5-2017

Online Banking Application with Angular JS, RESTful Web Services and Cassandra Database

Naga Sandeep Singh Bondili
Saint Cloud State University, nsbondili@stcloudstate.edu

Online Banking Application with Angular JS, RESTful Web Services and Cassandra Database

by

Naga Sandeep Singh Bondili

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

March 2017

Starred Paper Committee:
Dr. Dennis Guster
Dr. Susantha Herath
Dr. Kasi Balasubramaniam

Abstract

The purpose of this project is to show how new technologies help the banking and financial industries to process and retrieve data for users. The primary goal of this project is to demonstrate the new technologies Angular JS, RESTful Web Services, and the Cassandra database which are faster than the other conventional software frameworks. This has been achieved by creating two software applications; one is an online banking application with the latest technologies and the other with traditional frameworks. After examining the time taken by the techniques to render the web page to the user, and the availability and scalability of the software, we will understand the need for the new technologies in the banking and financial sectors. The application that is developed should help the users, guide them and should perform user requested transactions with zero fault tolerance; it should give precise data without losing or showing error data. It should also provide security to the transaction. Unauthorized transactions should be prohibited from entering the server either from UI (User Interface) or from a mid-tier connection. The proposed application uses the Cassandra database which has high scalability and availability. It also uses Service Oriented Architecture (SOA) for better service and performance of the application. In the UI we use Angular JS which is fast and can handle the REST (Representational State Transfer) Application Programming Interface CRUD(Create, Read, Update & Delete) operations.

Acknowledgement

The effective finish of this paper could not have been conceivable without the direction of my professors Dr. Dennis Guster and Dr. Susantha Herath. I also would like to thank Professor Kasi Balasubramanian for being part of the committee.

Table of Contents

List of Table

Table Page

List of Figures

Chapter I: Introduction

Introduction

The Online Banking Applications help the bankers and customers to connect and also helps in saving time and money of the customers by avoiding a trip to banks to perform any banking transactions. Banks collect huge amounts of data such as daily transactions, calculating interest, tracking loan, and also in portfolio management. The bank customers can keep track of their funds and manage their funds whenever they want to so long as they are authorized. It should be fast enough to maintain and retrieve this huge amount of data. The application should be smart enough to restrict the access to users who are not permitted to perform such actions. In this application, we are using Spring Security, which performs specific actions when the user has roles assigned and which are allowed to carry out an action. If the user does not have permission, the application prohibits the action by giving an error message to the user. This application was developed using Service Oriented Architecture (SOA). It can have one or more services associated with each request. Here we use spring security to add security to the service. Moreover, this service contacts mid-tier architecture to fetch data from the database.

Problem Statement

In the current scenario, the database availability and scalability is a big problem. Some banks restrict their user's transaction statement to a particular period instead of allowing them to view the entire report. For example, they may be limited to a period of 5 years. Sometimes the availability of the database service is also a big problem.

Nature and Significance of the Problem

The importance of this problem is availability and scalability of the database. At the same time, it should provide maximum security to the data and application without any malicious activities or unauthorized data entering into the server.

Objective of the Study

The objective of the study is to increase the efficient use of technology in improving the Online Banking Application.

Study Questions/Hypotheses

Before implementing this project, there were also questions like:

1. Does Cassandra help in data availability for large amounts of data?
2. Will Angular JS be quicker than present UI technologies?
3. Does the REST service help in multipart requests?

Summary

In short, this paper develops a web application that uses Cassandra, Angular JS, and Restful Web services which help in storing, retrieving, and displaying data without much time delay or user wait time.

Chapter II: Background and Review of Literature

Introduction

The online banking system has changed a lot over the years, helping bankers and customers by reducing their effort, time, and cost. But at the same time, it should be reliable, efficient, and secure. The Online Banking Application should be efficient and reliable allowing zero fault tolerance.

Background Related to the Problem

Banks have many operations such as loans, fixed deposits, portfolio management, mutual funds, etc. After the growth of computers, the bookkeeping or recording of every transaction is now done by computers (DeCandia et al., 2007). After the Internet emerged, they slowly moved to a centralized environment followed by a move to a more distributed environment. Banks are currently using a distributed environment. However, the problem is the Clusters (a group of servers) they are using which are not sufficient and they may need to keep adding servers to their data centers. There is a saturation point for the Relational Database Systems (RDBMS) Park, Nguyen, & Won, 2015). Even if it is scalable for a large amount of data, it takes a lot of time to process the request, and it is difficult to maintain copies in case you need to protect or backup the data.

Current System Problems

The current system problems are (a) data unavailability, (b) less scalable, and (c) a heavyweight process.

Proposed System

In the proposed system, these are the following advantages:

1. Using the Cassandra database, it helps in database scalability and processes high volumes of data in a small amount of time (Ramanathan, Goel, & Alagumalai, 2011).
2. The data has multiple copies, so there is no problem of losing data.
3. If the data center fails, it automatically fetches the data from another data center.
4. Using SOA helps in improvised data flow and also adds flexibility to the application.
5. Using Angular JS in the UI helps in processing the REST API and it can perform create, read, update, and delete (CRUD) operations without moving to another page.

About the Technologies

Apache Cassandra. Apache Cassandra is an open source distributed database management system, which is used to store large amounts of data on different servers. It provides high availability with no single point of failure. It allows asynchronous master-less replication (DataStax Academy, n.d.). It was developed by Facebook in 2008.

Cassandra Query Language (CQL) is used to interact with the Cassandra database. The most common way to connect to the Cassandra database is using a shell (cqlsh).

Following are some of the key points about the Cassandra database:

- It stores in key value pairs.
- It is a distributed design and is inspired by the Amazon Dynamo database and Google's Big Table (Chang et al., 2008).
- It is not a relational database.
- It is highly scalable, consistent, and zero fault tolerant.
- It is Cloud-enabled.
- It has high performance.

Cassandra architecture. Cassandra handles a lot of workload across multiple nodes. Cassandra addresses the issue of failures by utilizing a shared peer-to-peer framework crosswise over homogeneous nodes where information is disseminated among all nodes in the cluster. Every node often trades state data about itself and the other nodes over the group by utilizing a peer-to-peer gossip communication protocol (Lakshman, 2008). A consecutively composed submit of every sign on, every node catches write activity to guarantee information durability. The data is then written into the commit log first and then to the mem table—it is like cache memory when it is full the data is flushed to the SS table (Sorted String).

Gossip protocol. Gossip protocol helps to communicate between each node each second it transfers state messages between the nodes. The gossip procedure runs each second and trades state messages up to the other nodes in the cluster. A Gossip Protocol has a form connected with it so that amid a gossip business, older data is overwritten with the most recent and flow state for a particular node. As a matter of course, a node recollects different nodes it has gossiped with between consequent restarts.

Failure detection. Failure detection is a technique for locally deciding from the gossip state and history if another node in the framework is down or has returned back up. Cassandra utilizes this data to abstain from routing client solicitations to remote nodes at whatever point possible. The gossip procedure tracks the state from different nodes both specifically and by implication. Cassandra utilizes a gathering recognition instrument to figure for each node the limit that considers system execution, workload, and historical conditions. Amid the gossip exchange, each node keeps up a sliding window of between entry times of gossip messages from different nodes in the group.

Configuring the phi_convict_threshold property modifies the affectability of the failure detector. Lower values improve the probability that a lethargic node will be set apart as down, while higher qualities diminish the likelihood that transient failures are bringing on a node failure. Node failures can come about because of different causes; for example, equipment failure and system blackouts. Node blackouts are regularly transient yet can keep going for broadened periods. At the point when a node returns online after an outage, it might have missed writes for the replica information it maintains. A repair mechanism exists to recover the lost information for example: indicated handoffs and manual repair with node tool repair.

Data distribution and replication. Data distribution and replication goes hand-in-hand within Cassandra. Information is sorted out by a table and recognized by a primary key, which figures out which node the information is stored on. Replicas are duplicates of rows. At the point when information is first composed, it is additionally alluded to as a replica. Snitch is the crucial feature, it checks the health of all the clusters and reads the best available model. Replica placement strategy is where the data should be stored in different groups to avoid single point failure to make the data available. Replication factor decides how many copies of data you want to store in different nodes. Partitioner determines how and where your data should be stored; it decides where the first copy of data goes.

Virtual nodes. Virtual nodes, known as Vnodes, appropriate information across nodes at a better granularity than can be efficiently accomplished if figured tokens are utilized. Vnodes simplify many errands in Cassandra. Tokens are consequently computed and allocated to every node. Rebalancing a cluster is, therefore, important when including or evacuating nodes. At the point when a node joins the cluster, it accepts its obligation regarding an even partition of

information from other nodes in the cluster. On the off chance that a node comes up short, the heap is spread uniformly across different nodes in the cluster. Rebuilding a dead node is quicker because it includes each other node in the cluster. The extent of nodes allocated to every machine in a cluster can be relegated, so smaller and bigger PCs can be utilized as a part of building clusters.

Data replication. Data Replication is another important feature in Cassandra. Cassandra stores replicas on various nodes to guarantee consistent quality and adaptation to internal failure. A replication strategy decides the nodes where replicas are placed. The aggregate number of copies over the bunch is alluded to as the replication variable. A reproduction component of 1 implies that there is one copy of every column and the only duplicate of every column in the cluster. A replication factor of 2 means two copies of every column, where every duplicate is on an alternate node.

- Simple Strategy: Utilize just for a single data server and one rack. On the off chance that you ever expect more than one server farm, utilize the NetworkTopologyStrategy.
- NetworkTopologyStrategy: Highly prescribed for maximum deployments since it is much less demanding to grow to different server farms when required by future development.

Partitioner. Partitioner decides how information is distributed over the nodes in the clusters. A partitioner is a capacity for inferring a token speaking to a column from its partition key, commonly by hashing. Every line of information is then circulated over the cluster by the value of the symbol The Murmur3Partitioner, which gives quick hashing and great execution

- The RandomPartitioner conveys information uniformly over the nodes utilizing an MD5 hash estimation of the row key.
- Cassandra gives this partitioner to requested apportioning, it is incorporated for backward compatibly.

