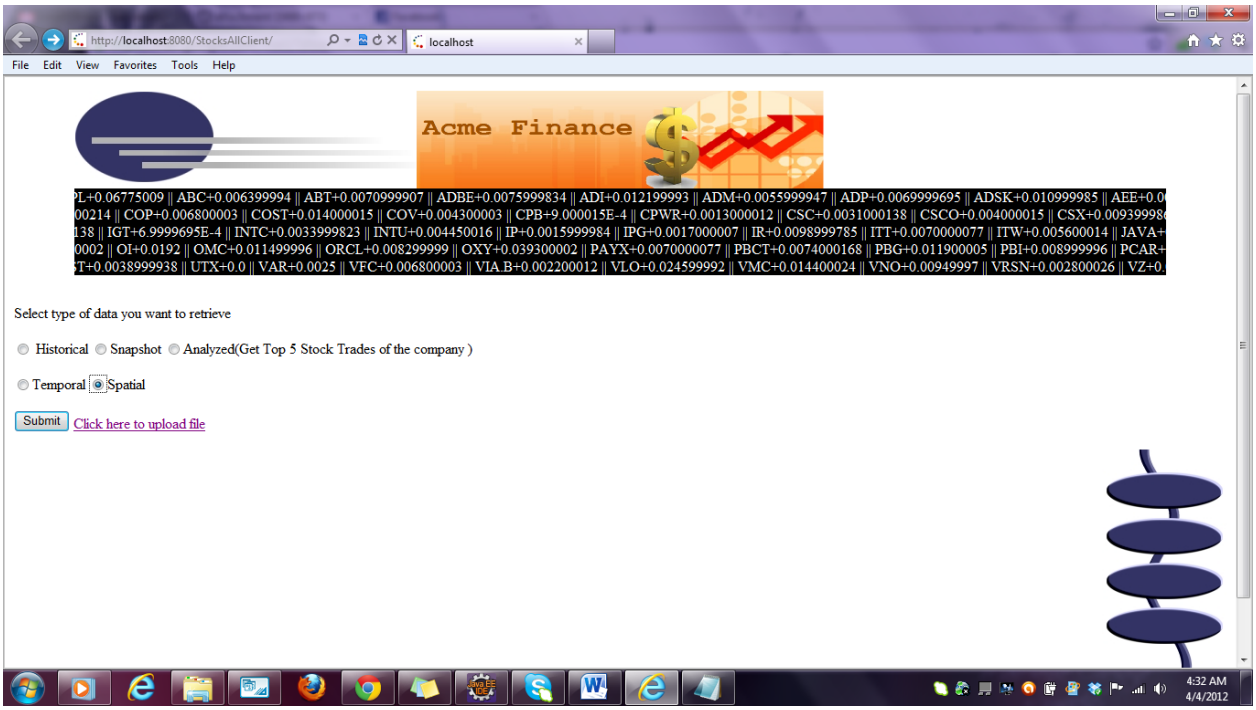
Else the date range is enabled



Upon selecting all values the file is downloaded :

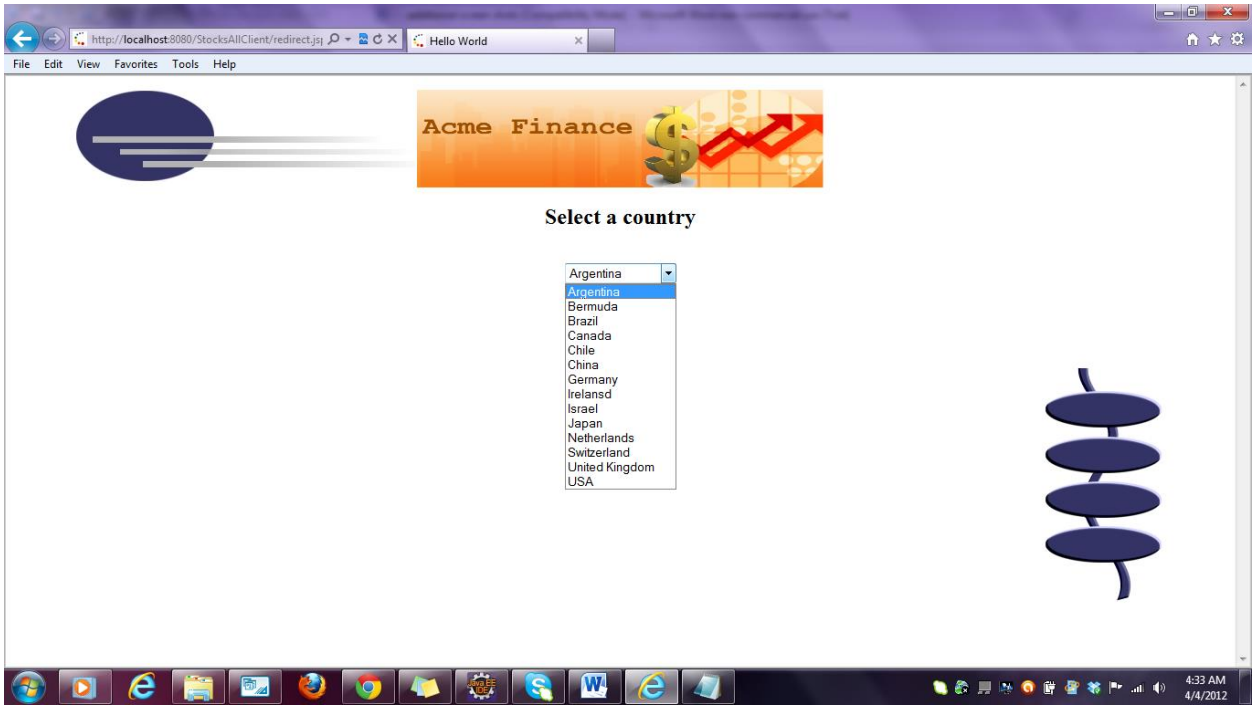




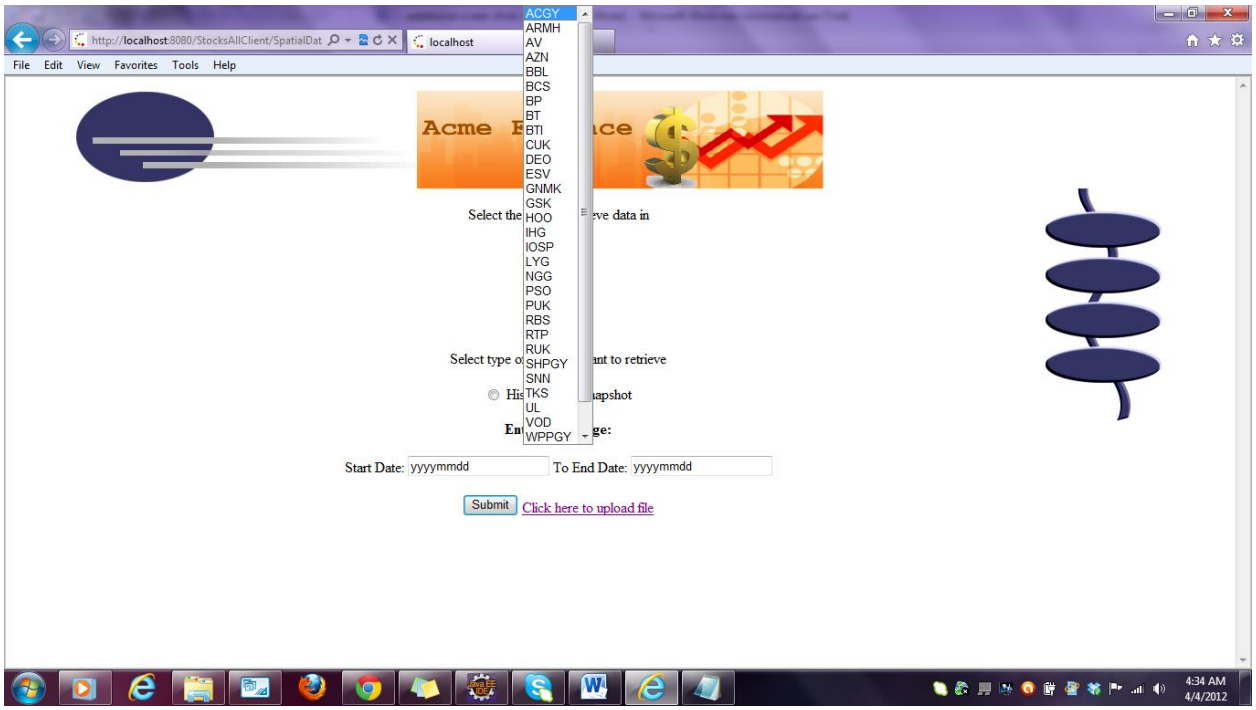


upon selecting spatial data:

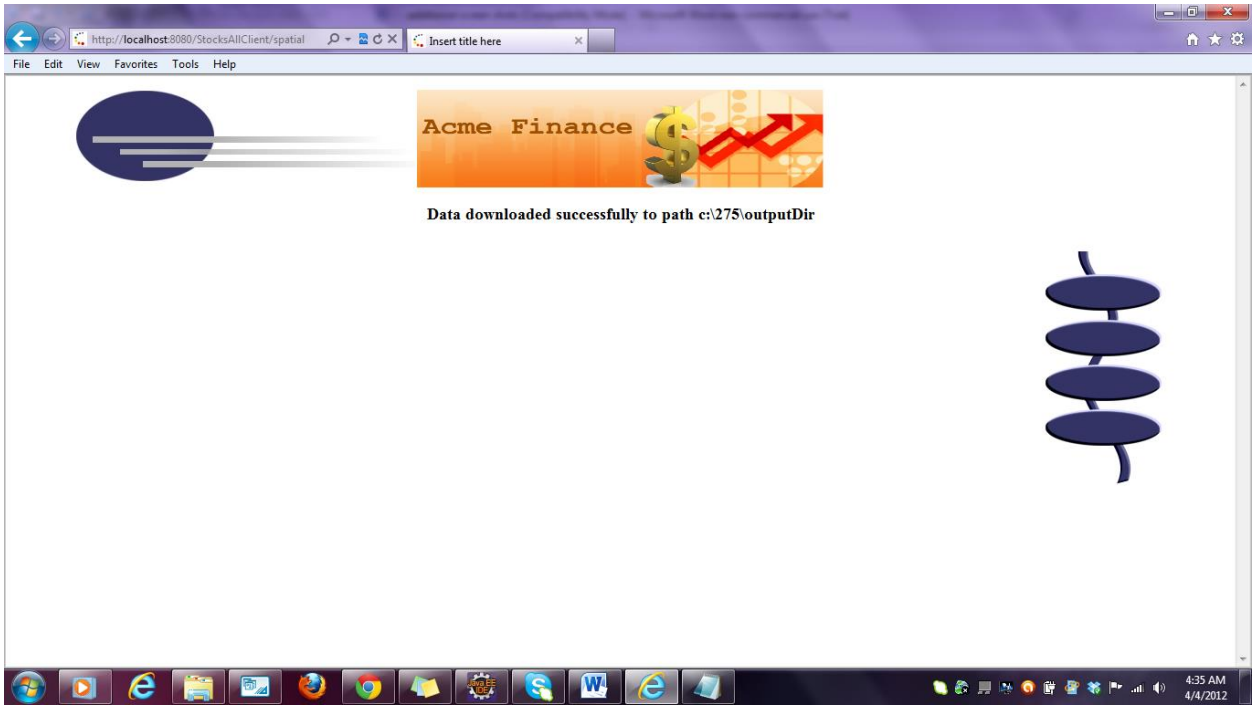
A page with dynamic content of countries from the database is displayed



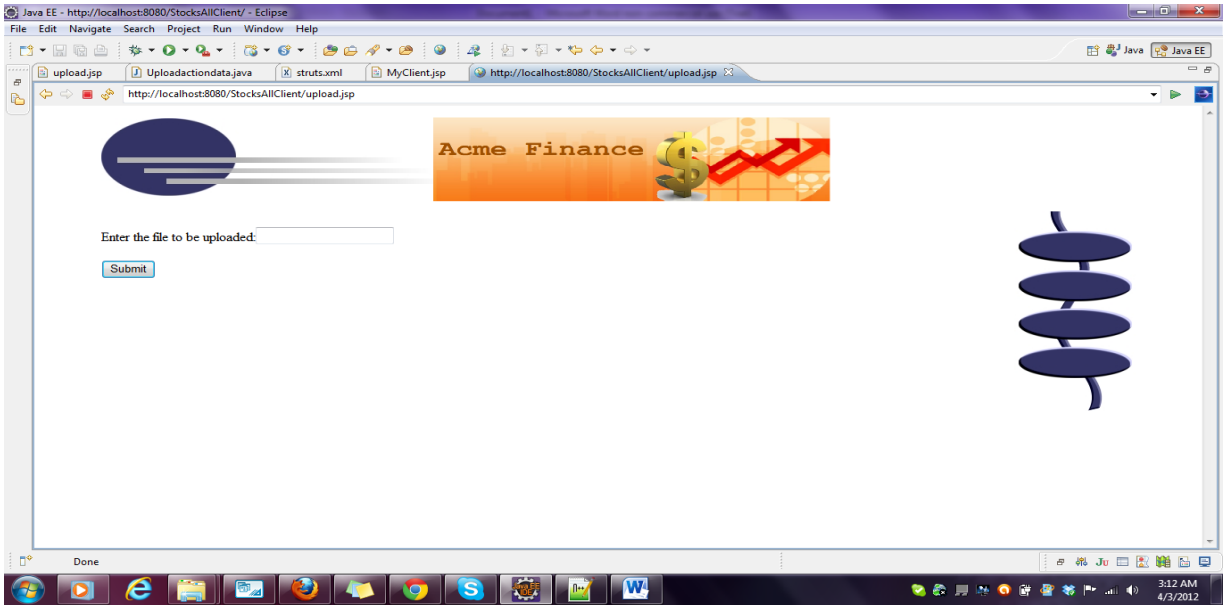
Upon selection a page with format and type is displayed.The ticker values are dynamically loaded based on the country selected



