Developing Rich Internet Application from Thick Client Architecture

Varsha Kuldharme, Prateema Karande, Vinayak Kalaskar, Pranav Kamble, Prof.Bhavana Tiple

Abstract: **Server Monitoring manager is a software which remotely monitor and control servers. This complete thick platform dependent application is to be replaced by a thin highly interactive application. server monitoring is done through remote procedure calls. All application programming interfaces are implemented using J2EE. A Single Page Application's (SPA) using HTML5 is the front end which provide interface for services**.

Keywords— **Rich Internet Application, Web services, Thick client, Model View Architecture, SOAP, Backbone, JQuery.**

I. INTRODUCTION

Web is complete dynamic and continuously evolving structure with highly interactive and responsive rich application that serves the user from the tiniest technologies to the large enterprise based business logics that performs complex local task stores and data locally and only communication with remote servers to fetch and update linked data.

Server Monitoring manager is the platform dependent application which is a multi-media system that supports the contact system to navigate and manage transactions across multiple media channels providing various task like video call, chat, regular call, web based e-commerce.

This application aims – "To solve the dependencies of the present application in a very interactive and responsive approach towards building the loosely coupled architecture using the rich internet application which enables the remote and reliable access." Rich Client user interface can provide high performance internet rich user experience for an application that must operate in stand alone, connected, occasionally connected and disconnected scenarios. While this technology used to create stand alone application, they can also be used to create an application that runs on the client machine but communicate with services provided by other layers that exposes the operation the client requires like data access, information retrieval ,searching sending etc.

MVC-Model View architecture is a design patterns used for development of web application that implements the framework which achieves a clean separation between three components of the pattern i.e . Model-View-Controllers. "Separation of concerns" means that one layer doesn't care how another layer is implemented.

a. Each layer relies solely on the behavior specified in the API of other layers.

b. The API specifies the behavior of the interface between a layer and the layers that interact with it.

c. The APIs may be considered contracts to which each development team agrees before going off to implement the behavior promised by the interface.

A public subscribe design pattern will ensure lower coupling and high cohesion in the system Data binding is used to display data whenever possible especially for multi rowed data.

II. NEED OF JAVASCRIPT IN MVC FRAMEWORK

Many things that go into structured application than linking DOM manipulation libraries, templates etc. Since application's main motive is to fetch the server status and the related data for view rendering and data manipulation will be occurring in the browser JavaScript Model View framework. It will simply download single payload containing all styles and markups which user needs for common task and to perform the additional behavior in the background.

BENEFITS:

a. A very modularized architectural application
b. A very easy server side consistency
c. Very low coupling with DOM i.e. separating DOM from data.

III. ANALYSIS OF TECHNOLOGIES APPLIED IN DEVELOPMENT OF SYSTEM

The implementation of this application shows and realizes the idea of MVC design pattern which is based on separation of concerns that allows

a. Reuse of business logic across the application
b. Development of multiple UI's without touching the business logic.

Discourages the streamlining, upgrade and the maintenance task. This MVC is gained by using backbone.js. It is a lightweight JavaScript library that adds structure to your client side code. It makes it very handy to decouple the concern in application providing maintainable and cohesive code.[1]

IV. MVC USING BACKBONE.JS

Backbone.js is a lightweight JavaScript library depends on jquery.js and underscore.js. It provides a proper skeleton to the application to make is scalable and maintainable. We will use backbone.js to create a SPA. In backbone.js MVC is Model View Collection, and the controller logic is implemented by the routers . Since it provides the separation of concern, Model is unknown about the view, whereas the view is only responsible for displaying the data in collection.

We would directly go for the **example** using the backbone.js.

Consider a server Monitoring application where the user has to keep a check on the server status and its updates. Each server would be having its domain, where these servers can be listed according to the domains. Here for making the GUI very responsive and to enable the application support on device viewports, Twitter Bootstrap with its Java Scripts and CSS is used which helps to achieve the required GUI. One needs to add bootstrap.js and its related bootstrap.css file to make it working.

Backbone.js works with JSON type data. It modularizes the data once we declare for which models are created. It also works with the nested JSON object. In typical application using backbone, one has to declare a model, a collection and a view separately . View can be made to listen changes by binding events. Views can be made to listen to the changes on models or collections and render on manipulations on models or collections. So its jus simple like a contraption i.e. all the independent components are connected to each other by means of events and on just a single user action the chain of events take place in the background without user knowing it.

V.UNDERSTANDING ROUTERS

As we have mentioned above that it's a complete single page application, we need to work on different views available. Now in our application we are having tabs which provide three different views:

1. Server view
2. Device View
3. Agent View

To switch between these views we need a mechanism that would help us route to the appropriate view selected by the user without changing the page.

VI.NEED OF REQUIRE.JS

Require.js helps the application to be modular by allowing dynamic linking and loading of JavaScript files. Require help the high maintenance of the project i.e. the linking of the JavaScript files which required at the runtime instead of loading them on the page load. Using require.js we can scale up the level of the application we need. By using this, it is possible to be able to port our application in any machine and the application would be working properly-with some minor changes required at the backend. To start up with require one needs to manage the folder structure of the application so to keep the simplicity.