Write Operation in Cassandra is little different than in a normal relational database. When the write operation takes place, it is stored in the commit log permanently; the data is written to the mem table. The mem table has a threshold limit that is predefined. When the predefined limit is exceeded it flushes the data to the SS Table (Sorted String).

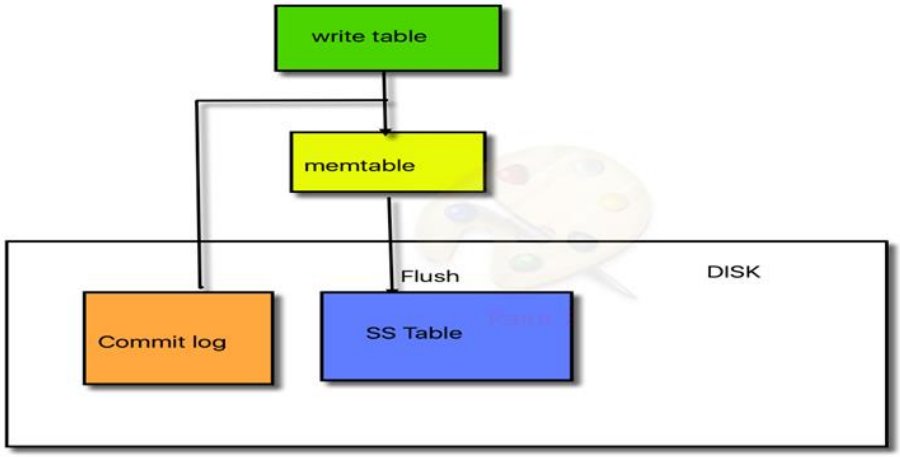

Figure 1. Write Operation

Compaction is another important feature used to maintain the SS tables. Whenever there is an update, it does not over-write by deleting the existing row; it creates the new row with the updated timestamp, and the old version of the data is marked as a tombstone, which is eligible for deletion. There is a process, which regularly runs, which looks for tombstones and deletes them from the database. Each data will be inserted with the timestamp; this helps the read process to select the most recent time-stamped data, and the old data will be marked for deletion.

When the data is marked for deletion, it gives an update to the other nodes, to maintain parity. To Read/Write Cassandra uses the Chubby Protocol to interact with the nodes. The Protocol selects the master node. The master node is then responsible for sending or receiving the data from all other nodes (Hakka Labs, 2014).

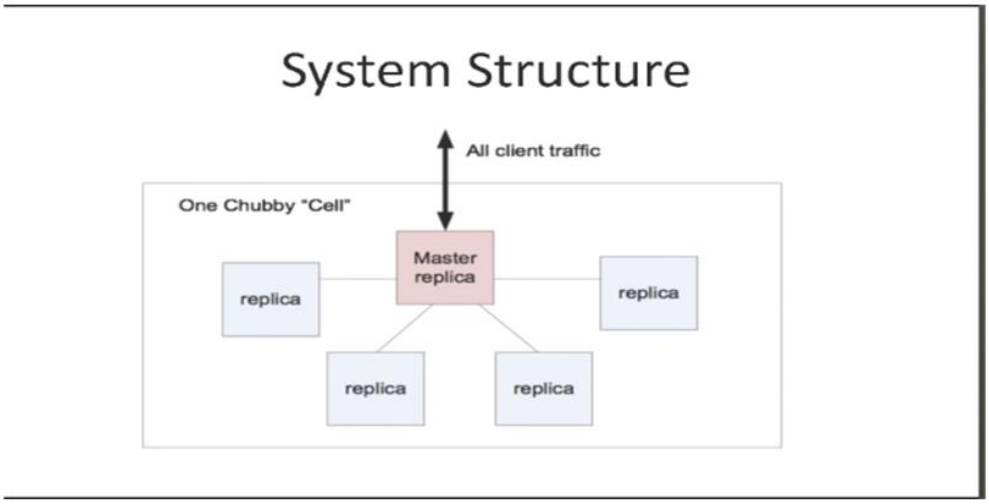

Figure 2. Chubby Protocol Architecture

The chubby protocol uses the Paxos algorithm to select the master node. The Paxos algorithm has a client which proposes the data to one of the nodes which are responsible for creating the replicas. This node is called Proposed. When the Proposer offers a data value, the acceptor should accept the value.

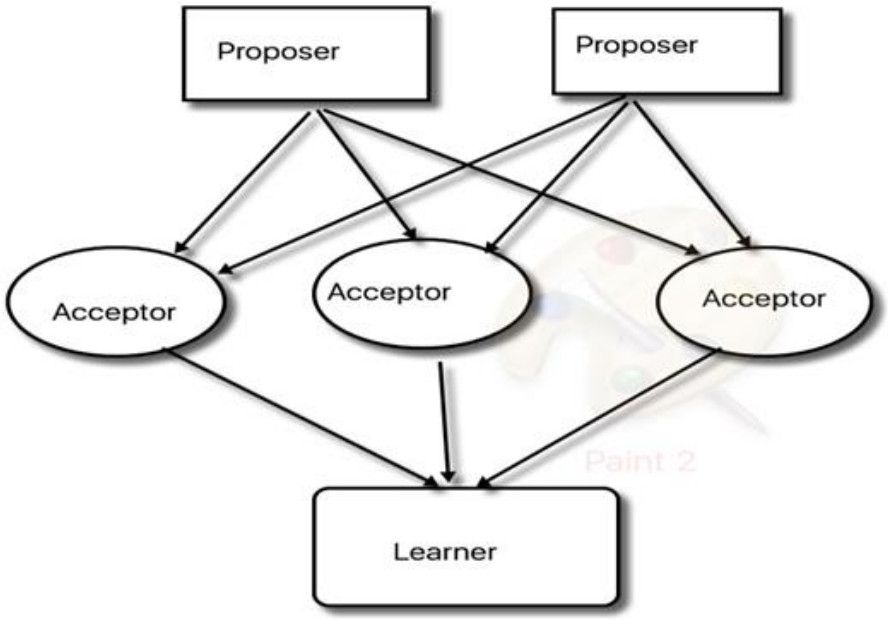

Figure 3. Paxos Algorithm

When the majority of nodes accept the value, the proposer becomes the master node, and the data will be adopted by all of the other acceptor nodes.

Java

Introduction to Java. Java is a programming language invented at Sun Microsystems Inc. by Bill Joy, James Gosling, Mike Sheridan, and Patrick Naughton. Java's first version 1.0 was released on January 21, 1996. And the latest version is Java SE 8 which was issued on March 18, 2014. Sun MicroSystems Inc. released the Java based web browser in 1995 which can run applets in the browser. Later, many web browsers incorporated the use of Java.

Features of Java.

1. Simple: Java is simple because it uses mostly C++ syntax for coding. It avoids complex structures and pointers and also avoids redundancy inside the code. Java has a garbage collector which reduces the developer's responsibility to clear memory.
2. Object Oriented: Java is an Object-Oriented Programming language which means it creates objects. All the code or set of instructions is written in a class. We can say a class is a block which has the collection of instructions and these classes act as a blueprint to create objects. Each object has characteristics of the class. These objects maintain the data inside the defined variables. Each object created from the same class has different references.
3. Distributed: Java can be used in a distributed environment because it supports protocols like TCP/IP and UDP. It can transfer the information from one machine to another; this helps the programmers to communicate and execute different operations on different computers.
4. Interpreted: After a developer writes the code, he compiles the code using the Java compiler, which generates bytecode. This binary code is neutral and can be executed on any machine which has the Java runtime environment. In this way, it avoids the distribution of source code.
5. Robust: Java is strong when compared to many other programming languages. It has good exception handling techniques during runtimes which does not allow the application to crash so it helps the programmers to create highly reliable applications.

6. Security: Java provides security by avoiding attacks like tampering with the data, and impersonation. Java programs run only on the Java runtime environment and interact very little with the operating system. Java does not provide pointers, hence the ability to access the objects requires the proper security role.
7. System Independent: When the Java code is compiled, it creates files with the "class" extension. These files contain byte format code, which can be run on any platform. It may be run on Linux, Windows, or Macintosh operating systems. These "class" files need the Java runtime environment for the execution process.
8. Portability: In Java, when the program is executed, the result is the same on any given machine. For instance, the Java programming language has the primitive data type which is 32 bits in size, which is consistent across all other platforms. Hence, we can say it is very portable in nature.
9. High Performance: Java has an interpreter inside its Java Virtual Machine (JVM) which is slow. To eliminate this problem, Java Soft introduced the Just-In-Time (JIT) compiler. Whenever a Java method is called while executing the program, this JIT compiler converts the byte to native machine code instead of converting all the code to machine code. JIT helps in implementing the programs faster.
10. Multi-Threaded: A thread can be referred to as a process executing a set of instructions. It can run multiple such threads simultaneously.
11. Scalability: Due to Java's compact and platform-independent nature, it can be installed on any machine such as mainframes or mobile devices. It can run on many

processors simultaneously. As it supports multi-threading and even the JVM supports multi-threading, this helps in scheduling and memory management.

12. Dynamic: Java is a dynamic language because it can solve the operations which can change dynamically, that means, it can take different inputs and produce the results accordingly.

Java virtual machine. Java Virtual Machine is the critical component in the Java runtime environment. It can be considered as the heart of Java. It provides the runtime environment for the Java code. Whenever any Java command is executed, a JVM instance is created. It provides various functionalities like loading, verifying and executing the code. It provides memory area, method area, and garbage collection.