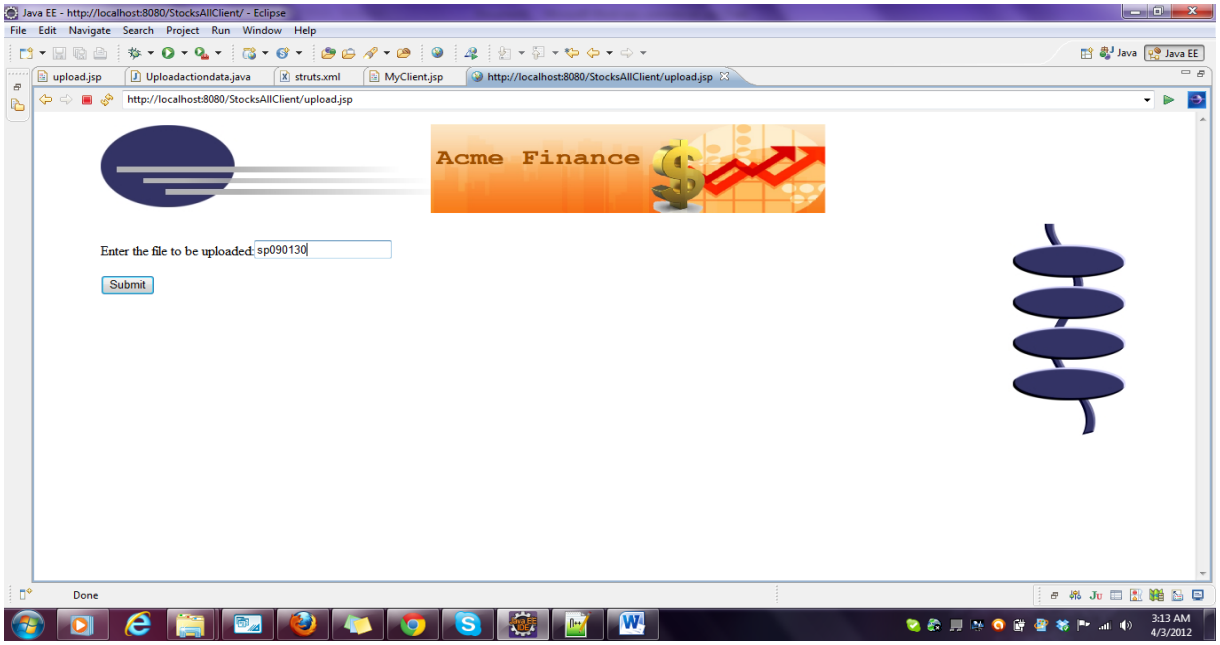
Upon selection of all the required fields the data is downloaded



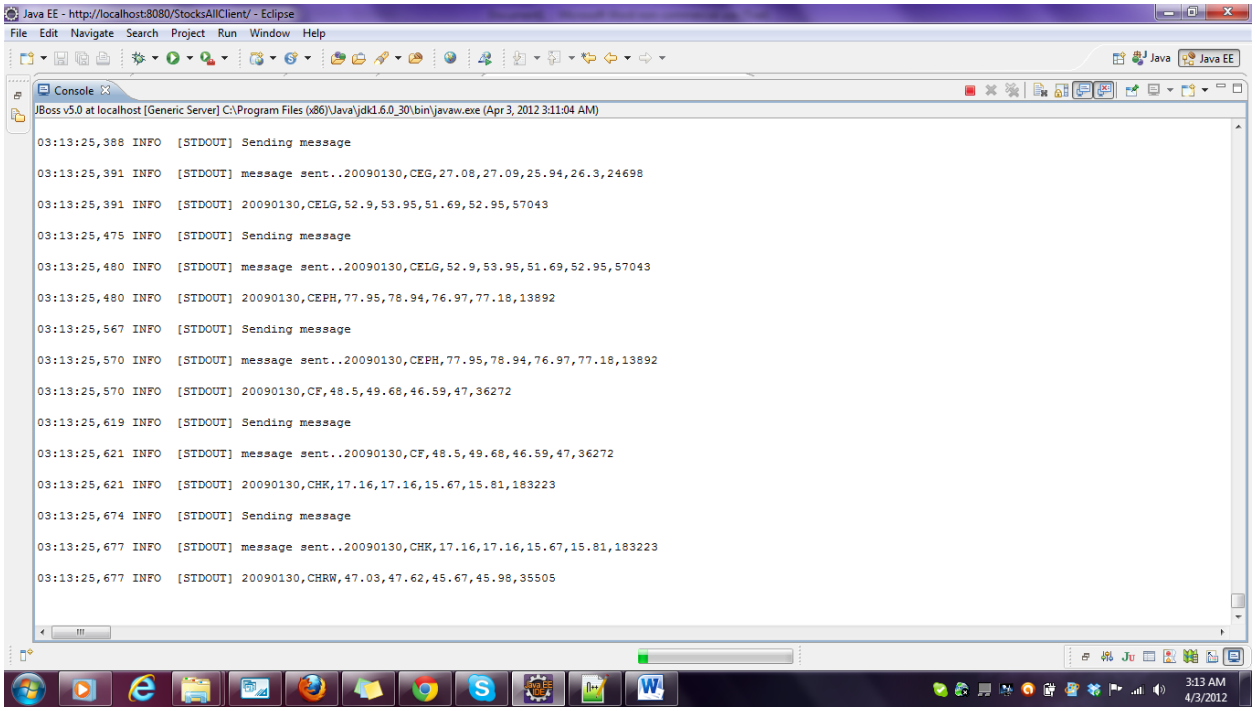
on clicking the upload file link



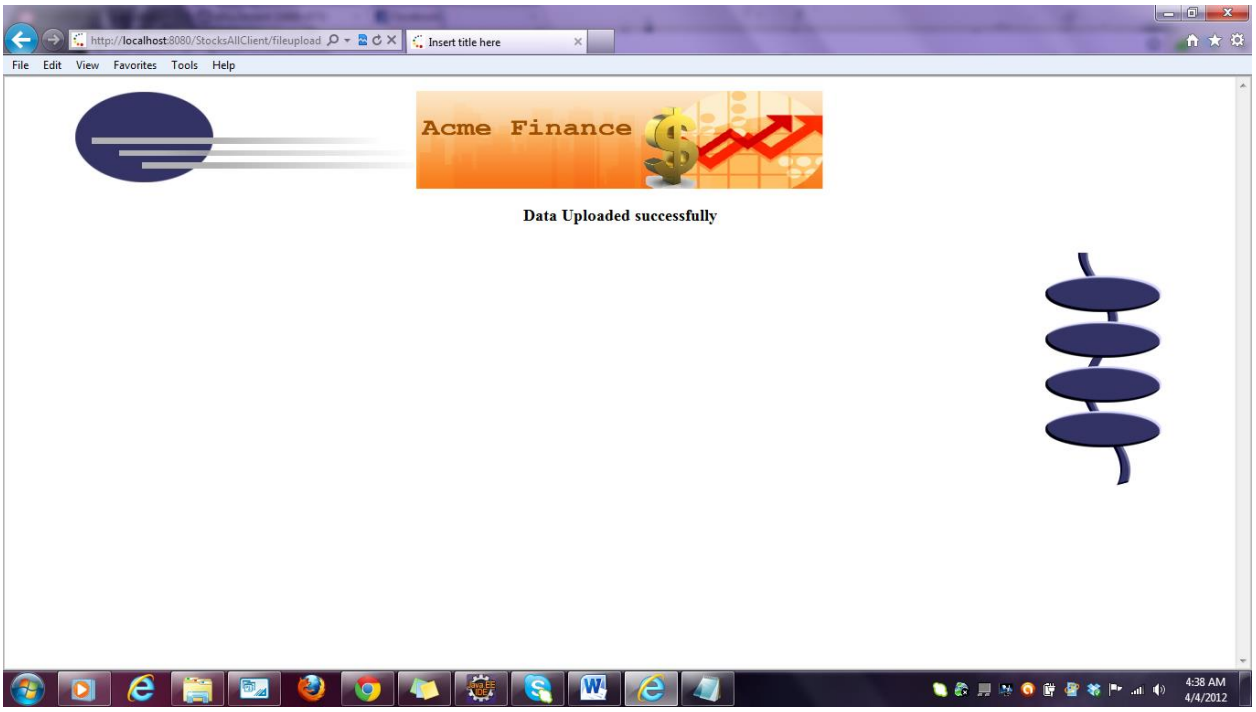
It opens upload.jsp Enter a file name



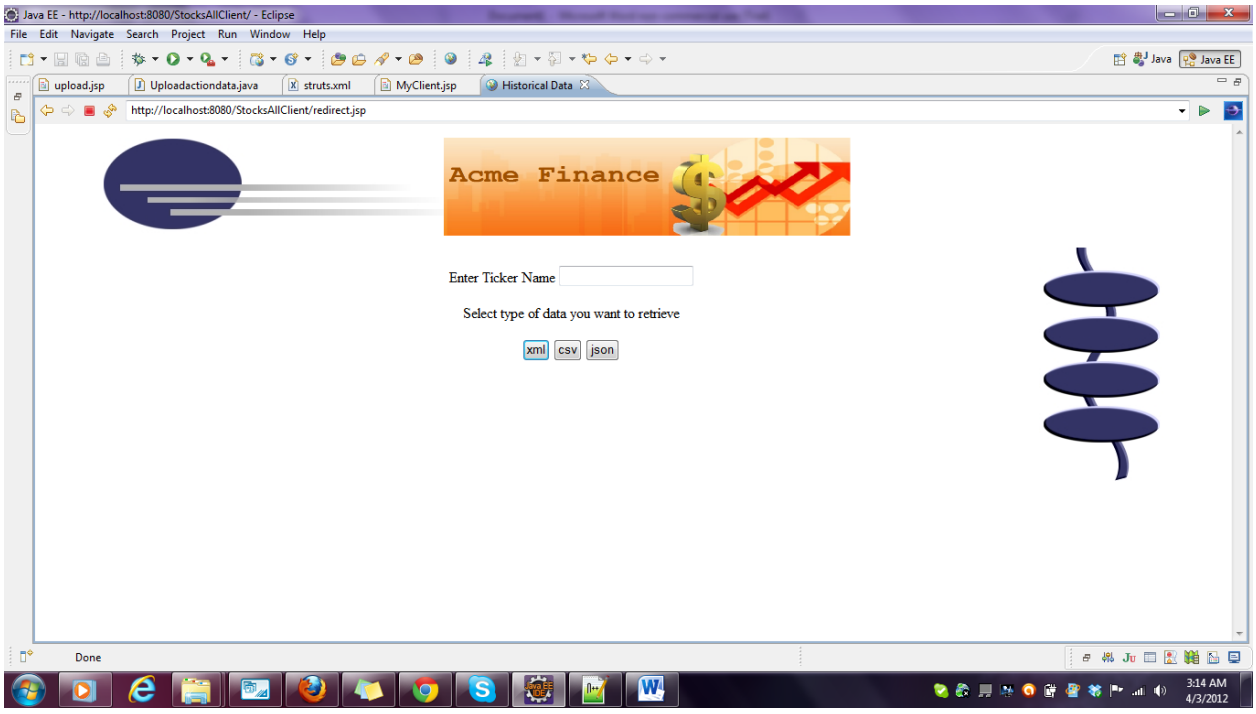
Output on the console indicating the data is being entered in the database:



