THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Multi-Tenant and Multi-Vendor Based E-Commerce Platform Using Spring Framework and Restful Web Services

K. Suresh Babu
Assistant Professor, Department of Computer Science
School of Information Technology, JNTU, Hyderabad, India
S. Raghu Nandan
M.Tech (CS), School of Information Technology, JNTU, Hyderabad, India

Abstract:
This paper details the high level architecture and design principles that are being used in the development of the new e-commerce system. One of the goals that we are looking at in the near future is to provide the entire ecommerce as a platform which can host other tenants in addition to a web portal.
The goals of this architecture are to attain

- *High availability,*
- *Scalability,*
- *Extensibility and*
- *Maintainability*

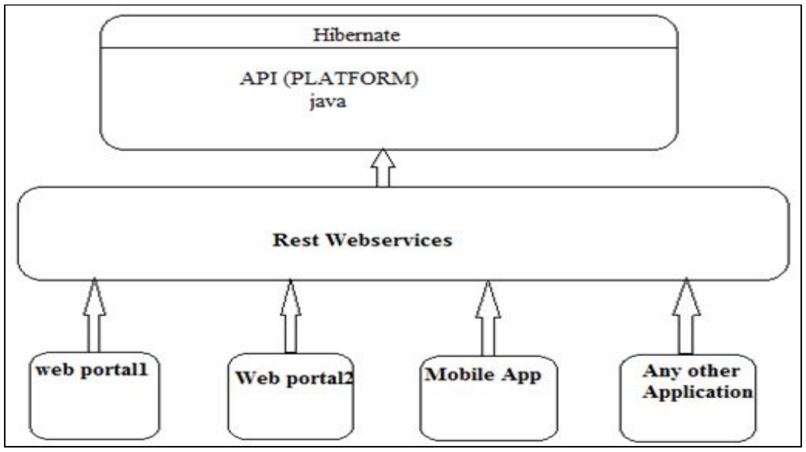
Keeping in mind the above goals a decision was made to go with REST services based light weight J2EE architecture. This e-commerce platform has an advantage of dealing with multiple tenants as well as with multiple vendors. Each tenant can have multiple vendors with a set of business services. These business services were given for each vendor by tenant. Based on these services only a vendor can get the resources through REST API. RESTful web service check for authentication and authorization for every vendor request from any web portal or any other applications (using .net, java, android, IOS etc.).

1. Introduction

E-commerce is fast gaining ground and used business paradigm. More and more business houses are implementing web sites providing functionality for performing commercial transactions over the web. It is reasonable to say that the process of shopping on the web is becoming commonplace. This application is an ecommerce platform for fashion based products. This provides services to its customers to buy and sell products like apparel, accessories etc. This deals with developing an e-commerce website for online purchase. It provides the user with a catalog of different products available for purchase in the store. In order to facilitate online purchase a shopping cart is provided to the user.
The various users/actors in the system are as follows:

- Buyer – one of the two primary end users of the system. Buyer can purchase products from the portal. The identity of the buyer can be his/her Facebook account or a web portal account created using an email address.
- Seller – the other primary end user of the system. Seller can sell products using the portal. Seller needs to register with portal using an email address.
- Admin – Has control over the entire system except creating users.
- Super Admin – Admin + user creation ability.