***Internal architecture of JVM*.**

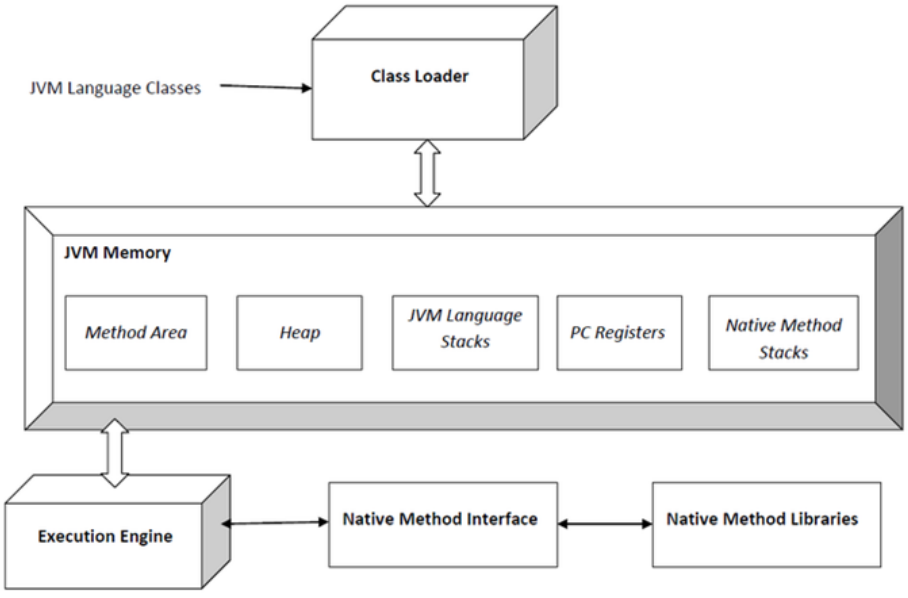

Figure 4. Java Architecture (Java Virtual Machine, n.d.)

1. ClassLoader: When the Java code is scheduled to execute, it creates a JVM instance and the JVM instance loads the “class” files. It searches the Method area location related to the program.
2. Method Area: JVM store the information of loaded class types in a logical memory area called the Method Area. The JVM looks in the method area as it executes the application. For each method, it has a specified information like a method name, the return type of method, and input parameters and access modifiers of that particular method.
3. Heap: Heap memory is created whenever the class instance is created by running the Java application. It stores the objects of the application. This is shared by each thread in Java. There is a separate heap built for each running application. So no two applications can access the same collection. The Java code can only create memory inside the heap, but it cannot destroy or release the memory it is using.
4. Stack: When the program starts executing, JVM creates a thread. This thread invokes the methods and stores its instances in the stack frame. This is new stack frame and then cited as a current frame. This stores the information about the parameters, local variables, and results. It is destroyed whenever the invocation is completed.
5. Program Counter: It stores the information/address of the instruction that is currently being executed.
6. Native Method Stack: Native method refers to that code which JVM does not control the access rights. It is typically written in C/C++. We can call the native method

which is written in C from JVM so it goes back and forth from JVM for executing the native method.

7. Execution Engine: The byte code loaded into the runtime environment is executed instruction by instruction. It is like a virtual processor and uses JIT Compiler or Interpreter depending upon the need of code execution.

Object Oriented Programming (OOPs) in Java.

Classes. Class is a block of statements which is responsible for performing operations. The structure of a Java Class includes (a) package, (b) import statements [consuming libraries]. (c) comments [description about the class], (d) class, (e) variables, (f) constructors, and (g) methods.

Package. A package is a perspicuous container of the set of classes. In other words, we can call it a folder which maintains classes. It is always defined at the top of every class, defining which package the class belongs to. It cannot be mentioned inside the class or after the class.

In Figure 5, we can see the package name and the class defined after the mentioning the package name.

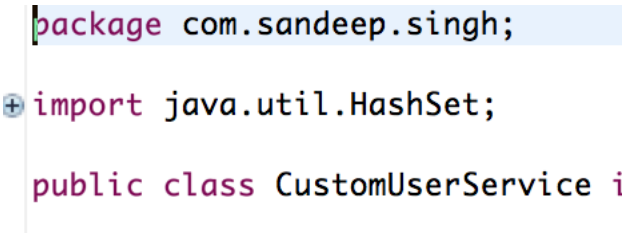

Figure 5. Package Name and Class Defined

In Figure 6, we can see the folder structure and how the classes reside inside a package.

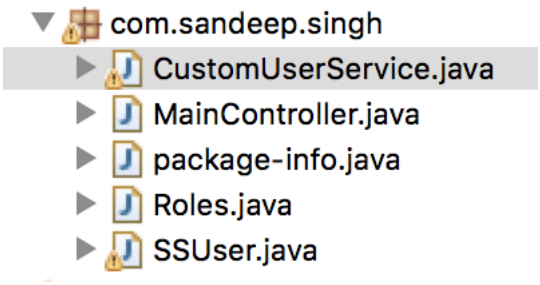

Figure 6. Folder Structure

Import. Import statement is nothing but invoking the library functions and classes. This library can be termed as different packages. In Java, to avoid redundant code, we incorporate the same functionality which is defined in other packages to the current class. This helps to avoid writing the same code multiple times.

Comments. Comments are part of classes which are not compiled or executed by the Java. Those are plain texts which describe the functionality of class for future reference. This acts as a documentation of the class.

1. **Classes:** A class is a block of statements which has variables and set of instructions in it. In Figure 7, we can see the Class declared with Access modifier (Public), which means who can access the class, the key name ***class*** should be defined before the name of the class.

```
1  package com.sandeep.singh;
2  /* This class is just a demo and
3   * does nothing
4   */
5
6  public class DemoClass {
7
8  }
```

Figure 7. Class Declared with Access Modifier

- **Variables.** Variables can be considered as the reference or name of the memory locations, where the data of the classes is stored. Each variable has a particular type, which stores either numbers or strings or just Boolean values.
- **Constructors.** Constructors are the instance methods. Whenever the class is instantiated the constructor sets the values of the objects to the variables or members of the class.
- **Methods.** A Method can be considered as a subprogram which can be executed only when it is called by its method name. Each method has its name and implementation. In Figure 8 you can see the method defined inside the class. It is first determined by the access modifier Public and then followed by the return type, that means what type of value it is going to return after executing this method, then the method name should be given. Inside the parenthesis, we can specify the parameters that we are passing to the method as an input.

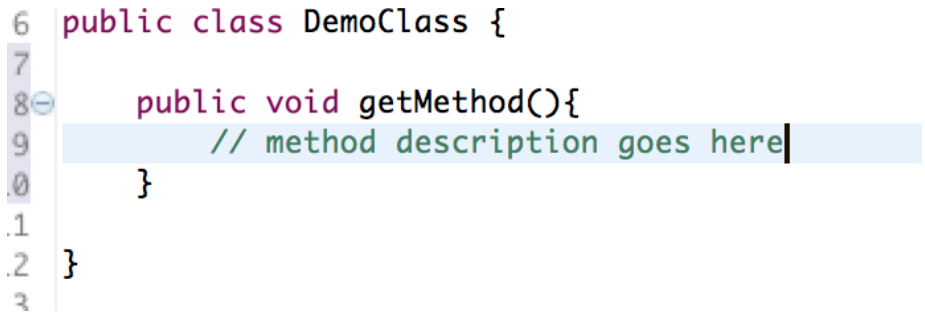

Figure 8. Method Defined Inside the Class.

2. **Object:** Object is the actual behavior of the class. We can say that the class acts as a blue print, and the object is the real-time existence of the instance of the class. It has the same behavior of the class. Whenever an object is created, there is separate

memory allocated to that object and its variables in the heap memory of JVM. The Syntax to create an object is as follows:

ClassName object= new ClassName();

"new" key is used to create objects.

3. **Inheritance:** The process of acquiring the properties of the parent class into the child class is called inheritance. This helps in reusing the classes instead of creating a new one. As an analogy, we can say that a Honda car is a parent class and it has properties such as tires, engine, doors, and steering wheel which are standard components. This class can be inherited to the child class of an Acura TL. Does the Acura have the same properties like tires, doors, engine, and steering wheel? Acura uses many of the same parts as Honda; that means many of the same properties of the parent class can be reused in the child class. This reduces the developer's efforts to create the same piece of code twice and also helps in reducing the redundant code.
4. **Polymorphism:** Polymorphism means having multiple forms. In Java, polymorphism means having the same method name with different parameters and return type. It has two types of polymorphisms:

   OverLoading: The process of having different signatures is called overloading.

   ```
   Class ClassName{
   void draw(double d){
   // method signature
   }
   void draw(int i){
   ```

```
// method signatures
}
}
```

Depending upon the parameters passed, compiler selects the method.

Overriding: When we import the parent class and use the same method name of the parent class method and changing its implementation in the child class.

```
Class A{
Void method1(){
}
}
class B extends A
{
void method(){
}
}
```

5. **Abstraction:** Abstraction is hiding the method or details of the class. It exposes the features of classes. The abstract classes start with the keyword Abstract before the class. The abstract class should contain at least one abstract method inside it. The abstract methods are method declarations but not definitions. It may contain normal methods as well.

6. **Encapsulation**: Encapsulation is a process of binding the data inside the class or a method and helps it work as a single unit of data. It contributes to protecting the data from unnecessary modification. We call these classes as wrapper classes as it wraps the data inside it.

Java Database Connectivity (JDBC). JDBC is an interface which allows programmers to connect to the database and helps in performing database transactions. JDBC needs drivers that allow it to connect to the database. There are different JDBC drivers available:

- Type 1 Driver/JDBC-ODBC driver: This type of connection depends on the client side libraries. This means the operating system should have required libraries of the file which support the connectivity to the database. Here, we create Data Source Name (DSN) which represents the database. When creating the DSN we specify the name and location of the database. This is only used for experimental procedures on a single machine.
- Type 2 JDBC Native API: This type of connection depends on upon the client side libraries, but these drivers are provided by the database provider. These libraries are unique to each database. The advantage of this type of driver is, it is considerably faster than the type 1 driver
- Type 3 Network Protocol Driver: This kind of driver does not require any libraries installed on the client side. It uses the middleware application servers for the database transactions. By this approach, we can connect to any database. The user requests are then converted into database specific commands and retrieve the data and send it back to the user.

- Type 4 Database Java Driver: This is also called a Direct to Database Java Driver. This helps in converting the database call from Java to database provider specific protocols. It is purely platform independent and is also better than Type 1 or 2 drivers.