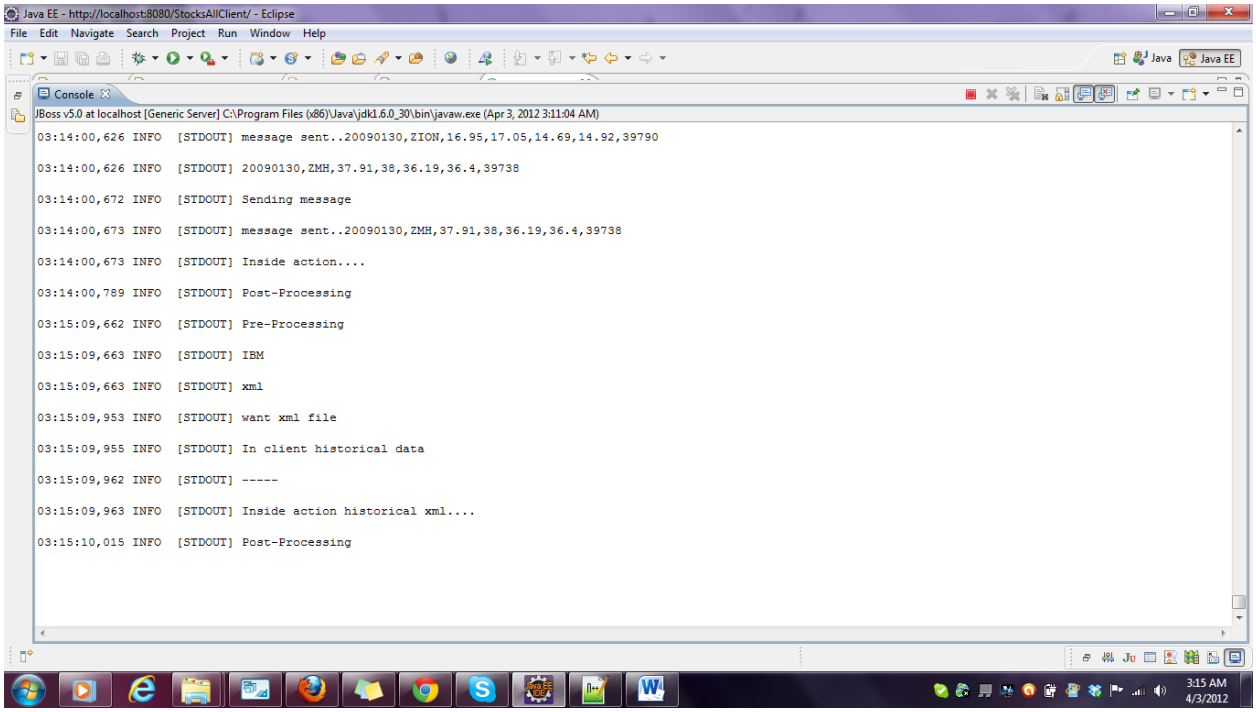
Fileuploaded successfully message



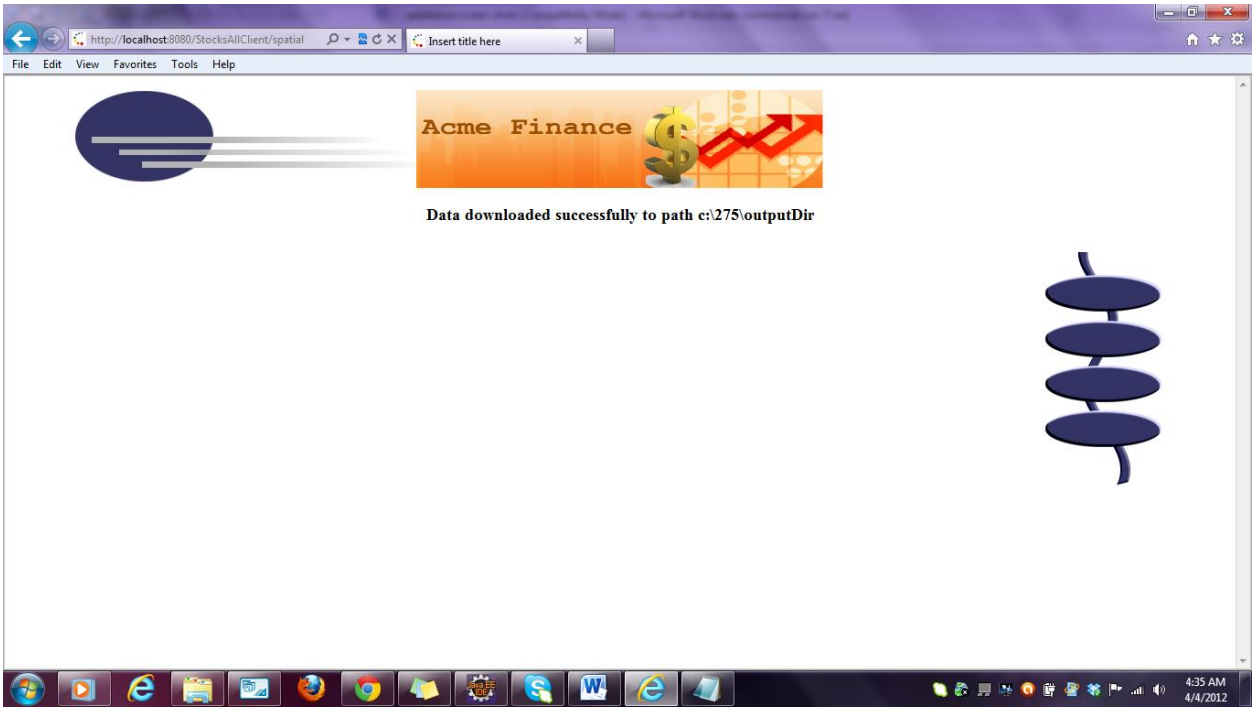
On slecting historical data on the home page



A screenshot to indicate that the file is being written in the given path

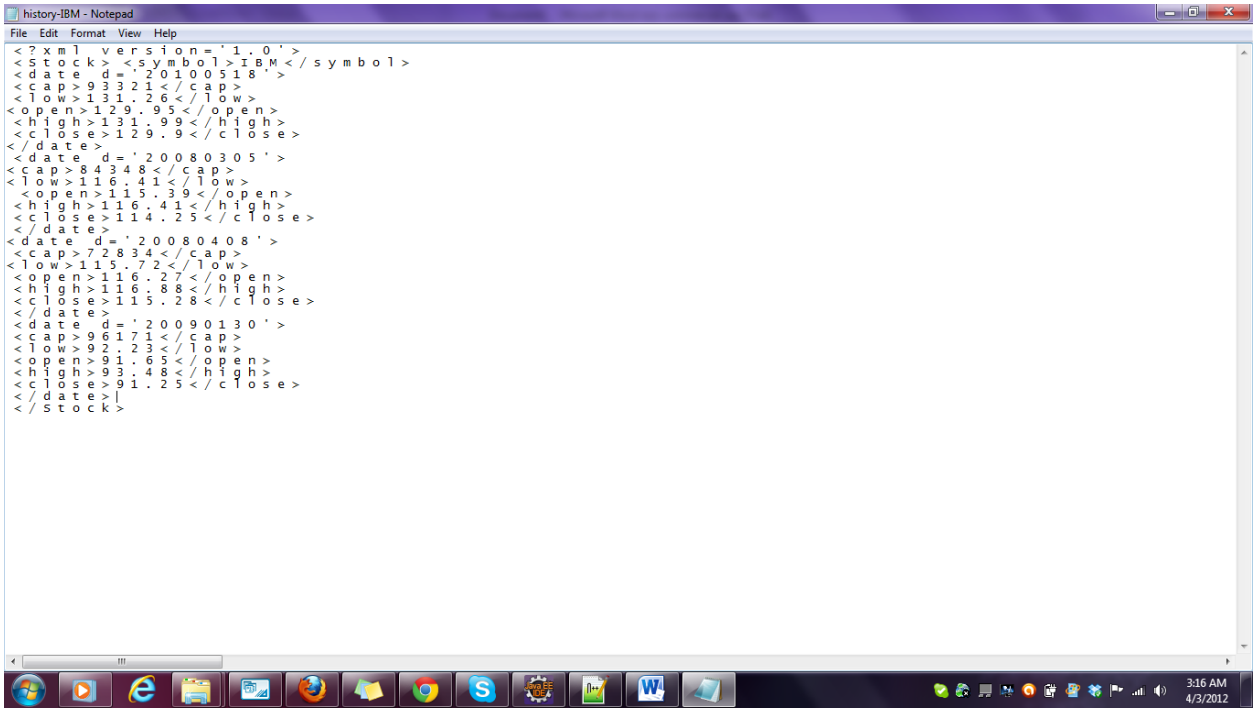


Data downloaded successfully message

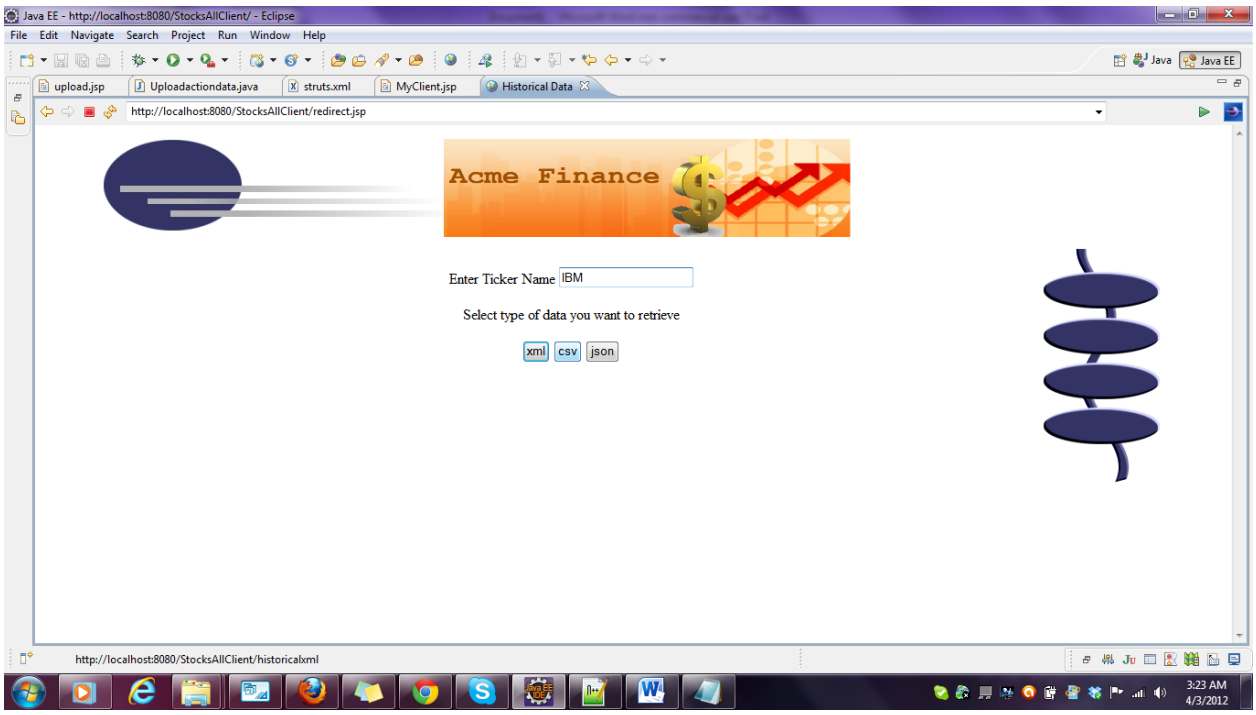


Historicaldata downloaded for IBM ticker value in xml format

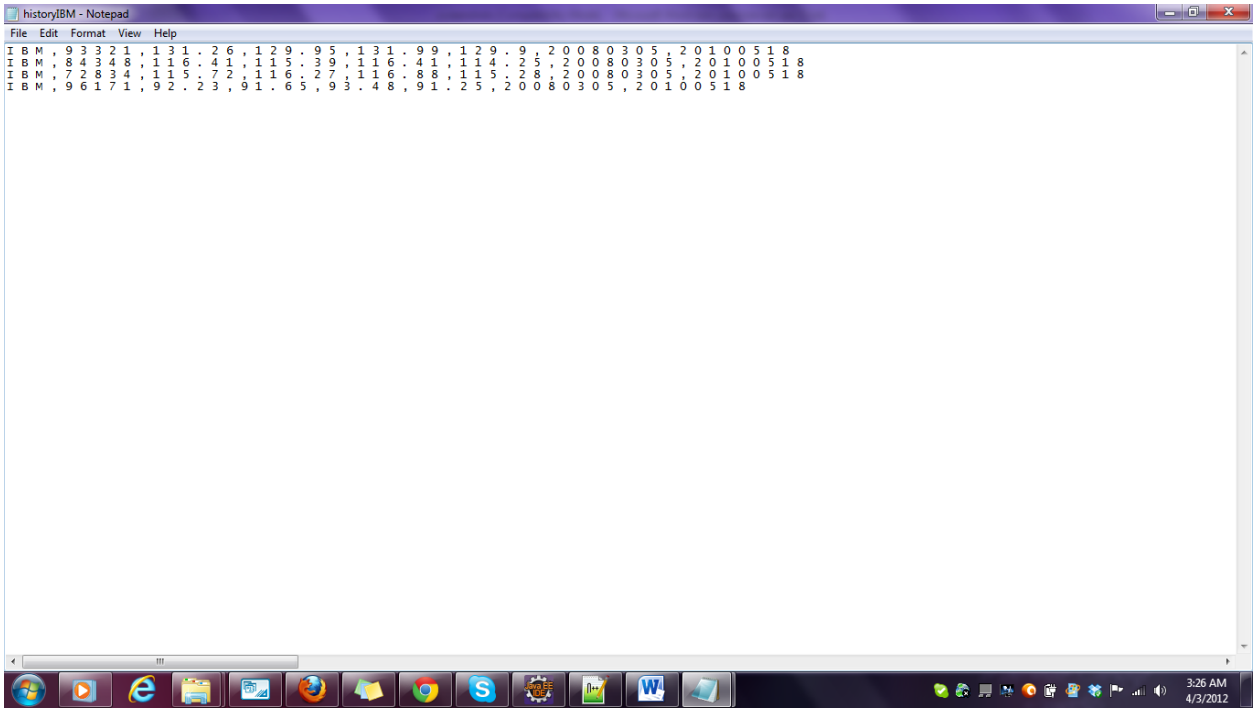
Historicaldata downloaded for IBM ticker value in xml format