Folder structure: Folder structuring helps to package the required files for the application. With a neat folder structure it is simple to keep the application modular. It is easier to modify the application by editing or adding new modules to it by simply creating new files and linking it to application using require.js.

For the example in the consideration, the need of creating different views, models and collections can be easily fulfilled by defining them in new files and linking them to the application.

VII.MODULAR IMPLEMENTATION OF MVC USING BACKBONE.JS AND REQUIRE.JS

For the example in consideration, we will use

1. **Backbone.js** for MVC implementation.
2. **Require.js** for modularity.

For this application, eclipse JUNO IDE for web development is used.To go ahead to use require.js follow these steps:

1. Maintain '**JavaScripts'** folder to keep all necessary JavaScript files.
2. Maintain the '**libs'** folder in '**JavaScripts'** folder which will contain dependencies like all required libraries such as**jquery.js, underscore.js** and so on.
3. Maintain a '**modules'** folder in 'JavaScripts' folder which will contain all needed implementation modules, in this case, all custom '.js' files for our application.
4. In main **'.html'** file, specify full path of 'EntryPoint' file in 'data main' attribute of **<script>** tag which will use require.js as a loading library and will load the JavaScripts mentioned in 'EntryPoint' file.
5. We need to create three important JavaScript files for rendering three different **View**
    a. ServerView.js
    b. AgentView.js
    c. DeviceView.js

These file will contain the required logic for the specific view. ServerView is our default view. To get data from server, an **AJAX** call would be made in the ServerView which will return the **JSON** object of all the servers. This object is fetched using the collection.Each view fetches data in the same manner.

The 'server' objects fetched from server are mapped with 'server' **Model**. The server model will have same attributes as server object has.

Collection is responsible to fetch the data and populate the Models from the received data. In this case, it populates the server objects into server models on successful response from the server. Collection does this work very efficiently and relieves programmer from manually fetching and populating data from JSON response object.

Routers are responsible to provide the logic to navigate between the three views. Router.js will have all implementation logic and it will be located in 'JavaScripts' folder.

The complete line of action goes like this:

1. On the loading of our main page firstly the data-main attribute i.e. 'EntryPoint'.
2. EntryPoint has paths specifying which file to search in which folder while loading.
3. Then the routers are been called to route to the user to specified view.
4. In routers, all views are loaded.
5. While loading the view '**require'**checks what all JavaScript files are required n it loads only those specified files which are required at this moment

VIII.ADVANTAGES

a. Separation of data, collection of data and rendering logic is done.
b. Separation of data, collection of data and rendering logic is done.
c. Data is loosely coupled with DOM.
d. Application is modular.
e. Application is highly scalable i.e. any number of modules can be added, removed or replaced without affecting implementation much.
f. Dynamic linking is possible.

IX. BENEFITS

a. Application supports all the modern day browsers (Google Crome, Firefox, IE etc.)
b. It works on standard J2EE compliant container.
c. It is Asynchronous Module Definition Compliant.
d. It provides obfuscated code before presenting to the user.
e. It supports HTTP/HTTPS communication protocol.
f. It is themable and configurable.
g. It is resistant and able to recover from the component failure.
h. This application can be extended to be supported on various devices such as mobile phones, PDA's, tablets etc.

X. FUTURE SCOPE

It can also support voice recognition, gesture recognition.

XI. CONCLUSION

By using this interactive approach, a complete platform dependent architectural structure can be designed into a very user friendly, rich, responsive, reliable and a very interactive system. This provides us a thin client application that is deployed on any machine with OS - Windows, Linux, supporting browser - Google Chrome, Firefox,IE.

REFERENCES

[1] Fei Wang, "The Development of Rich Internet Application Based on Current Technologies", International Conference on Web Information Systems and Mining,2009

[2] Luigi Lo Iacono and HariharanRajasekaran, "Secure Browser-based Access to Web Services",NEC Laboratories

Europe, IT
Research DivisionSankt Augustin, Germany,{lo_iacono, rajasekaran}@it.neclab.eu

[3] James McGovern, Sameer Tyagi, and Michael Stevens, "Java Web Services Architecture," peachpit Press, 2004, pp.20-45.

Varsha Machhindra Kuldharme
Dept. of Computer Engineering
Maharashtra Institute of Technology,
Pune – 411038, Maharashtra, India

Pranav Vikas Kamble.
Dept. of Computer Engineering
Maharashtra Institute of Technology,
Pune – 411038, Maharashtra, India

Prateema Sunil Karande
Dept. of Computer Engineering
Maharashtra Institute of Technology,
Pune – 411038, Maharashtra, India

Vinayak Suryakant Kalaskar
Dept. of Computer Engineering
Maharashtra Institute of Technology,
Pune – 411038, Maharashtra, India

Prof.Bhavana Tiple
Dept. of Computer Engineering
Maharashtra Institute of Technology,
Pune – 411038, Maharashtra, India