Java new features. Java lambda expressions are important components incorporated into Java SE 8. A lambda expression gives an approach to speak to one technique interface utilizing an expression. A lambda expression resembles a strategy; it gives a rundown of formal parameters and a body (which can be an expression or a piece of code) communicated as far as those parameters. A lambda expression can be comprehended as a brief representation of a mysterious capacity that can be passed around. We do not need to specify any name to the anonymous class that we create.

For example, let us say we have an interface with the name Calculate and we want to implement this in a Multiply class. Until Java 7, the following was the implementation:

```
interface Calculate{
int operation(int x, int y);
}
This interface is implemented in the following class
Class Multiply implements Calculate
{
    @override
    public int operation(int x,int y){
    return x*y;
```

```
        }
        public static void main(String args[]){
        int x=5,y=10;
        int s=operation(x,y);
        }
}
```

We are overriding the method of interface in the base class and passing the parameters to it. But in Java 8, using the lamda expressions, we can eliminate the long verbose code. Below is the example of the class implementing the same interface Calculate.

```
Class Multiply implements Calculate
{
        public static void main(String args[]){
        int x=5,y=10;
        int s=(x,y)->{ return x*y; }
        }
}
```

In the above example, we can see the we (x,y) are the input parameters that are being passed, the System will infer the type of the parameters and prints the x and y values.

The code between '{……}' is the code that is going the be executed after '->' sign.

It makes sorting even easier with very little code. Let's see how sorting can be done in Java 7:

```
Collections.sort(list,new Comparator<String>(){
@Override
public int compare(String s1,String s2){
return s1.compareTo(s2);
}
})
```

But we can implement this even easier in Java 8:

But we can implement this even easier in Java 8:

```
Collections.sort(list,(String s1,String s2)->s1.compareTo(S2)));
```

After comparing both of the Sort codes we can understand how much code can be eliminated. If the user wants to use the same interface, but with different implementation, he can use these lamda expressions whenever he wants.

Default methods. Java 8 added Default methods to the interfaces. This helps to use the lambda expressions extensively. It helps to enable the features of the interface without breaking the class implementation. Let us look at an example of interfaces with the default method:

```
Interface Calculate{
        default void display(){
        System.out.println(a+b)
        }
 }
```

The following class shows how the default method is being invoked or executed.

```
class Hello implements Calculate.
{
        public static void main(String args[]){
        Calculate.display();
        }
}
```

An interface can have more than one default method in it.

Streams. The stream is another conceptual layer presented in Java 8. Utilizing stream, you can prepare information decisively. It corresponds to an arrangement of objects from a source, which bolsters the total operations. A stream gives an arrangement of components of a particular sort in a sequential way. A stream registers components upon request, though it never stores the elements. The vast majority of the stream operations return the stream itself so that their outcome can be pipelined. These transactions are called transitional operations, and their capacity is to take the center, handle them, and return yield to the objective. Gather() strategy is the last service which is ordinarily present toward the finish of the pipelining operation to stamp the finish of the stream. Its operations do the iterations inside over the source components given, as opposed to Collections where unequivocal emphasis is required. Let us look at the following example.

```
Example:  List<String> someList=Array.asList(“hello”,”how are you”);
List<String> result = someList.stream()
  .filter(e -> e.equals(“hello”))
```

```
.map(e -> e.toUpperCase())
.collect(toList());
```

In the above example, we will take the list and pass it to the stream object, filter, manipulate the object and send back the result. Below are the Important operations of Stream:

- Map the stream components into something else; it acknowledges a capacity to apply to every last part of the stream and returns a surge of the qualities the parameter work created. This is the bread and margarine of the streams API; delineate you to play out a calculation on the information inside a stream.
- Reduce plays out of a diminishment of the stream to a solitary component. You need to total all the whole number values in the stream–you need to utilize the lesser work. You need to locate the most extreme in the stream–diminish is your companion.
- Filter gives back another stream that contains a portion of the components of the first. It acknowledges the predicate to computes which components ought to be returned in the new stream and removes the rest. In the imperative code, we would utilize the contingent rationale to determine what ought to happen if a component fulfills the condition. In the practical style, we do not trouble with uncertainties, we channel the stream and work just on the qualities we require.
- Collect is the best approach to escape the streams world and acquire a solid accumulation of qualities, similar to a rundown in the case above.

Nashorn. Nashorn, a tremendously enhanced javascript motor is presented, to supplant the current Rhino. Nashorn gives better execution, as it straightforwardly accumulates the code

in memory and passes the bytecode to JVM. Nashorn utilizes summon active component, acquainted in Java 7 with enhancing execution.

REST service. Rest Service is a stateless Web Service. It means the service itself handles the request using the query string or header. Whenever there is a request to the service, it is considered as a new application, and it is processed. Each application might have the same headers, body, or query strings and when we make a call to the rest service it always considers the request as a new claim (Jersey.java.net, n.d.). Rest Service works on code on demand, a request always points to the specific method which executes specific functionality. Depending upon the request and parameters, we will handle the request.

For instance, let us see the below URL:

www.localhost:8080/restservice/myMethod/params

In the above URL, the rest service refers to the rest service project and my method refers to the specific method and parameters.

The code would be as follows:

```
@Path(“/”)
Class RestService{
@GET
@Path(“myMethod/[params]”)
public String myMethod(@PathParam(“params”) String params){
return “response”;
}
}
```

You can see in the above function, when the service is requested, depending upon the URL, it directly points to the particular method along with requested parameters.

There are primarily four types of request which can be made using REST Service: (a) GET, (b) POST, (c) PUT, and (d) DELETE.

GET. GET is used to retrieve the information. Whenever the request is made, it sends the response depending upon the user passed parameters, it passes the response either in JSON or XML format or just any string.

Example:

GET http://www.example.com/resource/123456

5.6.2. **POST**

POST. POST is used to create a new resource. It is also used to create subordinate resources. The main advantage of a POST call is that we can send the request in headers and bind the data.

Example:

POST http://www.example.com/resource/123456

But in the header there will be data that needs to be POSTed. On successful creation, it will return the 200-status code.

5.6.3. PUT

PUT. PUT is used to update any resource rather than creating any resource. It contains the data of the resource which is going to be updated.

Example:

PUT http://www.example.com/resource/123456

DELETE. Delete is used to delete any resource or record. It is pretty straight forward, we will pass the resource id which needs to be deleted.

GET http://www.example.com/resource/123456

Advantages of RESTful WebService

- You can have scalability without effecting the other modules
- It is a light weight
- Easy to build
- It has high performance when compared to conventional web services
- By using REST we can perform CRUD

Spring Security

Spring Security is a Security framework for Java EE framework which was introduced in the year 2004. This is mainly used for authentication and authorization activities. This helps to provide login and logout functionality to the application. It controls and restricts the user action on the application depending the security role of a particular user. It also helps to intercept the service requests and authenticate. Advantages of Spring Security

- Helps in Authentication and Authorization
- IT is Open Source Security Framework
- Easy to maintain
- Helps in developing loosely coupled Applications
- Easy to develop

There are two types of Authorizations—Method Level Authorization and URL Level Authorization.

Method Level Authorization is used to secure the methods. Following is an example to illustrate the method level Security:

```
public interface SecuredInterface{
@PreAuthorize(“hasRole(‘user_role’)”)
public String method();
}
public SecuredClass implements SecuredInterface
{
public String method(){
return “you are authorized”;
}
}
```

PreAuthorize is the method used to authenticate the user before accessing the method. It uses “hasPermission” to check for the user role. The user roles are passed as in the header of the request along with the user id, matches the user id and roles in the header with the user id and roles in the database. If they are matched. the request is authenticated and the method will be executed. Similarly, we can use “PostFilter.”

In PostFilter, when we want to filter the results after executing the method, for example if the user has access to the method and does not have access to the records, then the PostFilter will remove the records for which the user does not have access to.

URL Level Authorizatio**n** is used to authenticate the user when he tries to access the URL. For this we use SiteMinder. SiteMinder is another concept in Spring Security. It is used to

intercept incoming requests and authenticate the URL whether the request is passed through the preferred or authorized client. It looks for the headers in the request which has security tokens. The tokens are encrypted information. Then the SiteMinder decrypts the token and checks the authorized roles with the user id and authenticates the request and then the request is passed to the specified method for further processing. Following is the security xml file which is inherited into the web.xml file.

```
<http auto-config='true' >
    <intercept-url pattern="/**" access="ROLE_USER" />

    <custom-filter ref="siteMinderFilter" position="PRE_AUTH_FILTER"/>
  </http>
   <beans:bean id="siteMinderFilter"
class="org.springframework.security.web.authentication.preauth.RequestHeaderAuthentication
Filter">
      <beans:property name="principalRequestHeader" value="SM_USER"/>
      <beans:property name="exceptionIfHeaderMissing" value="true"/>
      <beans:property name="authenticationManager" ref="authenticationManager" />
 </beans:bean>
  <authentication-manager alias="authenticationManager">
  <authentication-provider ref="preAuthenticationProvider"/>
  </authentication-manager>
```

```
<beans:bean id="preAuthenticationProvider"
class="org.springframework.security.web.authentication.preauth.PreAuthenticatedAuthe
nticationProvider">
  <beans:property name="preAuthenticatedUserDetailsService">
    <beans:bean id="userDetailsServiceWrapper"
class="org.springframework.security.core.userdetails.UserDetailsByNameServiceWrapp
er">
      <beans:property name="userDetailsService" ref="customUserService"/>
    </beans:bean>
  </beans:property>
 </beans:bean>
 <beans:bean id="customUserService"
class="com.sandeep.singh.CustomUserService"/>
  </beans:beans>
```

From the above xml file, we are expecting the SM_USER header in the url. Now, if the url has the header SM_USER, it checks for the value in it and then it passes the value to the CustomUserService class. The CustomUserService class checks the value in the SM_USER and authenticates if it matches predefined security roles of the particular user. If it does not match, it will then throw a 403 authentication failure status code as a response to the request.