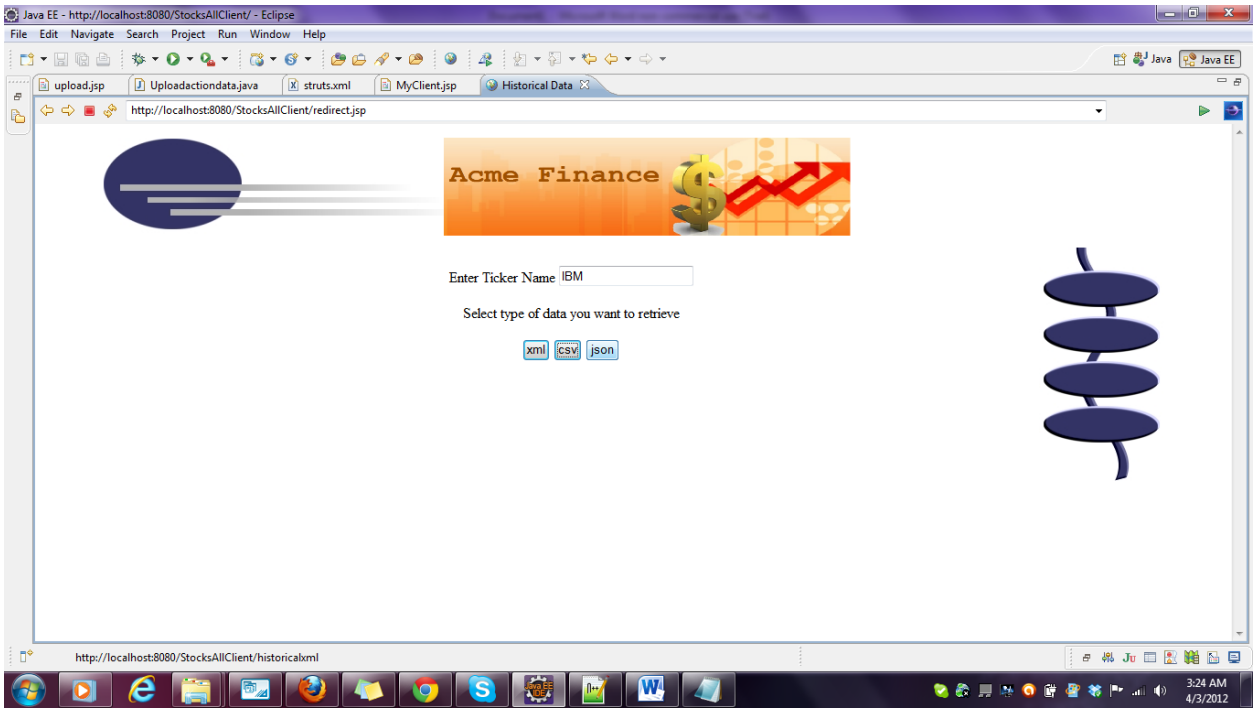
Downloading data in csv format

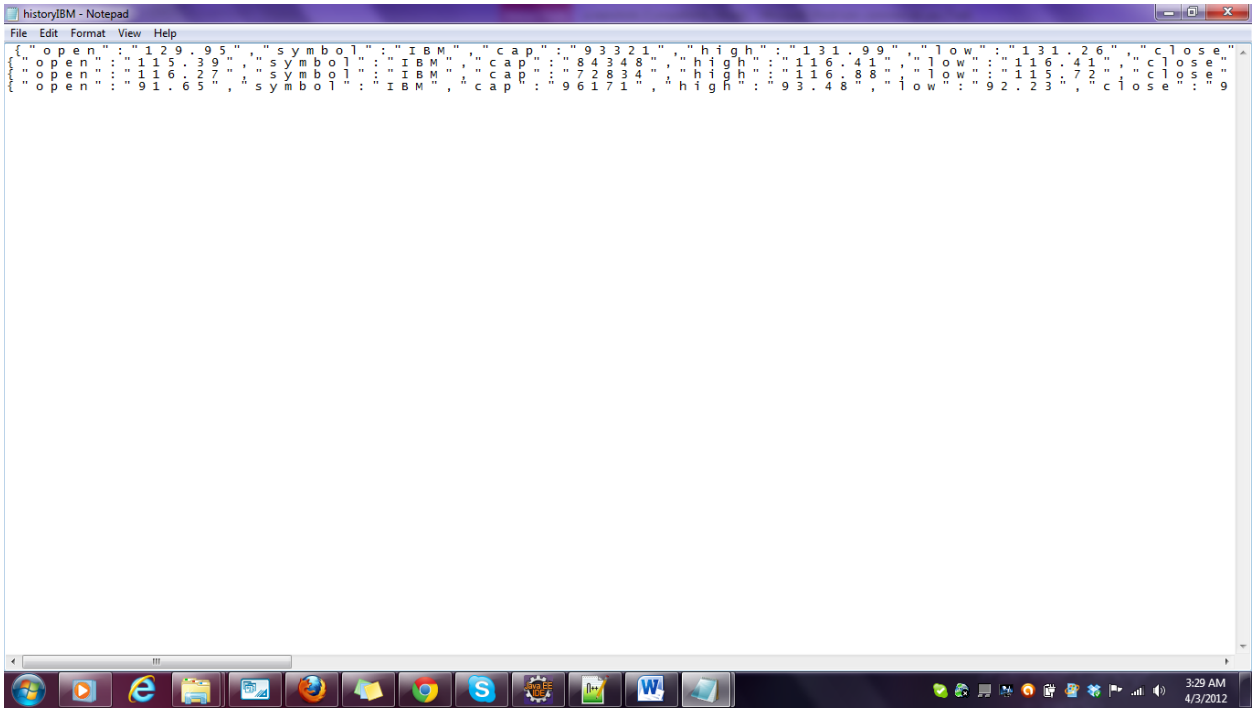


Historicaldata for IBM in csv format

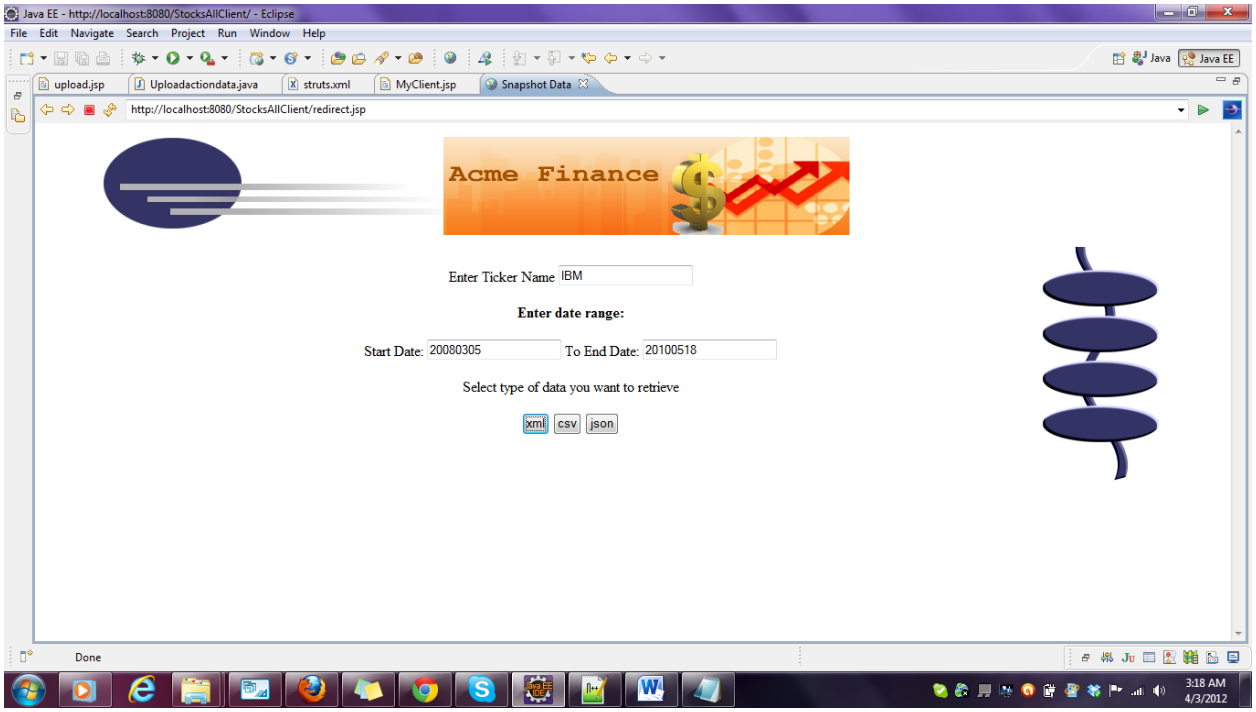


Historicaldata in json format

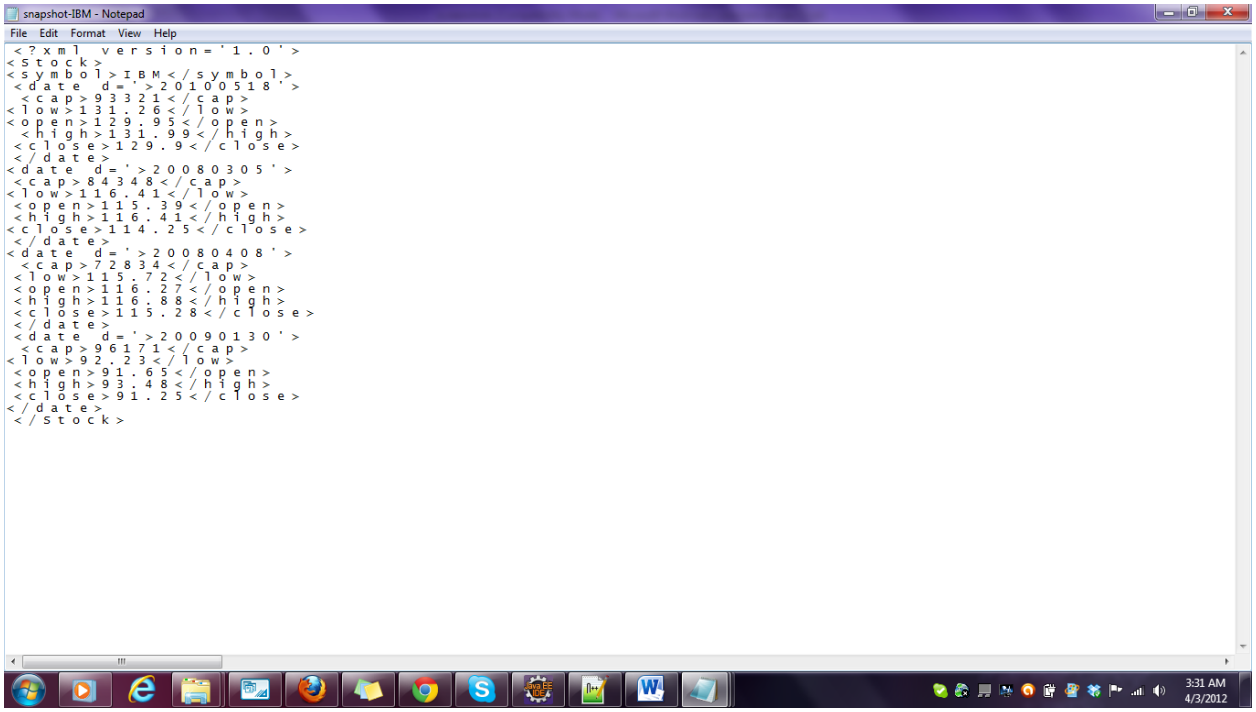




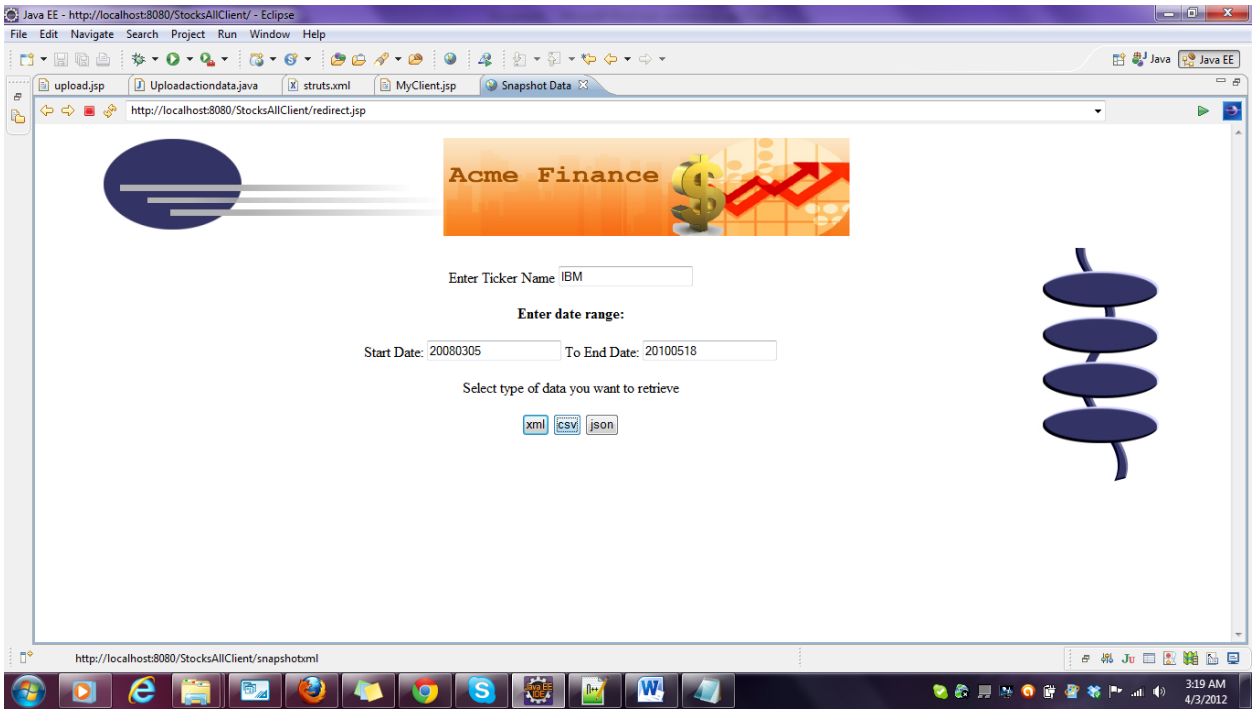
On selecting the snapshot data option



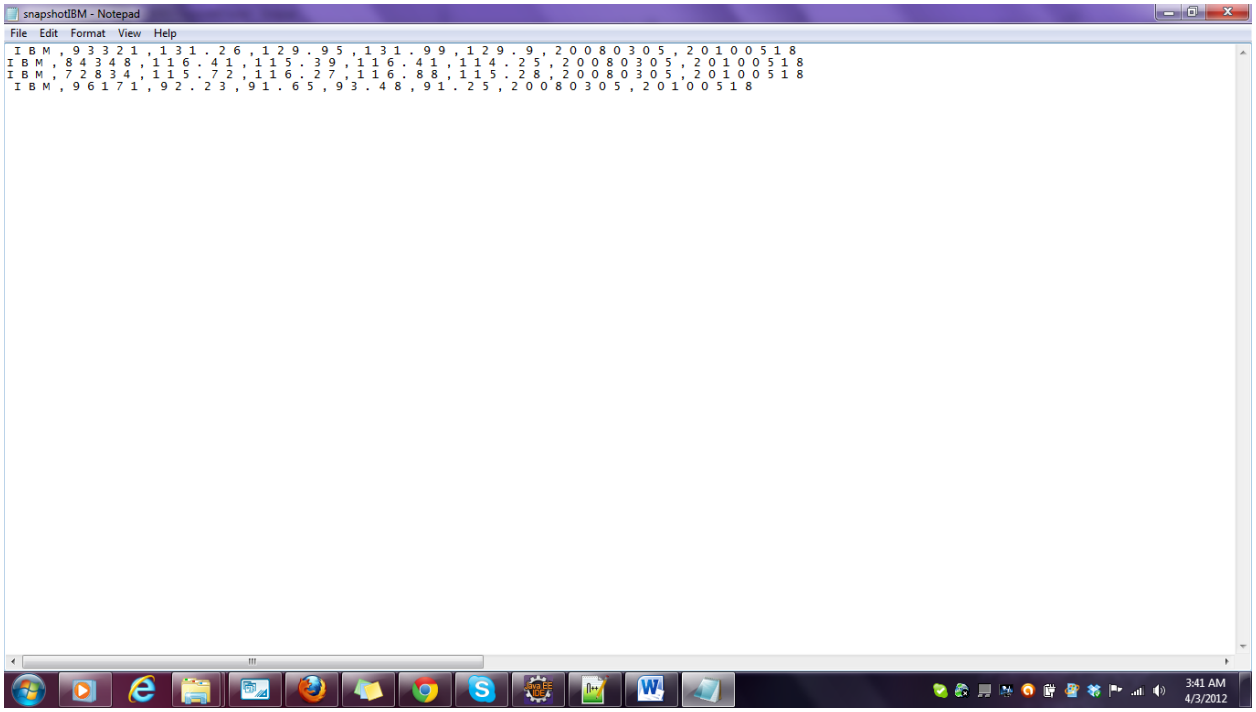
Enter the ticker value and date range and select format