2. System Architecture



3. Technology Overview

It will follow 3-tier architecture with the following layers

- **Persistence layer**: The persistence layer shall consist of an RDBMS (MySQL) and Mongo DB for storing images. Mongo DB shall be used for storing images which can be accessed by multiple instances of the web application and the API server.
- **Application layer**: The application layer shall consist of J2EE applications and services that serve http requests. We use JAX-RS, Spring MVC, Dependency Injection and Spring Security in this layer. This layer consists of two applications – Web application and API. Web application consumes the API and serves UI requests. The interface between Application layer and RDBMS shall be through an ORM framework. The framework of choice here is Hibernate. The API application will expose the backend functionality in the form of a REST API with JSON or xml output. The API application uses JAX-RS and the implementation used is Apache CXF. The communication format between Web application and API shall be JSON over http. We shall use JAX-RS client to ease development here. The application layer can be deployed to one or more tomcat application containers. All the requests from UI are sent to the Web Application, which services them by consuming the API. All data access is via API calls.
- **UI layer:** The UI layer consists of html pages generated by a template framework like JSP. In addition to this, we shall use JavaScript and other JavaScript libraries like jQuery, jQueryui to make the UI dynamic. We also take advantage of newer UI paradigms like MVVM by using knockout**.**js to bind the UI to a single view model so that any changes in the view model are reflected in all the UI elements.
- **Security** for API is provided by checking Authentication and Authorization. Each and every request is processed through a filter before sending it to actual service. In this filter we check for Authentication and Authorization.
- **Authentication** is used to check whether the requested user exists or not and **Authorization** is used to check whether the requested user has permissions for that service or not.
  Basically we check Authorization header for every request, parse the value to get Vendor Id and API Key. By API key we get the Seller from the database and compare with the given Seller Id. If there is no value or incorrect value in the header, then 403 Forbidden is sent in the response. A Seller should be added to database and get ApiKey to access this service. There are permissions which are grouped together and mapped to each service.
  Every seller may have different types of business services with particular business channel and business services which are mapped with permission groups which help to get all permission of particular API service.
- **E.g.** A Vendor can have Business service with Business channel 'A' that has 90% as Vendor's share and 10% as Tenant share. Similarly same Vendor can have another Business service with Business channel 'B' that has share of 80% and 20% for Vendor and Tenant respectively.

This is used at the time purchasing a product. When a product is sold for 1000 Rs, money is transferred to Vendor and Tenant account according to the model in the database.

4. Software Components

The various different software components of the system are as follows

- **Data Access Component:** This will be a jar file that contains all data-access code. This component will contain all the DAOs and Query objects that interact with the DB using Hibernate. This is not an independent deployable and will be embedded in web application and API deployment files.
- **API Application:** This will be a war file that can be deployed on to an application container. This layer is responsible for exposing the backend functionality in the form of easy to consume REST API with json or xml as the response format. This component is authenticated and authorized.

- **Web Application:** This will be a war file that can be deployed. All the requests coming from the Web UI are handled by this component. This component is also responsible for handling security (authentication and authorization). The application will directly contact API for any read requests and all write/update.
- **Web UI:** This is not an independent component. This is part of the web application. This consists of jsp, JavaScript, html and css. We make extensive use of templating in the form of jsp and knockout templates and data-binding.
- **MySQL:** MySQL is the RDBMS of choice which is in contact to API. Any call either read or write to this database should be done through application API.
- **Mongo DB:** We are going to use Mongo DB as a replacement for a networked file system to store images. Usually images are stored on a file system. In our case we would be having multiple instances of the API and Web applications running in a clustered configuration, so storing the images on any one system is not going to work. We would need a networked file system that can be accessed by all instances of Web and API servers. Mongo DB provides that capability via **GridFS** component which can store files and can be accessed like a DB. While MongoDB stores the initial image uploads all images are served from **ReSrc.it** (a third party image CDN). ReSrc.it reads the image for the first time from API server and all future requests are handled by ReSrc.it.

5. Conclusion

This entire E-commerce platform uses Spring and RESTful web services, by which we can attain high availability, extensibility and maintainability. From any application we can send request to the API and get the resources in a required format like JSON, xml etc. Those applications can handle the response and display using different UI mechanisms.

6. References

1. http://predic8.com/rest-webservices.htm
2. http://www.infoq.com/articles/rest-introduction
3. http://www.vogella.com/tutorials/REST/article.html
4. http://en.wikipedia.org/wiki/Spring_Framework
5. https://developers.google.com/appengine/articles/spring_optimization
6. http://java.dzone.com/articles/spring-framework-architecture