Angular JS

Angular JS is a web application framework, currently maintained by Google. It is used to build the dynamic web application in a Model View Controller way. Angular JS helps in data

binding and dependency injection which is used to eliminate much of the code required. It is a powerful Java script which helps in building rich Internet applications (Angular JS, n.d.-a).

Features of Angular JS:

- Scope is the technique of gluing the objects between a controller and a view.
- Data Binding is a concept of automatically synchronizing the data between a model and the view.
- Angular expression is another important feature, which helps to print or calculate the data from the model in the view.
- Angular JS has many directives such as ngModel, ngBind and in a similar way it also helps us to create new directives.
- $route is another important feature, which helps in navigating the view.
- Filters in Angular JS helps us to filter the records depending on the user selection.

Provider in Angular JS is similar to the classes in Java. They are responsible for making the various components work. We primarily use Factory and Service. This helps to connect to web services and fetch the data and this data is modeled and bound to the view. Figure 9 shows the work flow of the Angular JS network.

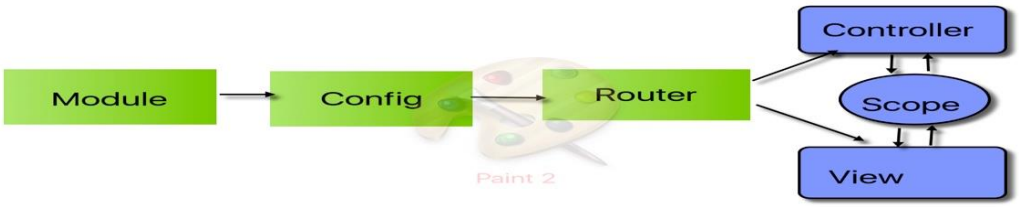

Figure *9.* Angular JS Work Flow

To consume a REST service in Angular JS, we use Factory method or http method.

Following is the example to demonstrate how $resource method consumes a REST Service. We need to remember that the REST Service can return either JSON (Java Script Object Notation) or XML format data.

```
Function getData(){
var user=$resource('myrest/api',{parameters:@value});
var data=user.get({parameter: paramValue},function(){
$scope.value=data.jsonData;
});
}
```

In the above example, the $resource fetches the data from the REST Service and stores in the data object and from the data object we retrieve the data and store it in some other variable and format accordingly.

Data Binding: Data Binding in Angular applications is the programmed synchronization of information between the model and view parts. The way that Angular executes information restricting gives you a chance to regard the design as the single-wellspring of-truth in your application (Angular JS, n.d.-b). The view is a projection of the model at all times. At the point when the model changes, the view mirrors the change and the other way around.

Most templating frameworks tie information in stand-out heading—they consolidate layout and model segments together into a view. After the merge happens, changes to the design or related areas of the view are not consequently reflected in the view. More problematic, any progressions that the client makes to the light are not reflected in the model. This implies the

designer needs to compose code that continually adjusts the view with the model and the model with the view. The model is the single-source of truth for the application state, significantly disentangling the programming model for the engineer. You can think about the view as a momentary projection of your model. Because the view is only a projection of the model, the controller is entirely isolated from the view and ignorant of it. This makes testing a snap since it is anything but difficult to test your controller in seclusion without the view and the related DOM/program reliance.

Example:

Inside the HTML page or view, we need to follow the following syntax.

```
<div ng-bind="hello"></div>
```

And inside the controller,

```
var app=angular.module('myApp',[]);
app.controller('myCtrl', function($scope){
$scope.hello="Say Hello!!"
});
```

Component: A component is fundamentally a directive that uses a less complex arrangement, and that is reasonable for a segment based architecture, which is the thing that Angular 2 is about. Think about a component as a widget: A bit of HTML code that you can reuse in a few better places in your web application.

Example:

```
angular.module("myApp", []) .component("hello",{ template: 'Hello!' });
```

in the view we define,

<hello></hello>

Routing Angular JS is a single page application. When we need to redirect to the different page, the page should not get refreshed and it should not update the page. Instead, it should include the HTML page inside the current view. $routeProvider helps in doing this, as we can see in the below example, we use the config component to route the page. When we click on the link instead of redirecting to the new HTML page, the routeProvider renders the HTML page, and the end user cannot see any page refresh or reload. However, he can see the required output.

Example: First we need to add <ng-app></ng-app>

Following to that create hyperlinks or any URLs where you want to redirect the page. Inside the controller, write adds the following script.

```
var app = angular.module(myapp, []);
 app.config([‘$routeProvider’,
  function($routeProvider) {
    $routeProvider.
      when(‘/order’, {
        templateUrl: ‘/order.html’,
        controller: ‘OrderController’
      }).
      when(‘/cart, {
        templateUrl: /carthtml’,
        controller: ‘cartController’
      }).
```

```
    otherwise({
      redirectTo: ‘/home
    });
  }]);
```

Services: Services are javascript functions and are dependable to do particular tasks as it were. Services make them an individual substance which is viable and testable. Controllers, channels can call them as they are required. Administrations are regularly infused utilizing the reliance infusion system of Angular JS. Angular JS gives numerous prebuilt administrations to illustration, $http, $route, $window, $location and so on. Every administration is in charge of a particular assignment for instance; $http is utilized to make an ajax call to get the server information. The $route is employed to characterize the steering data. Inbuilt administrations are regularly prefixed with the $ image.

Example:

```
mainApp.service(additionService, function(){
  this.square = function(a) {
    return a+a;
  }
});
```

Custom Directives: Custom directives are utilized as a part of Angular JS to develop the usefulness of HTML. Custom directives are characterized using the “directive” function. A custom directive essentially replaces the component for which it is enacted. An Angular JS application amid bootstrap finds the coordinating components and do one-time action utilizing its

gather() strategy for the custom mandate then processes the component using the join() technique for the custom order in light of the extent of the decree. Angular JS gives support to make custom orders for taking after the sorting of components.

Example:

```
 angular.module(‘myDirective’, [])
.directive(‘order’, function() {
   return {
      restrict: ‘E’,
      scope: {
         data: ‘=‘
      },
      templateUrl: ‘templates/order.html’
   }
})
```

In the above directive order is the custom directive and ‘E’ represents element. It defines how the directive should be used. There are four different types of custom directives.

- E-element
- C- Class
- A-Attribute
- M-Comment

Using E (**Element)** as restriction, we can create our own element using the angular js directive. Example:

```
app.directive("order", function() {
   return {
      restrict: 'E',
      link: function(scope, element, attrs) {
         scope.item = attrs.item;
      },
      template: '<div>item: {{item}}</div>'
   }
})
```

inside html,

```
<order item="phone"></order>
```

Now it displays the phone in the browser, we can add the styles to the div tag to make it more attractive.

Using C (**Class)** as a restriction, we can create our own element using the angular js directive. Example:

```
<div class="order" ></div>
```

Using A (**Attribute)** as restriction, we can create our own element using the angular js directive. Example:

```
app.directive("add", function() {
   return {
```

```
        restrict: ‘A’,
        link: function(scope, element, attrs) {
          var price = checkAuthorization(attrs.item);
            scope.totalCost=price*attrs.items;
          }
        }
      }
})
```

Using M (**Comment)** as restriction, we can create our own element using the angular js directive. Example:

```
app.directive(“order”, function() {
   return {
      restrict: ‘M’,
      },
      template: ‘<div>item: {{item}}</div>‘
   }
})
```

We need to add the following content in html file,

<!—directiveorder -->

The Directive Scopes main advantage is creating a reusable component and some extra attributes. By using them, we can implement two-way data binding.

“@” Scope: This sort of extension is utilized for passing the value to the directive scope. Suppose that you need to make a gadget for a notice message.

```
app.controller(“myCtrl”, function() {
   $scope.itemName = “Phone”;
})
app.directive(“cart”, function() {
   return {
      restrict: ‘E’,
      scope: {
         message: ‘@’
      },
      template: ‘<div class=“alert”>{{itemName}}</div>‘
   }
});
```

And in the html page, we can use the directive as:

```
<cart itemName=“{{itemName}}” ></cart.
```

“=“ Scope: Here the variable value is directly passed inside the directive. This helps in two-way binding of the data between the controller and view.

```
.directive(“cart”, function() {
   return {
      restrict: ‘E’,
      scope: {
```

```
            item: '='
        },
        template: '<input type="text" ng-model="text"/>'
    }
})
<book-comment item="phone"></book-comment>
```

Dependency Injection: Dependency Injection (DI) is a product plan design that arranges how segments get hold of their dependencies. The Angular injector subsystem is responsible for making parts, determining their conditions, and giving them to different segments as asked for them. DI is unavoidable all throughout Angular. You can utilize it when characterizing parts or when giving a run and config pieces to a module. Components, for example, administrations, orders, channels, and liveliness are described by an injectable industrial facility technique or constructor work. These segments can be infused with "management" and "esteem" parts as dependencies. Controllers are characterized by a builder capacity, which can be introduced with any of the "administration" and "respect" segments as conditions. However, they can likewise be furnished with unique conditions. See Controllers underneath for a rundown of these extraordinary dependencies. The run strategy acknowledges a capacity, which can be infused with "administration," "esteem" and "consistent" segments as conditions. Take note of the fact that you cannot introduce "suppliers" into run squares. The config strategy acknowledges a capacity, which can be infused with "supplier" and "consistent" segments as conditions. Also, take note of the fact that you cannot inspire "administration" or "esteem" segments into the setup.

Forms: Form and controls give approval administrations so that the client can be advised of invalid contribution before presenting a frame. This provides a superior customer encounter than server-side approval alone because the customer gets moment input on the most proficient method to amend any error. Remember that while the customer side support assumes an essential part in giving an excellent client encounter, it can without much of a stretch be evaded and subsequently cannot be trusted. Server-side support is still vital for a safe application.

Example:

<form> <input type="checkbox" ng-model="val">

<input type="radio" ng-model="radioVal" value="dogs">

<input type="radio" ng-model="radioVal2" value="dogs"></form>

Summary

Thus, using all the advanced technologies available in industry to develop the banking application will improve the performance of the application. As being the Single Page Application, it does not take much time to render the web components and as well as it processes the request in the client side and it just contacts the server to retrieve the data.