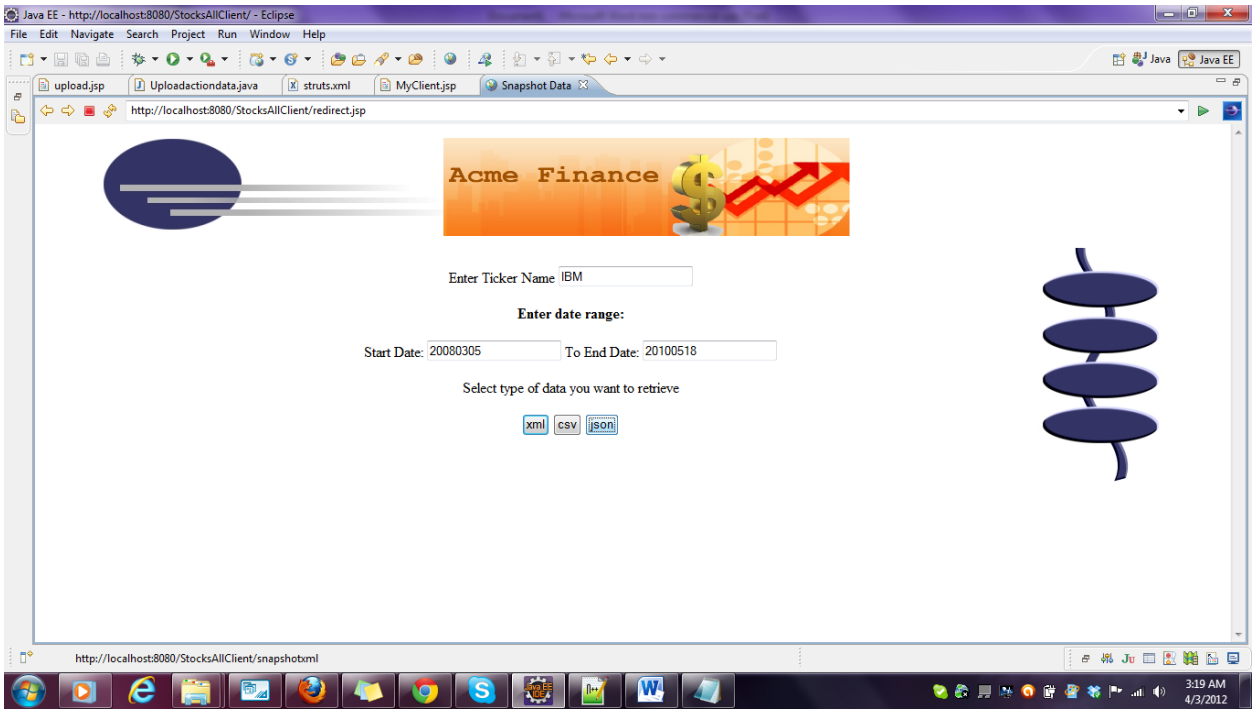
Data downloaded for snapshot data for tickervalue IBM in xml format for date range 2008-03-05 to 2010-05-18



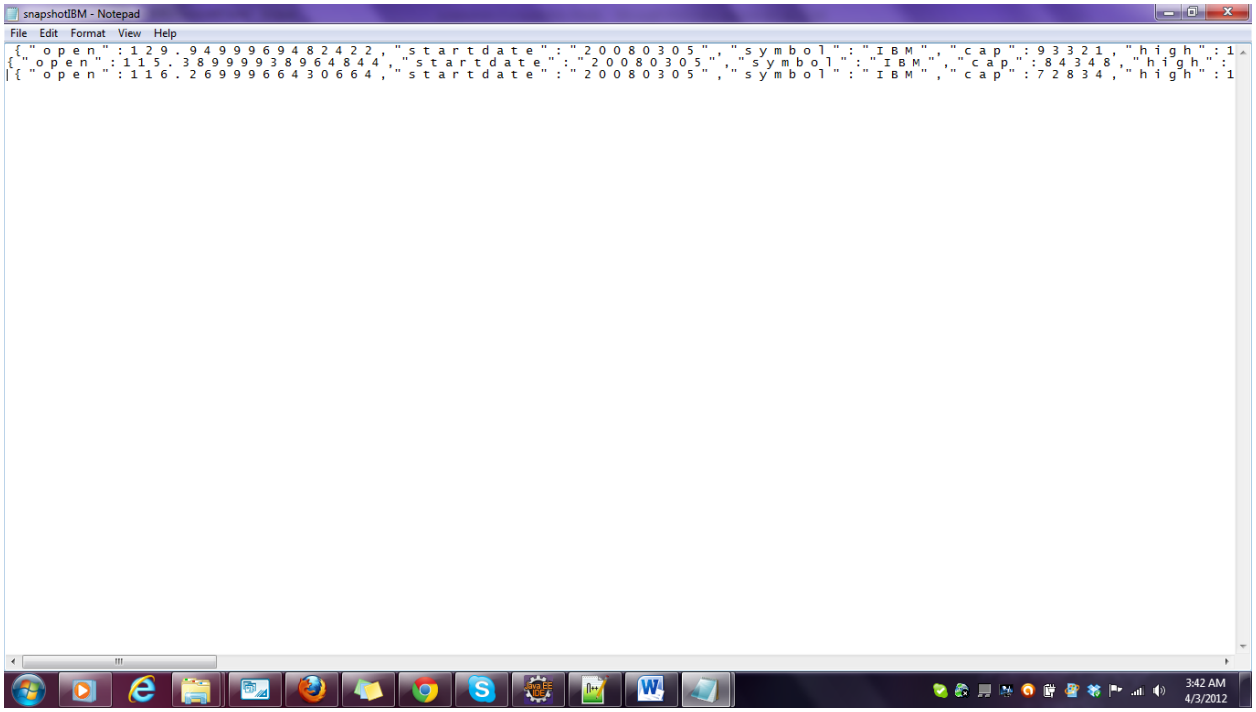
Data download for csv format



Data downloaded for snapshot data for tickervalue IBM in csv format for date range 2008-03-05 to 2010-05-18

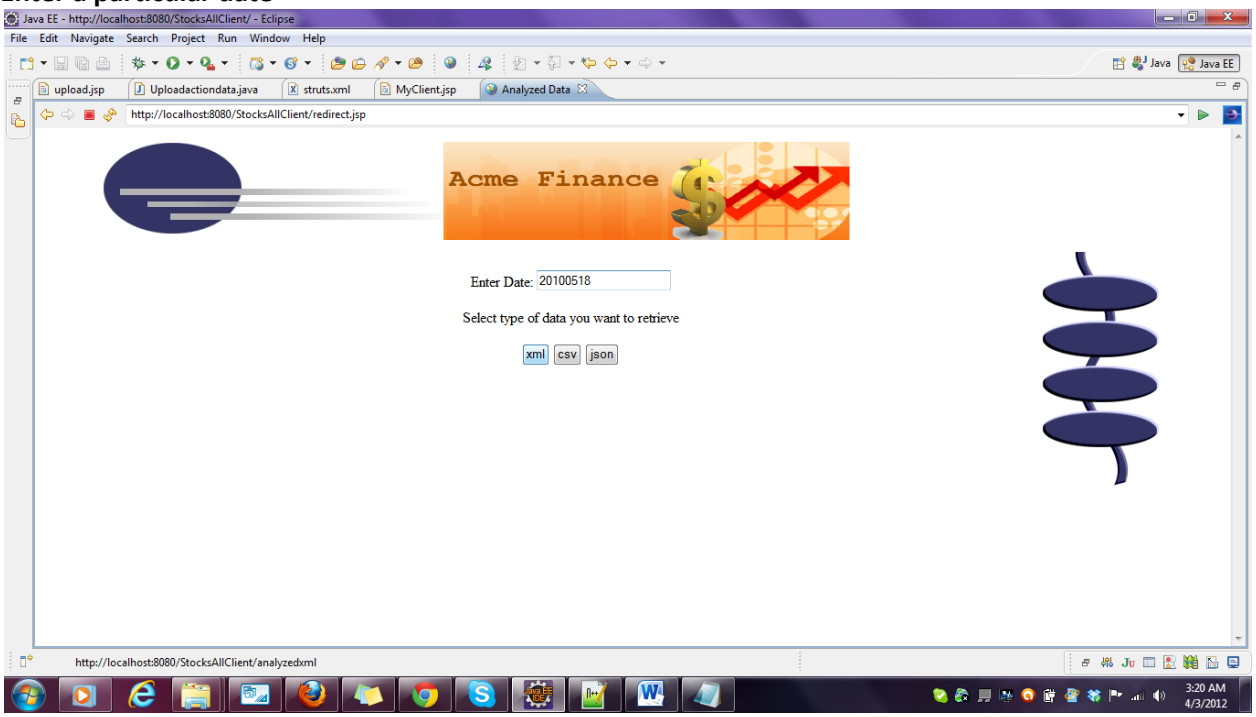


Snapshot download in json format

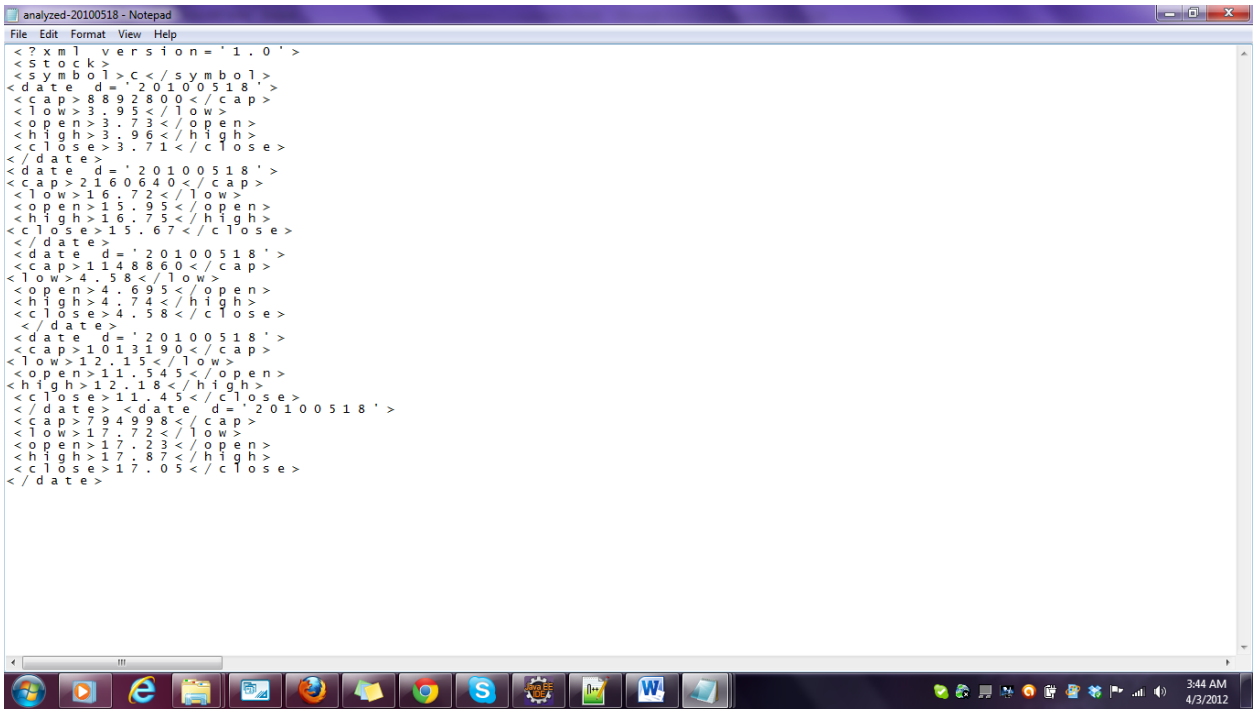


Data downloaded for snapshot data for tickervalue IBM in json format for date range 2008-03-05 to 2010-05-18