Chapter III: Methodology

Introduction

As this paper mainly deals with an Online Banking Application, the current problems faced by the users should be addressed. To do that we should gather the information or conduct surveys this is qualitative study and at the same the new proposed should show the better performance this can be quantitative study. So, the following mixed methodology is used to address the issues discussed.

Design of the Study

The mixed methodology helps in finding both the problems with the current system and also helps in finding what the issues are and how the proposed system will solve the existing problem.

Testing the Hypothesis

Create two applications one with Angular JS, Cassandra and using RESTful Web services and using Spring Jdbc for connecting to database. And, second, another application with a relational database like My Sql and JSP (Java Server Pages).

We test both the applications using Http Watch. Http Watch is a browser plugin, which calculates the time, by it we can know how much time the page is taking to load a single page. We can then compare the time of both of the applications and compare them.

Tools and Techniques

1. Eclipse (STS)
2. Chrome Debugger tool/FireBug

Hardware and Software Environment

Hardware requirements:

- Intel Pentium 4 or above
- RAM 4 GB
- 20 GB Hard Disk

Software requirements:

- Windows/linux/mac OS
- Cassandra
- JDK 1.6 or above
- Angular JS
- Tomcat WebServer
- My Sql database
- JAX-RS
- Springs
- Maven
- Tomcat Server
- Html, CSS

Advantages of Approach

The advantages of the proposed approach are mainly as follow:

- The data availability
- Zero fault tolerance

- High Performance
- High Security
- Data or the architecture can be updated any time without effecting the User Interface
- Highly modular

Project Explanation

To start the application, we need to start the UI, Midtier, and the database. First, we will need to start the database.

To start, on the database go to the installation directory of Cassandra and type *cqlsh*. After starting the database, it creates a database and adds the tables shown in Figure 10.

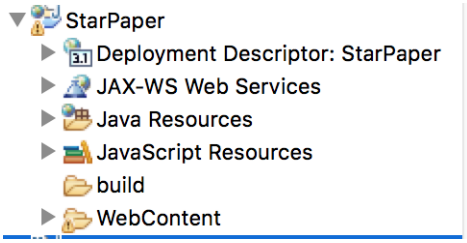

Figure 10. Folder Structure

Here we need the Spring Tool Suite (STS) and Eclipse. STS is a flavor of Eclipse. Here we are using two instances because we need one instance for the User Interface (UI) and another for the Midtier, which is used to connect to the database.

First, we need to setup the UI, to set it up open the STS and create a new dynamic web project with any appropriate name. After that, we see the project in the Project Explorer with the following folder structure.

Expand the Web Content folder; we will find the WEB-INF folder and an index.JSP file. If you open the WEB-INF file, we will find one web.xml. This file is the deployment descriptor

file. When a request comes to this web application first, it reaches web.xml file and searches for the requested web page and maps to the appropriate page and processes the request and sends it back the response. When we start this web application, it automatically starts the index.JSP file as it is a default landing page for the application, this need not be in every application, we can change the default landing web page according to the requirement, but here we are using the default landing index.JSP file.

As we are using this application for UI, which is developed using Angular JS, first we need to add the Angular JS libraries. Following is the list of libraries we need to download for this application:

- angular.JS
- angular-resource.js
- angular-cookies.js
- ui-bootstrap-tpls-1.3.2.min.js

All Angular JS libraries are 1.4.7 version.

To import the libraries, we need to use the following code:

<script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>

It directly downloads the libraries from the Internet. We can also download the libraries manually and specify the location of the file. Either way, it works the same. After importing all the libraries to make the angular content work, we need to add the angular tags to the DOM elements. First, we begin with ng-app that is the angular directive, which initiates the angular framework. It is usually placed in either HTML tag or inside the body tag, for instance: <html ng-app="rootApp">.

After this is done, we need to add the controller—a controller is a function which controls the data using a $scope. We add this in the similar way as we have added ng-app, but this controller should be after ng-app; for example:

<html ng-app="rootApp">

<body ng-controller="rootController">

</body>

</html>

Now we navigate to the login page that has the login form and design using Bootstrap and CSS. After entering the allowed credentials in the login form, we need to click on the Submit button.

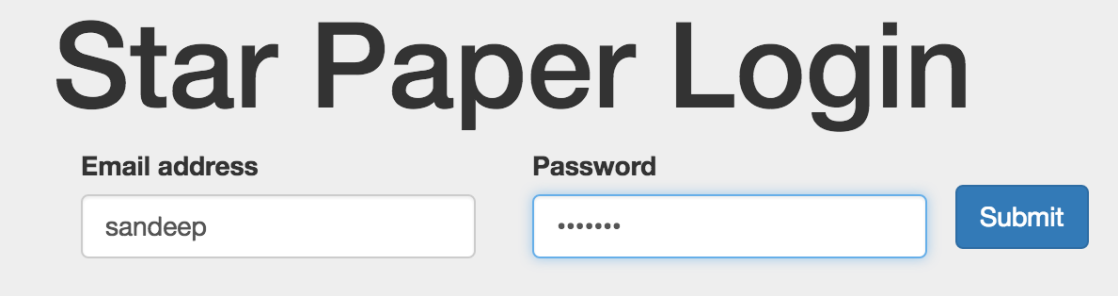

Figure 11. Application Login Screen

The request goes to the controller and from the controller it makes a REST API call. This REST API tries to match the entered data with the database to authenticate the request. If they match, it sends the approved response back to the controller, and the controller navigates to the home screen. However, if the credentials do not match with any of the database records, it sends the rejected response and the controller asks the user to enter the valid credentials. That is the big picture of the login process.

To explain in detail, when the user enters the credentials and submits the form, it reaches the controller (starModuleController.js). There is a click function associated with the Submit button which is triggered when it is clicked. Even before this click function is triggered, there is a layer of security called validation to check the valid input. If it passes the validation, then the request comes to the click function. Here, in the Click function, we use $http service. This service is used to request the data from the remote servers or make REST API calls. As we are seeking for information, we are making a GET call to the REST API. The following is the syntax to make the REST API call:

```
$http.get('http://localhost:8080/SpringRest2/hello', {
                headers: {'SM_USER':id}}
            ).then(function(response){
                $cookieStore.put('user', user.email);
                $cookieStore.put('token', id);
                $window.location="home.html";
                });
```

In the above function, the REST API location URL is provided, If the function gets a successful response it sets the cookies with the user email address and id and navigates to the home page. We can also see the headers attached to the request. These headers are used by the Spring Security to authenticate the request. In this application, we used site minder approach to authenticate the request.

I used Spring Security to authenticate each and every request that comes from the UI to the REST service. As the REST service is also built on the dynamic web application project, it

also has the web.xml file, where all the request processing starts. Inside this application, we write Spring Security FILTERS. These filters help us to authenticate the request and forward the request to the REST API. We need to Specify these filters in the web.XML file. When the request comes to the application, these filters first respond to the request. They first check for the headers in the request and, then, if it does not have any headers, it rejects the request and sends back a *500 Internal error* message to the user. When the request enters the filter, it forwards it to the CustomUserService.java class which has loadUserByUsername function and accepts the header value and validates it with the database and pulls all the roles associated with that header value and stores the data inside UserDetails object and passes it to SecurityContext. This SecurityContext is unique for each request; that means it creates a new thread with a new id for each request, and it will be active until the request is processed and the response is sent back to the client. After the client receives the response thread, the Java Virtual Machine (JVM) will kill it. If the response successful, we add the user id and a token to the cookies and navigate to the home page. These cookies are stored in the browser. If the response is a failure, it remains on the login page until the user enters the valid credentials. After login into the user account, the application uses the cookie data for the authentication. It adds these cookie values to headers of the REST API call and the Spring Security at the REST services checks for the headers and validates the headers and processes the request. Then open the Chrome browser and open the debugger tool. After that, enter into the home page, we can then find the menu on the left, click on the account info and you can see the REST API call and the response.

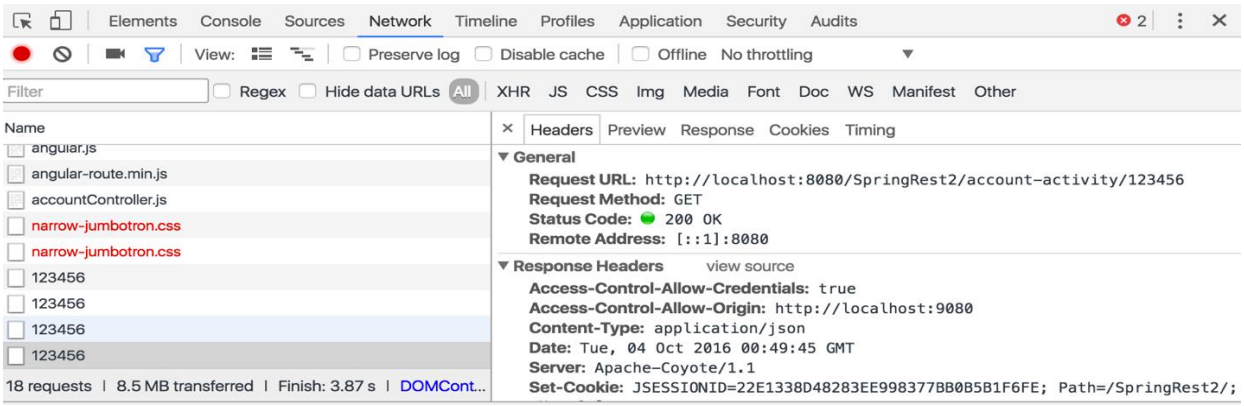

Figure 12. The Network Tab of Debugger Tool

The other application is built completely using Spring MVC and MySQl database. This Application is built using Spring MVC framework. The database is a MySql relational database, which is the conventional form of database. To create a Spring MVC application we need to create a maven web app. It generates a file POM.xml along with other folders. Inside the pom, we have to specify the dependencies for the Spring MVC. If we add dependencies inside pom.Xml, Maven automatically downloads the required Java archives (jar) and adds to the project, these files have various class files which are required. This reduces the programmer's effort to download and add the jar files and even makes it easier to move the application from one environment to another environment as Maven handles the required dependencies and the programmer doesn't have to worry about downloading and mapping the jar files to the application. After adding all the required dependencies for the spring application, web.xml should be configured such that it supports Model View Controller (MVC) style for the application. We need to create another XML file called spring dispatcher servlet.xml. This file handles the request made by the user and sends the request to the appropriate controller and

processes the request and sends back the model along with the view. In Figure 13, we can see how the process works.

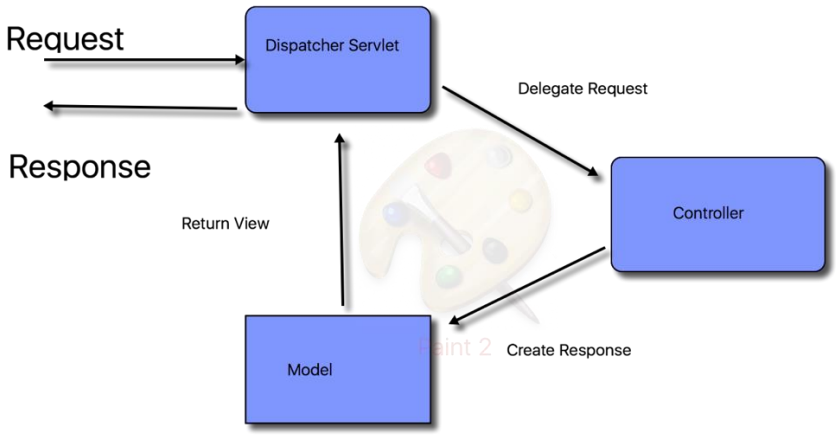

Figure 13. Spring MVC Architecture

When a request is placed, the dispatcher servlet handles the request and passes it to the internal view resolver class. There it selects the controller depending upon the endpoint of the request, it then passes that to the control and processes the request and creates a view along with the data attached to it. It then passes it to the dispatcher servlet and the output will be passed to the browser to display the view.

Results

Following is the application built using the REST service, Cassandra database and Angular JS.

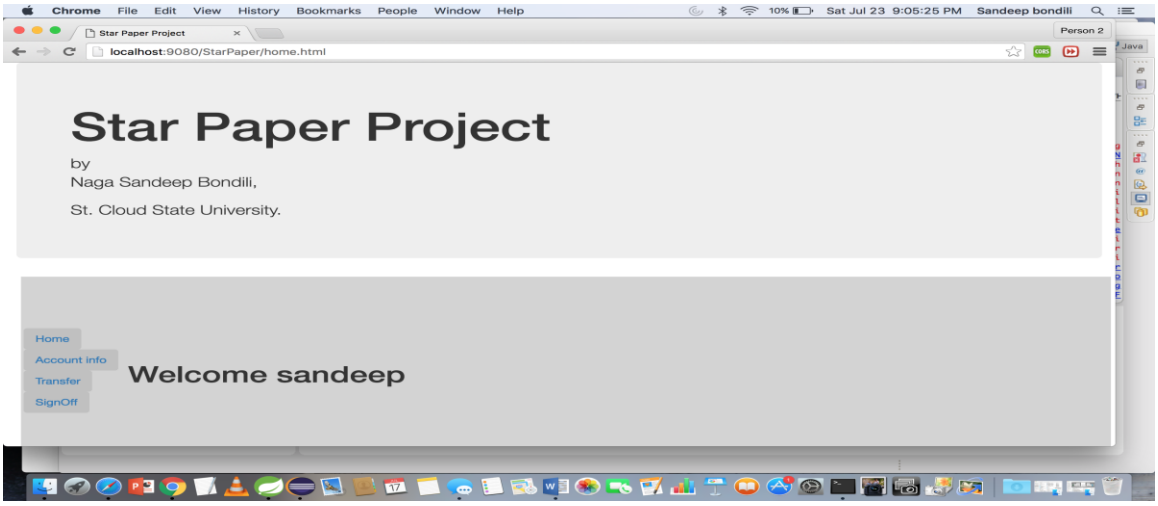

Figure 14. Star Paper Application Home Screen

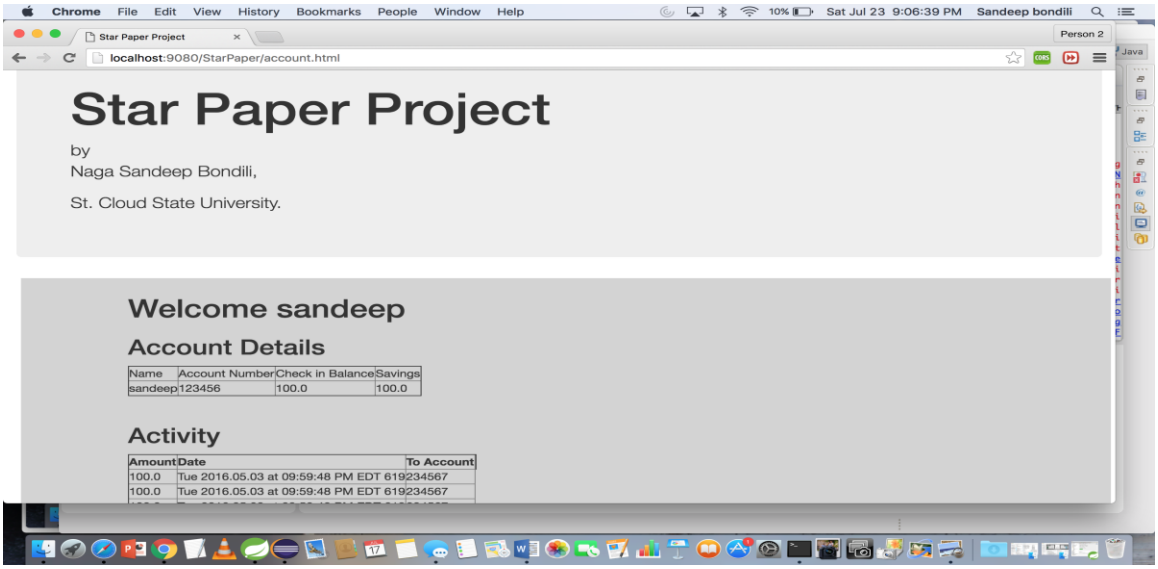

Figure 15. Star Paper Application Account Tab Screen

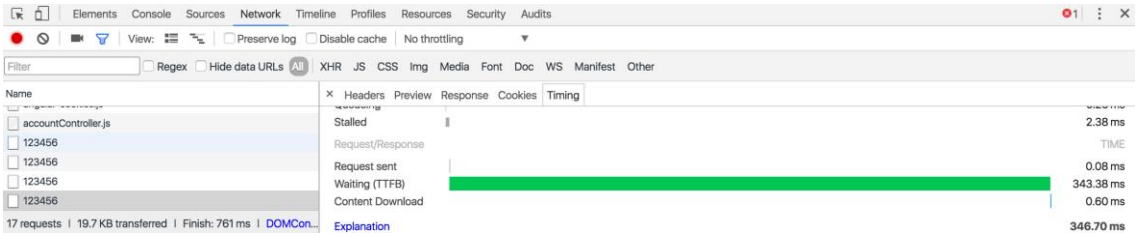

Figure 16. Network Tab in Debugger Tool to Display Elapsed Time

Here, in the above application, I am pulling 100 records. The application uses Spring JDBC to connect to a Cassandra database, REST Service is used to format the data and Angular JS to display the data.

In Figure 16 we can see the green colored bar which represents the time taken to display the data, which is 343.20 milliseconds.

Below is the application built using the Spring MVC and My Sql database.