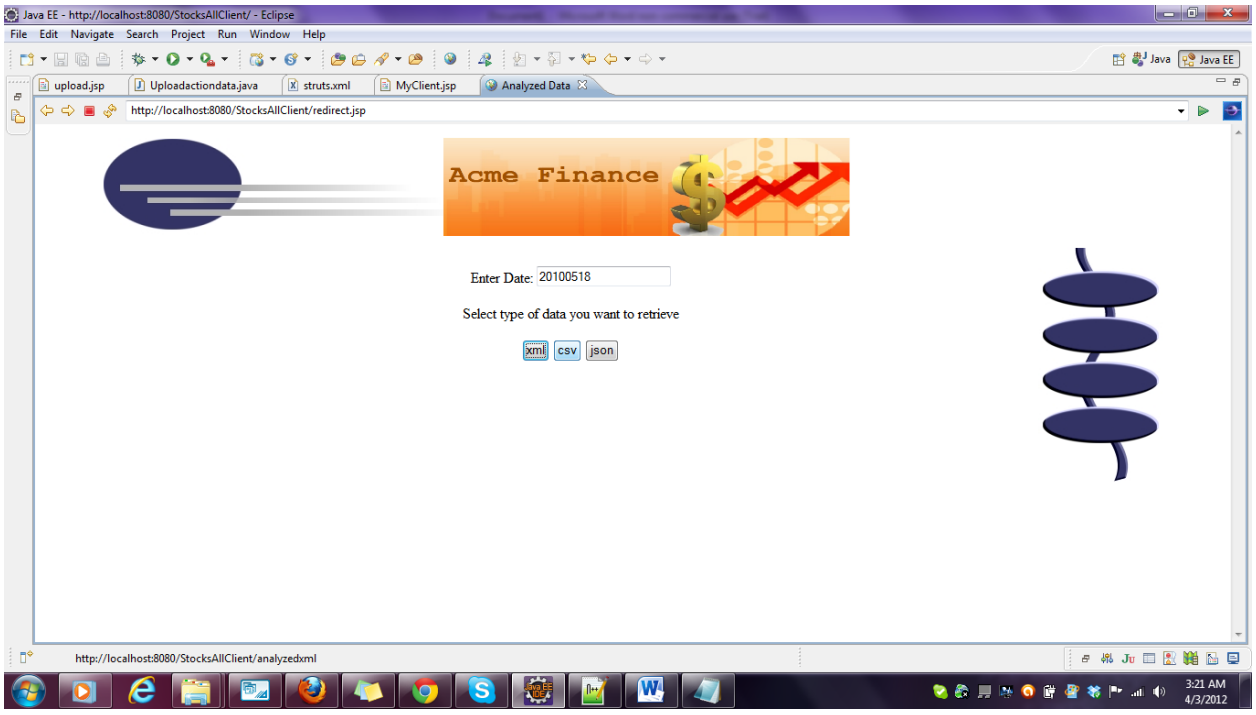
On selecting the analyzed data in the home page
Enter a particular date



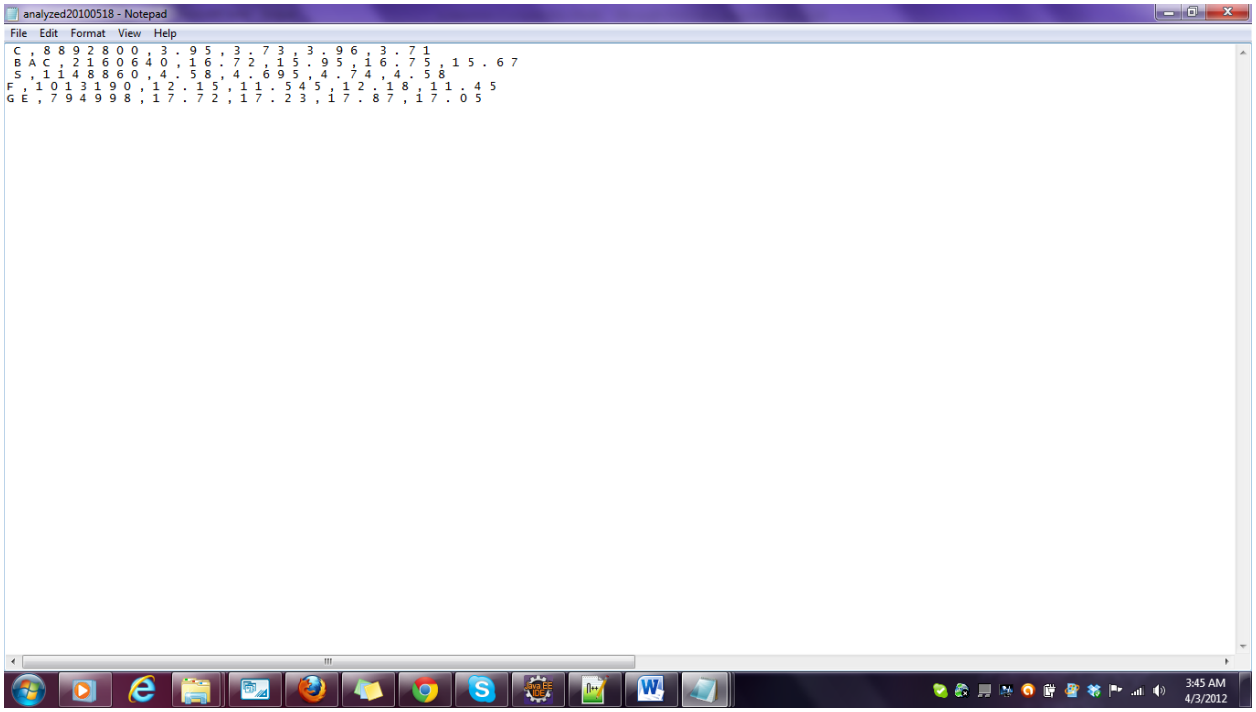
Data downloaded for the date 2010-05-18 in xml format



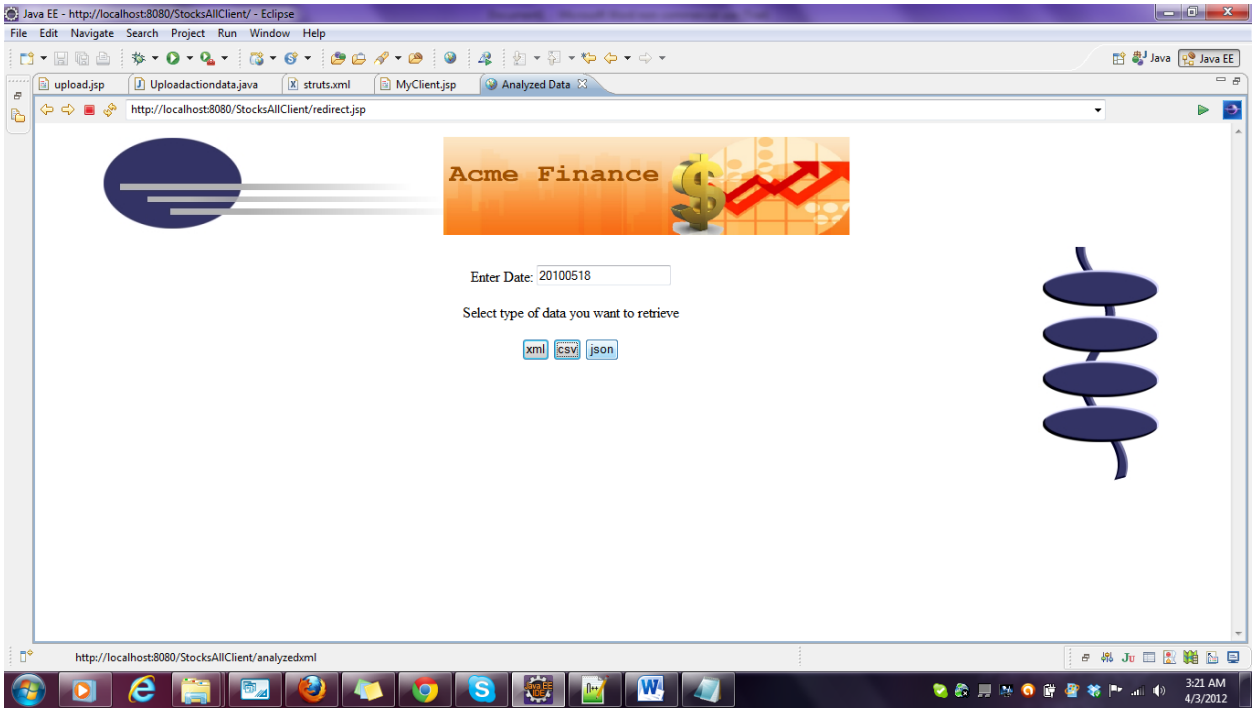
To download data in csv format



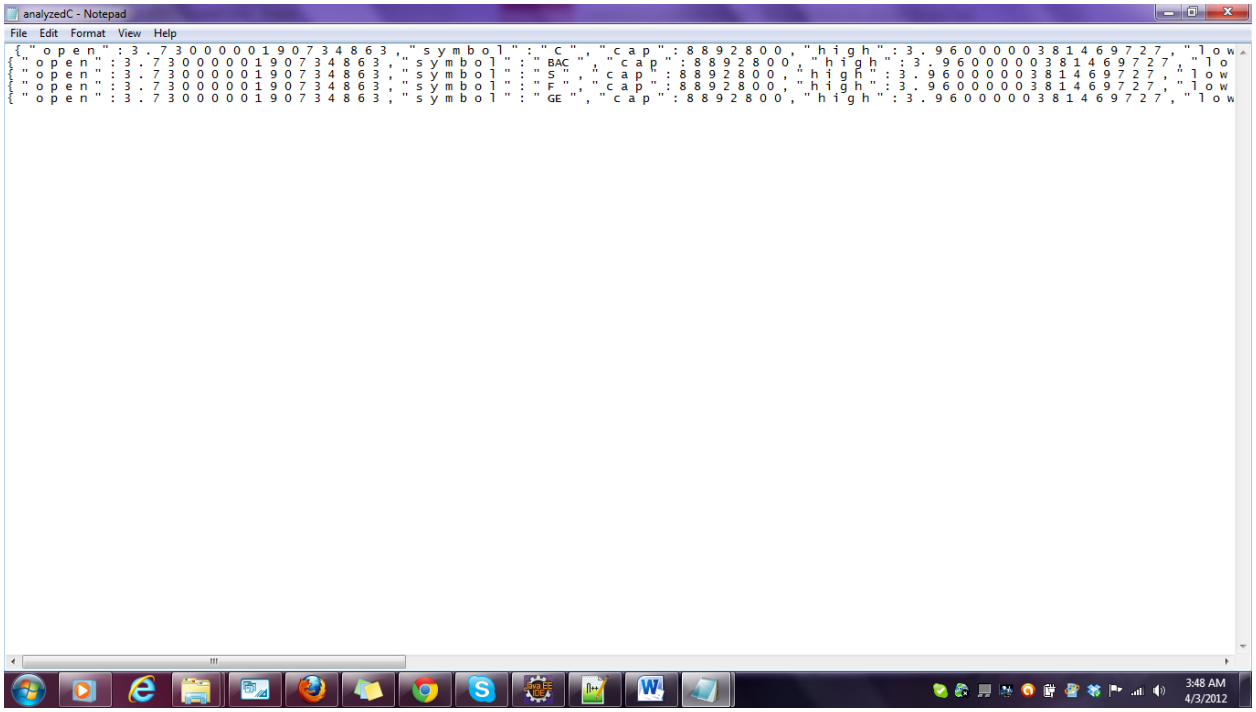
Data downloaded for the date 2010-05-18 in csv format



To download data in json format



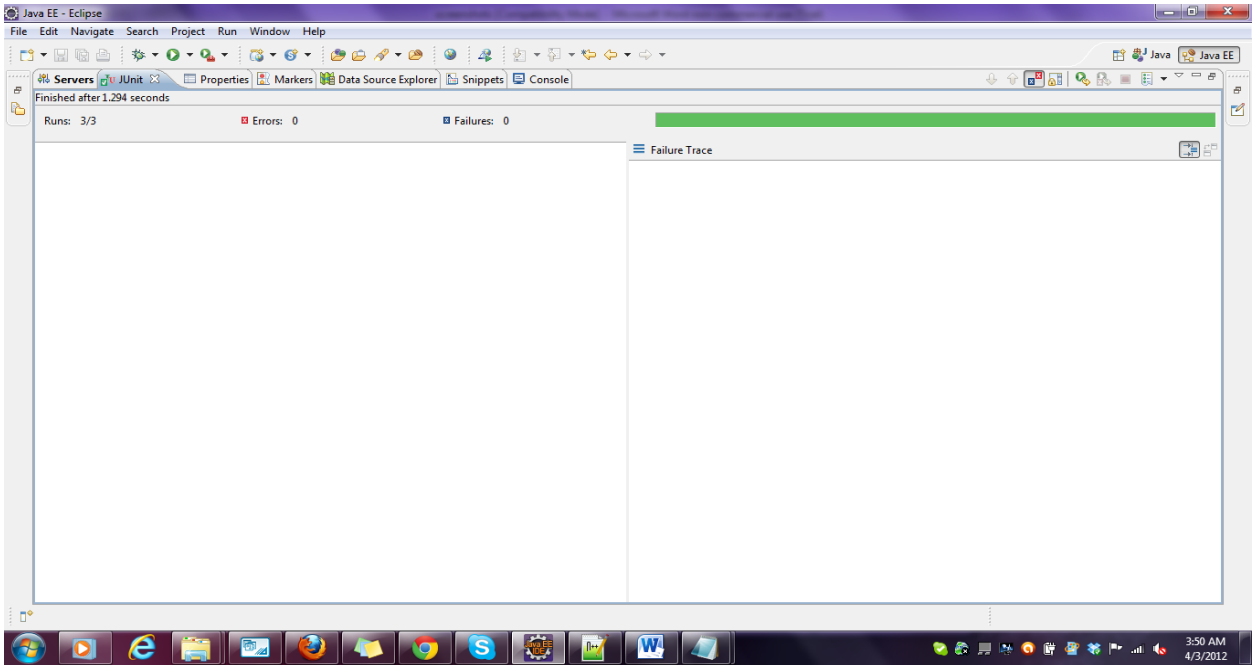
Data downloaded for the date 2010-05-18 in json format



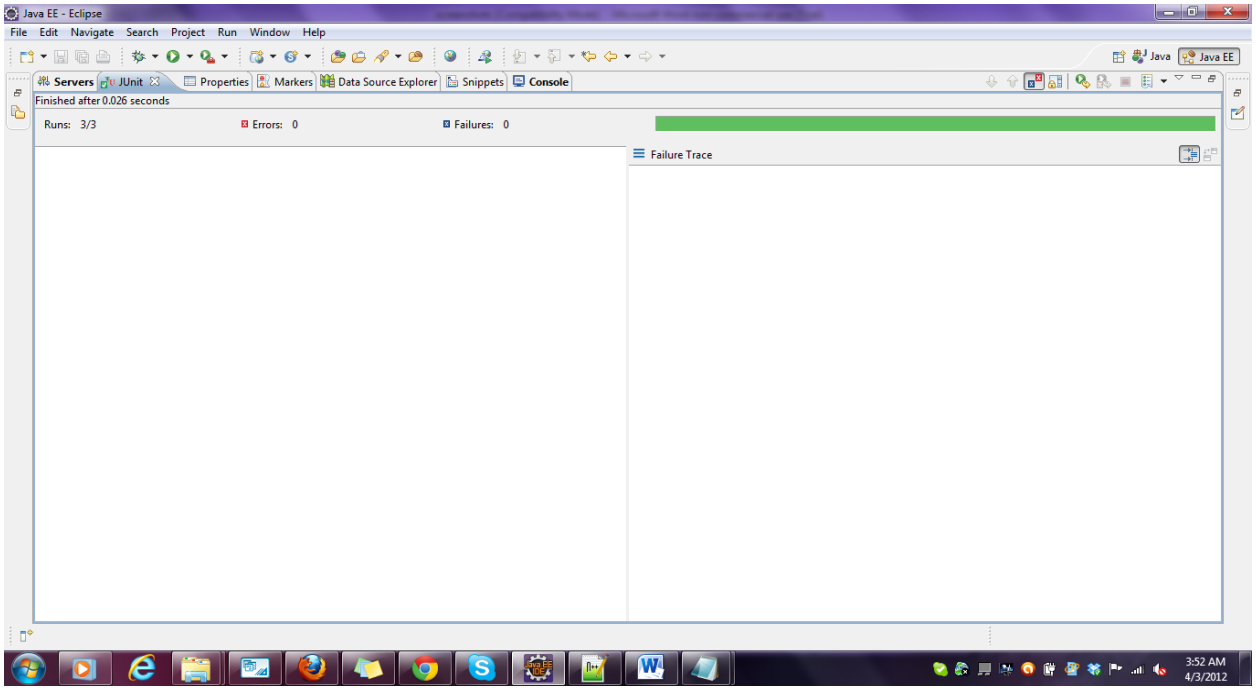
System Testing Results: JUnit & JMeter

Unit Tests: JUnit

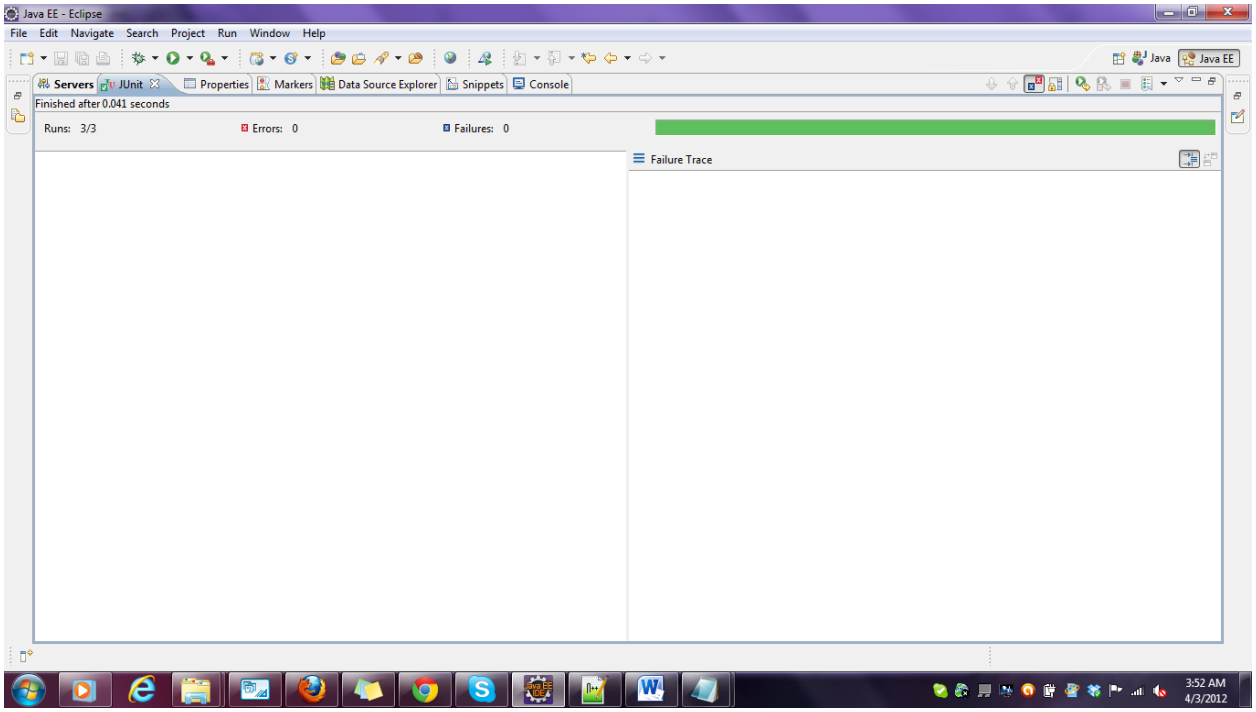
Junit test:StockProviderClient2Test.java