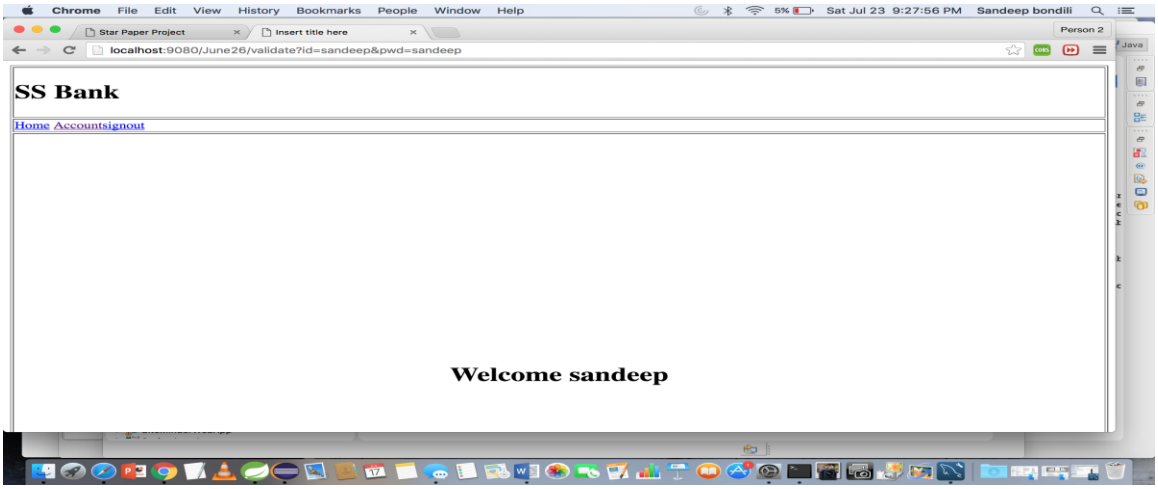

Figure 17. Home Screen of Application Developed in Spring MVC

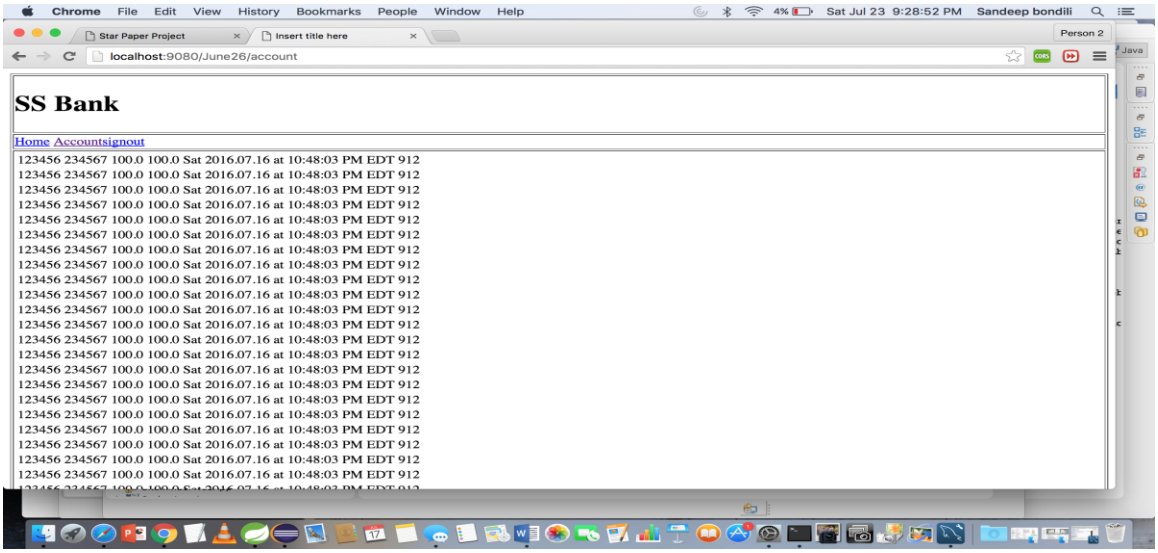

Figure 18. Spring MVC Application Accounts Tab Screen

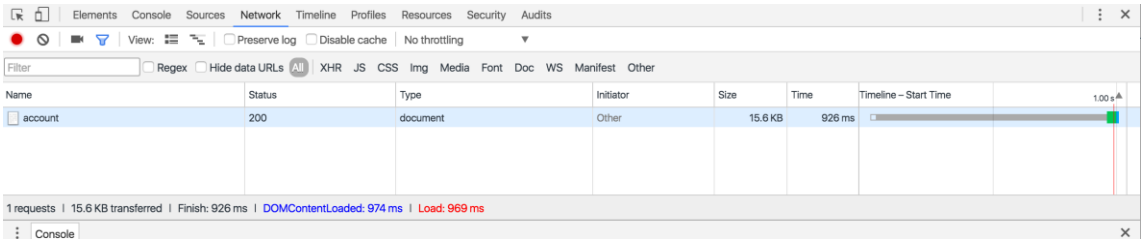

Figure 19. Spring MVC Application Accounts Tab Elapsed Time

From viewing Table 1 we can understand that the application developed using the new technologies retrieved the records faster than the application using the conventional technologies. In the first attempt, with 100 records, there is not much of a considerable difference between the elapsed time. But, while increasing the number of the records, in the third attempt with considerably high volume or records, we see a noticeable amount of difference in time. Hence, we can say that it is always a better choice to opt for new technologies for not just banking and finance, but also for every other industry that deals which huge amounts of sensitive data. But

there is a problem in opting for new technologies. There is always a budget constraint and also the technologies need to get updated regularly. So, it is necessary to choose the right technology depending upon the requirements. As the technologies get updated, choosing a technology to develop an application involves much research and vision for the application. After implementing the application, it should be useful for at least a few years as developing a new application whenever a new and better technology comes to the market will require a large amount of work.

Table 1

Results

	Number of Records	**Elapsed Time for Application with New Technologies**	**Elapsed Time for Application with Conventional Technologies**
First Attempt	100 records	343.38 milli seconds	926 milli Seconds
Second Attempt	10000 records	1018.12 milli seconds	1548 milli seconds
Third Attempt	70000 records	3.48 seconds	6.5 seconds

Chapter IV: Design and Installations

Java Installation (for mac osX)

You can download the jdk 8 from the oracle website download page and double click on it. Follow the instructions from the installation window and install the jdk into the machine. The Next step is to set the path variable and export it to the hardrive.

```
Sandeeps-MacBook-Pro:/ Deepu$ PATH=/Library/Java/JavaVirtualMachines/jdk1.8.0_60
.jdk/Contents/Home/:$PATH
```

Figure 20. Test the JDK Class Path

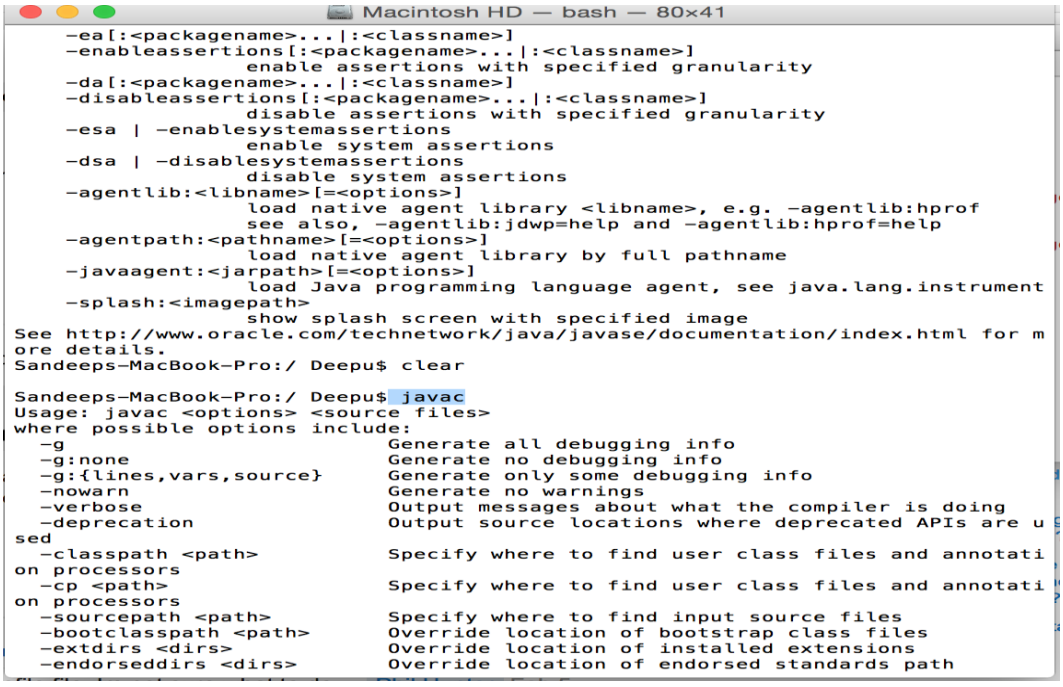

Figure 21. Testing Java from the Console

Installing Cassandra

Download Apache Cassandra from http://cassandra.apache.org. After downloading it, extract Cassandra into a folder of your choice. Then create two folders as shown in the screen shot in Figure 22.

Figure 22. Apache Cassandra in Local File System

After extracting, we need to create a lib folder and log folder and give permissions to those two folders (Figure 23).

```
mkdir /var/lib/cassandra
mkdir /var/log/cassandra

chmod 777 /var/lib/cassandra
chmod 777 /var/log/cassandra
```

Figure 23. Create Cassandra Log Files

Assign the permissions as shown above and go to the bin folder and start the database by typing./cqlsh. It will let you into the Cassandra database.

Chapter V: Conclusion and Future Work

By using the new technologies, we can improve the performance of the application. Angular JS, REST Services, and the Cassandra database help to perform the application faster and also keep the application modular. The technologies that are discussed are quick enough for the time being. There is a new version of Angular JS released in September 2016, Angular JS 2.0. It helps in component based developing which supports reusability in java script. So, it is always helpful to keep the applications up to date and safe from bugs and malware attacks.

References

Angular JS. (n.d.-a). *Angular JS API Docs*. Retrieved from https://docs.Angular JS.org/api

Angular JS. (n.d.-b). *Data binding*. Retrieved from https://docs.Angular JS.org/guide/databinding

Chang, F., Dean, J., Ghemawat, S. Hsieh, W. C., Wallach, D. A. . . . . Gruber, R. E. (2008). *Bigtable: A distributed storage system for structured data*. Retrieved from http://dl.acm.org/citation.cfm?id=1365816

DataStax Academy. (n.d.). *Distribution of apache Cassandra*. Retrieved from https://academy.datastax.com/?utm_source=google&utm_medium=cpc&utm_term=training&utm_content=experts&utm_campaign=DSA%20(US%20%26%20Canada)&gclid=COGEvPeiw9ICFUm2wAodgzQCEg 1

DeCandia, G., Hastorun, D., Jampani, Madan, Kakulapati, G., Lakshman, A., . . . Vogels, W. (2007). *Dynamo: Amazon's highly available key-value store*. Retrieved from http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

Hakka Labs. (2014). *The chubby lock service for loosely-coupled distributed systems* [Videofile]. Retrieved from https://www.youtube.com/watch?v=PqItueBaiRg

Java Virtual Machine. (n.d.). *Wikipedia.* Retrieved from https://en.wikipedia.org/wiki/Java_virtual_machine

Jersey.java.net. (n.d.). *Jersey 2.25-1 user guide*. Retrieved from https://jersey.java.net/documentation/latest/user-guide.html

Lakshman, A. (2008). *Cassandra: A structured storage system on a P2P network*. Retrieved from https://www.facebook.com/notes/facebook-engineering/cassandra-a-structured-storage-system-on-a-p2p-network/24413138919

Park, K., Nguyen, M. C., & Won, H. (2015). *Web-based collaborative big data analytics on big data as a service platform*. South Korea: Big Data SW Research Department, Electronics and Telecommunication Research Institute.

Ramanathan, S., Goel, S., & Alagumalai, S. (2011). Comparison of cloud database: Amazon's SimpleDB and Google's bigtable. *International Journal of Computer Science Issues, 8(2), 243-246.*