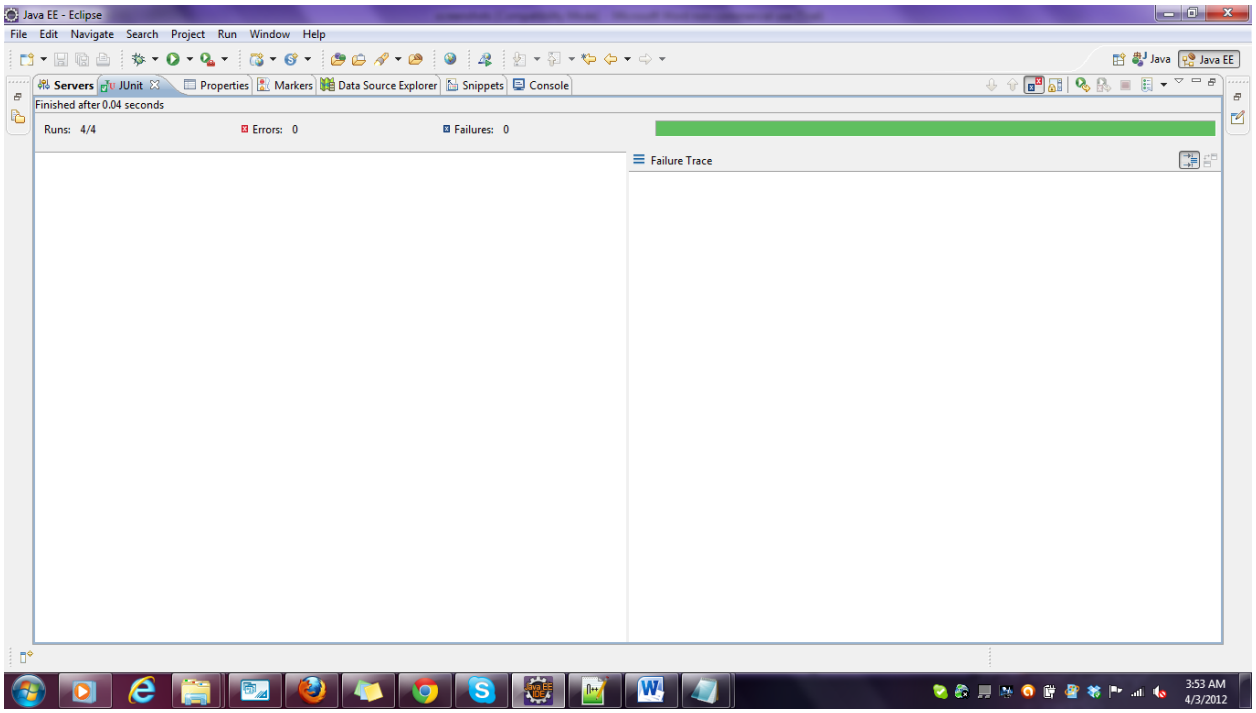
JunitTest:WriteCsv.java



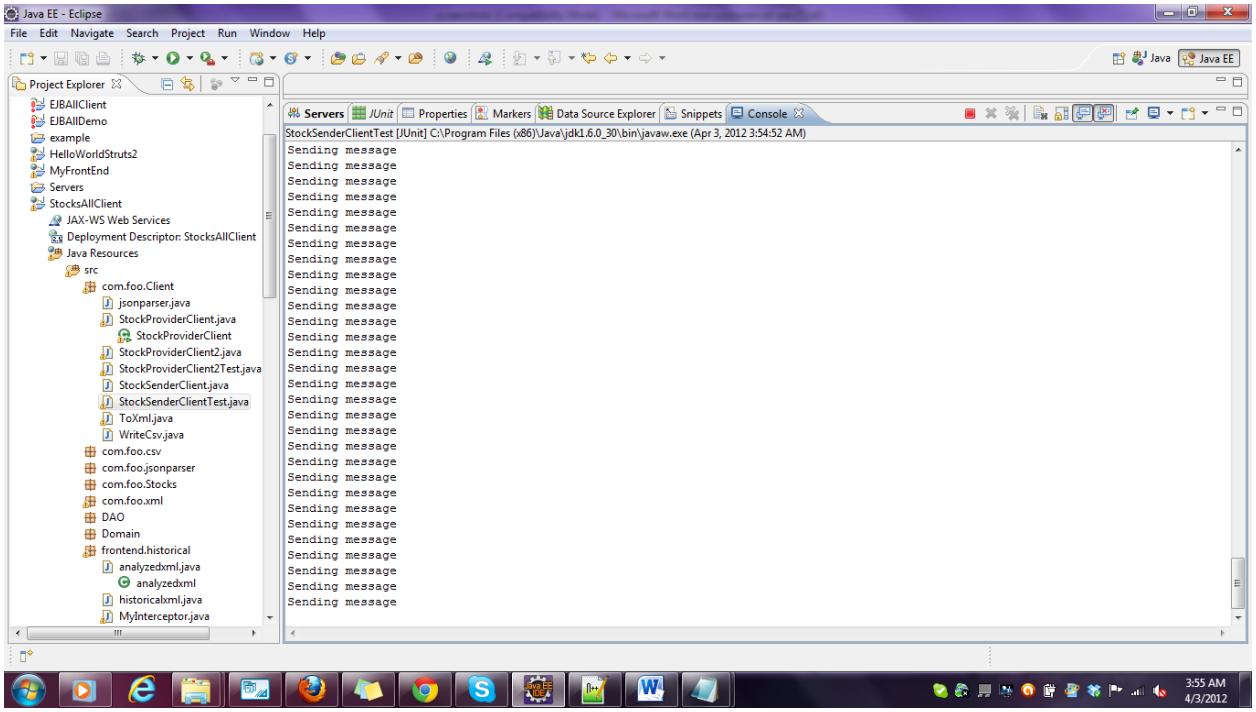
Junittest: jsonparser.java

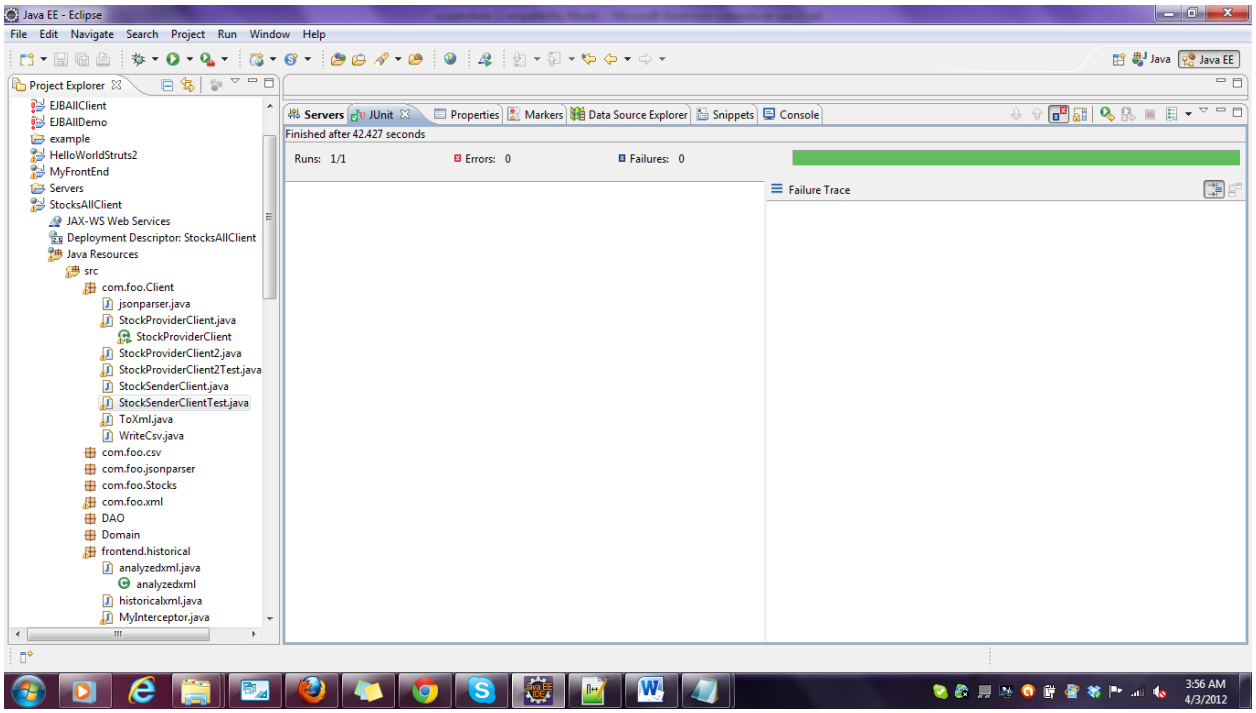


JunitTestLToXml.Java

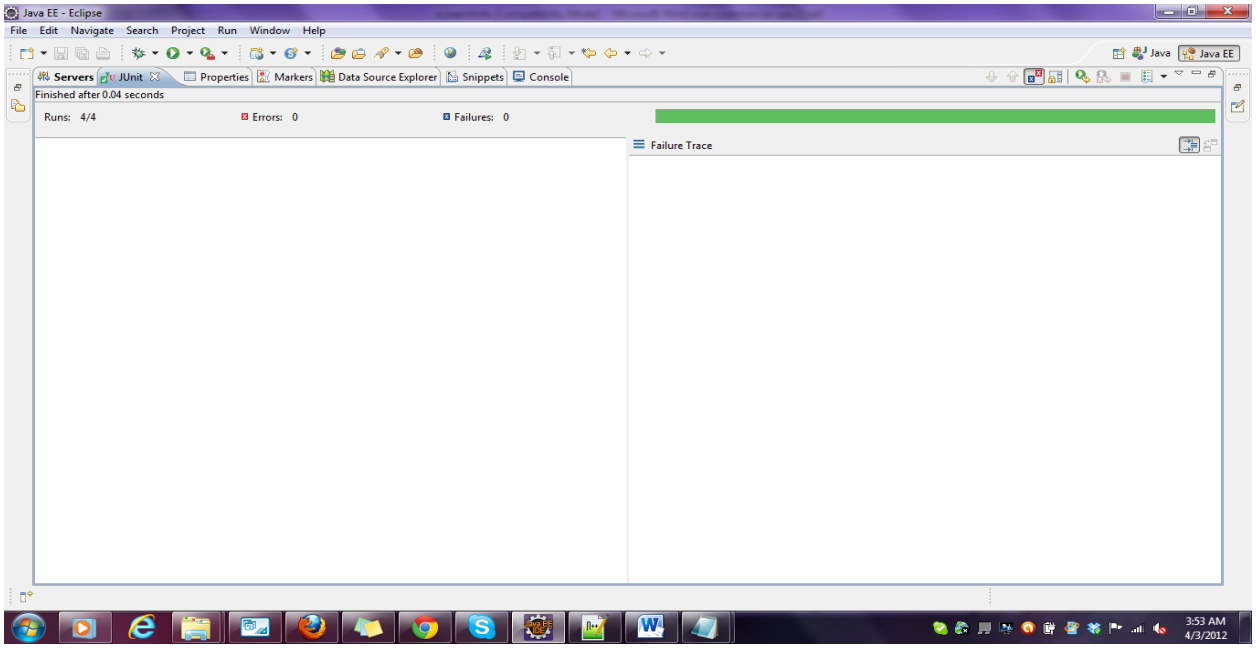


JunitTest:StockSenderClient.java

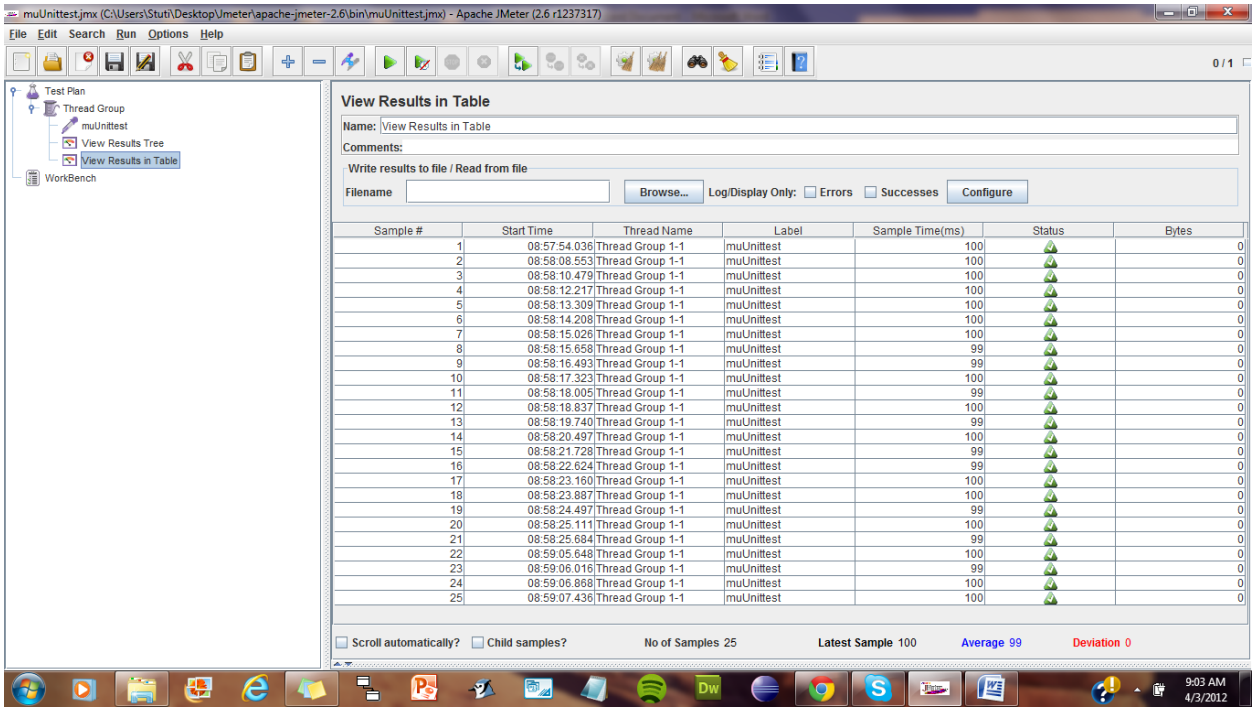


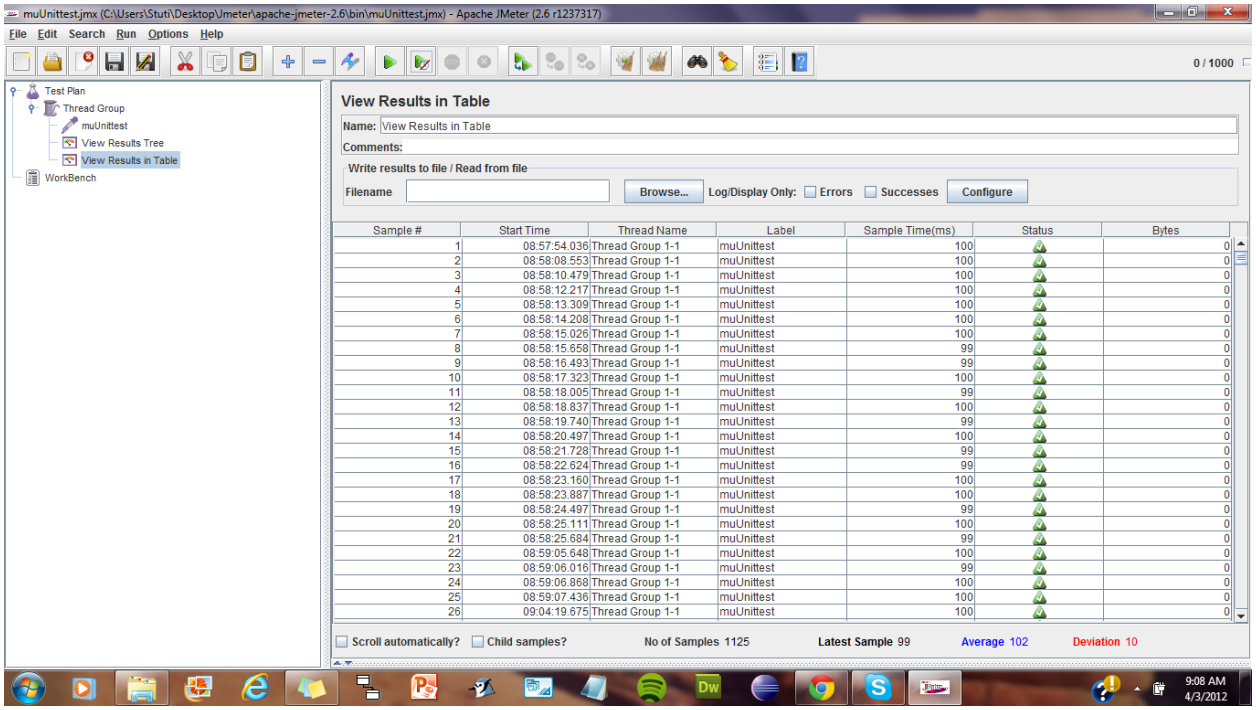


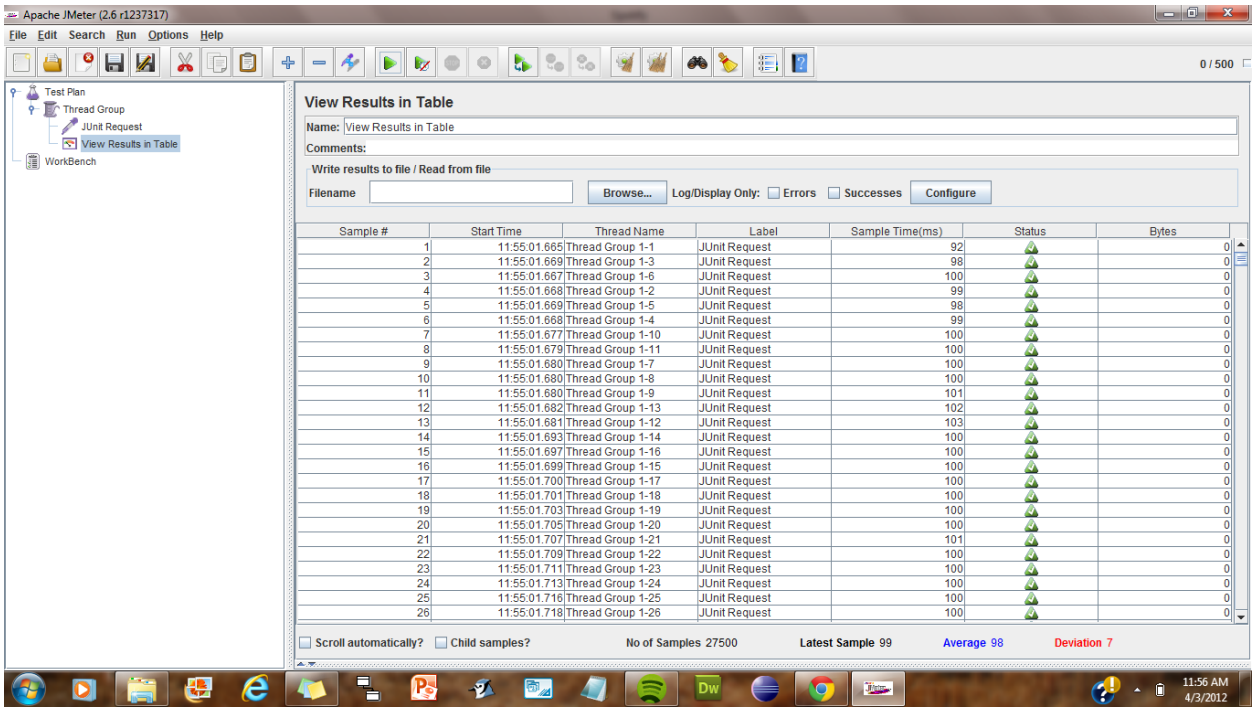
Junittest:StockProvider.java



Performance Tests – Jmeter






Apache JMeter (2.6 r1237317)
File Edit Search Run Options Help
0 / 500
Test Plan
Thread Group
JUnit Request
View Results in Table
WorkBench
View Results in Table
Name: View Results in Table
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Sample # Start Time Thread Name Label Sample Time(ms) Status Bytes
1 11:55:01.665 Thread Group 1-1 JUnit Request 92 0
2 11:55:01.669 Thread Group 1-3 JUnit Request 98 0
3 11:55:01.667 Thread Group 1-6 JUnit Request 100 0
4 11:55:01.668 Thread Group 1-2 JUnit Request 99 0
5 11:55:01.669 Thread Group 1-5 JUnit Request 98 0
6 11:55:01.668 Thread Group 1-4 JUnit Request 99 0
7 11:55:01.677 Thread Group 1-10 JUnit Request 100 0
8 11:55:01.679 Thread Group 1-11 JUnit Request 100 0
9 11:55:01.680 Thread Group 1-7 JUnit Request 100 0
10 11:55:01.680 Thread Group 1-8 JUnit Request 100 0
11 11:55:01.680 Thread Group 1-9 JUnit Request 101 0
12 11:55:01.682 Thread Group 1-13 JUnit Request 102 0
13 11:55:01.681 Thread Group 1-12 JUnit Request 103 0
14 11:55:01.693 Thread Group 1-14 JUnit Request 100 0
15 11:55:01.697 Thread Group 1-16 JUnit Request 100 0
16 11:55:01.699 Thread Group 1-15 JUnit Request 100 0
17 11:55:01.700 Thread Group 1-17 JUnit Request 100 0
18 11:55:01.701 Thread Group 1-18 JUnit Request 100 0
19 11:55:01.703 Thread Group 1-19 JUnit Request 100 0
20 11:55:01.705 Thread Group 1-20 JUnit Request 100 0
21 11:55:01.707 Thread Group 1-21 JUnit Request 101 0
22 11:55:01.709 Thread Group 1-22 JUnit Request 100 0
23 11:55:01.711 Thread Group 1-23 JUnit Request 100 0
24 11:55:01.713 Thread Group 1-24 JUnit Request 100 0
25 11:55:01.716 Thread Group 1-25 JUnit Request 100 0
26 11:55:01.718 Thread Group 1-26 JUnit Request 100 0
Scroll automatically? Child samples? No of Samples 27500 Latest Sample 99 Average 98 Deviation 7
11:56 AM
4/3/2012

Deployment: Ant Build

Ant build: