****

**CSC 258**

**Fall 2014**

**A Project Report on**

**Android Whiteboard Application**

**Instructor Name:**

**Dr. Jinsong Ouyang**

**Submitted by:**

**Neshant Somanath, Rohit Surve**

Abstract:

The objective of the project was to Develop a Mobile Application based using Android Mobile application development platform and SPRING Framework.

The Primary features include Drawing on the Whiteboard supported by saving the Whiteboard on the Client and the Server side. Design of the Whiteboard enables the user to draw images as well as type text on the same whiteboard. The business logic is implemented using Services developed using the MVC SPRING Framework which comprises of the server side code implementation.

The application supports the following features:

* Drawing on the whiteboard
* Support for text on the whiteboard
* Saving the whiteboard Locally and on Server
* Creating a User Login for the whiteboard

Table of Contents

[High Level Architecture: 4](#_Toc388218882)

[The Points Cloud Algorithm 5](#_Toc388218883)

[Database Schema 9](#_Toc388218884)

[Android Server: 10](#_Toc388218885)

[Android Client: 18](#_Toc388218886)

[Future Enhancements and Current Standing of the Project 21](#_Toc388218887)

[References 22](#_Toc388218888)

# High Level Architecture:



The High Level Architecture Diagram consists of the Following three facets.

1. Client: The client Application is installed on Android Phones. Registered Users can Login into the application to gain access to the whiteboard. The Client application also provides an option to save the whiteboard locally on the client device or on the Sever.
2. Server: Business Logic for communicating with the Android client application is done using SPRNG services deployed on application server.

The implementation of these services provide user authentication, saving of the whiteboard as an image onto the server.

1. Database: The Database schema is designed to store the user information.

# The Points Cloud Algorithm

Real time liveliness is very difficult to realize, there are numerous algorithms that attempt to do this with little success. Through this project we propose a simple algorithm that is designed to enable the sharing of points in real time that leverages the use of clouds. The core concept of this is to upload the points dynamically onto to the cloud. The cloud enables real time synchronization of the various elements.

A new client first registers with the server and obtains a registration ID which is unique for all clients that are registered on the server. Once receiving a registration ID, it listens for any action on the screen. Whenever any action is sensed it records the action and sends it to the cloud. It assumes a packet format as described

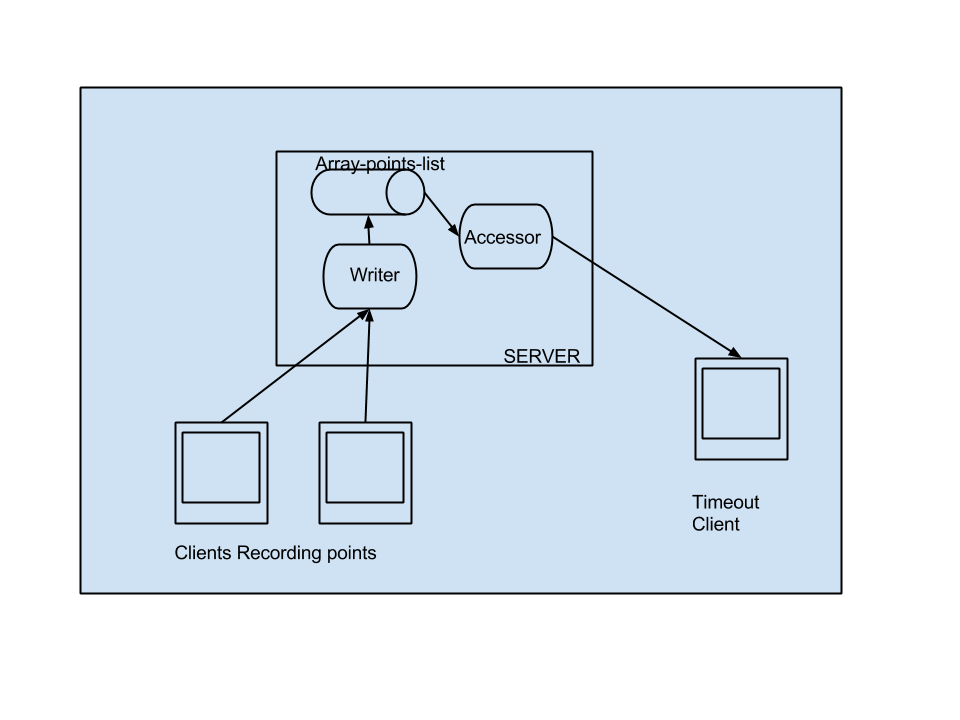
{Client ID, List Of Points}

The cloud ` each packet that arrived by its client id

Each Client has a thread-timer that is set, when the timer runs out it requests for the latest cloud-image (the list of point on the cloud at that instance) that is sent to it. The image contains an array of point-lists that can be associated to their origin system form the client id that has been stamped along.

A client will iterate through the point-lists and exclude all point-list that have a client-id matching that of its own (as it is the origin of that action). Any other point-list is rendered on to the screen. Since the timer is or a relative small interval the refreshing occurs relatively fast and the resultant screen seems to update in real time.

The receiving client will update its local whiteboard by adding the new points to it. Once updated the points will remain on the local screen, they will not be erased. Conceptually they have been unioned with the existing points on to the whiteboard.



The above diagram shows a real time use, there are many problems associated with distributed systems. This algorithm deals with the some of theses issues in the following way:

1. Write Consistencies:

This is achieved as there is no requirement that the submissions are received in total order. The clients can asynchronously submit the lists and the server will record them. When a client requests for the cloud-image, it will perform an exclusion operation on the cloud-image data(i.e it will ignore all the point-lists that have been stamped by its own ID) and render the rest.

All the point list are local occurrences that are sent to the server by the clients.

The systems maintains it’s consistencies by considering the fact that all the points need to be rendered.

2. Read Consistencies:

Since the there are multiple reads at the same time as there can be writes, The nature of the system is very free and flexible and thus it can easily be adapted to overcome these inconsistencies.

Further as the refresh rate of the screen if relatively high it introduce a concept called eventual consistency. This means that the screen will be eventually consistent with each other.

**ALGORITHM**

//On touch listener

when finger is down

    record the points

when finger is lifted

    send the points to the cloud

//Renderer

when timeout

    request for the latest cloud-image

    iterate through the cloud-image

if Client-ID matches local Client-ID

exclude

else

update

    reset timer

**ALGORITHM SERVER SIDE**

    //Upon request to update

    Increment the cloud\_limit

    add the new list at the cloud limit

    set true flag if success and notify client

//Request

Send the latest possible points-image

The core of the algorithm is based on having a small cloud size, we implement this by using a max size on the cloud. This loops around so that the size of the cloud is maintainable.

# Database Schema

The database is created in the MySql Database. The Database for the application Android Whiteboard Application is mainly used for Authentication purpose. It can further be extended to store the Entire White board into the database as a Binary Large Object which can be accessed by other users.



The Database Schema mainly consist of two tables namely:

1. Tab: This table contains the basic User information like the User name and Password. The information from this table is primarily used for authenticating the user in the Android application.
2. PicLab: This table mainly contains the White board image that can be stored for a particular user. It primarily contains 2 attributes – Name and Pic in which it associates every username with a Picture which is the White board stored as a binary large object. The Name and the Pic columns together form a Unique key.

# Android Server:



1. com.mkyoung.common.controller: This package comprises of the controller of the implemented MVC pattern architecture. It has the following class.
   1. JSONController.java

This class contains the following primary Interfaces.

|  |  |
| --- | --- |
| Interface | Description |
| **public** @ResponseBody PointsCloud getShopInJSON() | This function is used to test if there is successful mapping going on between JSON jaxon mapper and the object. |
| **public** @ResponseBody registrationNumber getRegistrationNum() | This function is used to register the client with and issue it with a registration number. |
| **public** Description postShopInJSON(@RequestBody allPassword psaa) | This function is used to assist in remote client login abilities |
| **public** Description postAddMemenerInJSON(@RequestBody allPassword psaa) | End-point to add user name and password into the DB |
| **public** Description thePOSTfunction(@RequestBody Shop stats) | This function is largely used for testing purposes |
| **public** UserStats create() | This function is largely used for testing the pojos. |
| **public** Description writeImg(@RequestBody FileManager stats) | The WriteImg function accepts a base64 image as one of the augments and the name in another, It then decodes the file and stores according to the argument |

//ENDPOINT: 1

//

/\*\*

\* <Definition>This function is used to test if there is successful mapping going on between JSON jaxon mapper and the

\* object.</Definition>

\* <Returntype>Just any type of POJO that has been mapped </Returntype>

\* **@return**

\*/

@RequestMapping(method = RequestMethod.*GET*)

**public** @ResponseBody PointsCloud getShopInJSON()

{

**return** pc;

}

//ENDPOINT: 2

/\*\*

\* <Description>This function is used to register the client with and issue it with a registration number.

<Description>

\*/

@RequestMapping(value = "/registerMe",method = RequestMethod.*GET*)

**public** @ResponseBody registrationNumber getRegistrationNum()

{

*Count* +=1;

registrationNumber rn = **new** registrationNumber();

rn.setRegistrationNumber(*Count*);

**return** rn;

}

//ENDPOINT: 3

/\*\*

\*

\* **@param** psaa

\* **@return**

\* <Decription>This function is used to assist in remote client login abilities</Description>

\*/

@RequestMapping(value = "/pass", method = RequestMethod.*POST*)

@ResponseBody

**public** Description postShopInJSON(@RequestBody allPassword psaa) {

System.*out*.println("Pass received");

//Create the description object that returns the value

Description obj = **new** Description();

//call BD

dbcon\_select DBobj = **new** dbcon\_select();

String VPassword = DBobj.getdata(psaa.getUserName());

String TPassword = psaa.getPassword();

**if**(TPassword.equals(VPassword))

obj.setDescription(**true**);

**else**

obj.setDescription(**false**);

**return** obj;

}

//ENDPOINT: 4

/\*\*

\* <Description>End-point to add user name and password into the DB</Description>

\* <Exposes>The allPassword Pojo</Exposes>

\* <URL></URL>

\*/

@RequestMapping(value = "/addMember", method = RequestMethod.*POST*)

@ResponseBody

**public** Description postAddMemenerInJSON(@RequestBody allPassword psaa) {

System.*out*.println("Pass received postAddMemberInHson");

//Create the description object that returns the value

Description obj = **new** Description();

//call BD

dbcon\_select DBobj = **new** dbcon\_select();

Boolean Valid = DBobj.inserData(psaa.getUserName(), psaa.getPassword());

obj.setDescription(Valid);

**return** obj;

}

//ENDPOINT: 5

/\*\*

\*

\* **@return**

\* <description>This function is largely used for testing purposes</Description>

\* <URL></URL>

\*/

@ResponseBody

@RequestMapping(value = "/description", method = RequestMethod.*POST*)

**public** Description thePOSTfunction(@RequestBody Shop stats){

Description obj = **new** Description();

obj.setDescription(**true**);

//<Information>logic to add the populate the cloud along with the garbage collector</Information>

//Garbage collector limit-> Cloud limit

**if**(cloud\_limit>100)

{cloud\_limit=0;}

//Populate the cloud

pc.arrList.add(cloud\_limit,stats);

//Increase the upper limit

cloud\_limit++;

Iterator it = stats.arrList.iterator();

System.*out*.println("----the new list----From Client No:" +stats.getClientID());

**while**(it.hasNext())

{

Point p = (Point) it.next();

System.*out*.println("Points "+p.getX()+" "+p.getY());

}

//sta.arrList.add(stats);

**return** obj;

}

/\*\*

\*<Description>The function to add the submitted points to the static array</Description>

\*/

**public** **void** add\_the\_points(Shop temp)

{

}

//ENDPOINT: 6

/\*\*

\*

\* **@return**

\* <description>This function is largely used for testing purposes</Description>

\* <URL></URL>

\*/

@ResponseBody

@RequestMapping(value = "/dec", method = RequestMethod.*GET*)

**public** UserStats create()

{

UserStats stats = **new** UserStats();

File file\_upload = **new** File("C:/Users/Neshant/Desktop/Test-file.txt");

String fileName = file\_upload.getName();

stats.setFirstName(fileName);

**try** {

String fileContent = *convertFileToString*(file\_upload);

stats.setLanstName(fileContent);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** stats;

}

//ENDPOINT: 7

/\*\*

\*

\* **@param** stats

\* **@return** Description

\* <Description>The WriteImg function accepts a base64 image as one of the augments and the name in another,

\* It then decodes the file and stores according to the argument name</Description>

\* <Exposes>The UserStats POJO</Exposes>

\* <URL></URL>

\*/

@ResponseBody

@RequestMapping(value = "/des", method = RequestMethod.*POST*)

**public** Description writeImg(@RequestBody FileManager stats)

{

//<Information>Local variables</Information>

//Create the description object that returns the value

Description obj = **new** Description();

**try**

{

ConvertStoreFile(stats.getFieldName(), stats.getFieldFileString(), stats.getFieldFileName());

}

**catch** (IOException e)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

//<Information>Return false to the client in-case of any error</Information>

obj.setDescription(**false**);

**return** obj;

}

//<Information>Return true in all other cases</Information>

System.*out*.println("Entered the des function "+stats.getFieldName());

obj.setDescription(**true**);

**return** obj;

}

1. com.mkyoung.common.dbcon: This package contains the implementation to interact with the database (MySql). It has the following class.
   1. dbcon\_blob\_inser.java : This class provides methods to connect to the database and perform the CRUD operations to register and new user and fetch user records for authentication.
2. com.mkyoung.common.model: This comprises of the Model. It contains the Pojos which are mapped using the Jackson-Jason Library.
   1. allPassword.java : This class is used for password authentication.
   2. UserStats.java : It is used for the First name – Last name relation.
   3. Description.java : This class is used to determine whether the Rest services will return either a true or false to denote the status of service response.
   4. Point.java : This class is used to capture the co-ordinates of the White board.
   5. RegistrationNumber.java : This class is used by the client to register itself with the server.
   6. FileManager.java : This file is used to send the Whiteboard images from the Android client to the Server.

# Android Client:



1. com.example.testapp
   1. SignUp.java: This class contains the Sign up Logic for a new client user. It contacts, the server and requests for authentication upon successful authentication it changes the state of the main activity page. This change in state is a transition from login to logged-in. the FSM is shown:

LOGIN SUCCESSFUL

Login logged in

Post login the user can save data on to the server and access files that are devoted to the server. In case of new users the login page provides a separate signup page. The login page is designed to not be on the android stack so that after signup is done the user must login again.

* 1. LogIn.java : This class contains the execution module to create a new user and perform user authentication.
  2. MainActivity.java : This class contains the code behind logic to perform user Login and storing the Whiteboard on the server by invoking the Spring Web services. Additionally it helps in submitting it in JSON format on the server to store it as an image.

This class has two states prior and post login. After login we see a logout function. This class also invokes the whiteboard view that is a custom view created on android using the graphics library and the android canvas infrastructure.

* 1. LView.java: This class extends the android view to create a custom view for the Whiteboard. It contains the functionality to type text and draw on the whiteboard. The class is pivotal in the algorithm that has been suggested. At startup it registers with the server and acquires a ID.

This class has login to implement a listener that records points. It also has text support for the whiteboard, this is done using the greedy concept. A snippet is shown:

**if**(Ct.length()<19)

{

canvas.drawText(Ct, viewWidthHalf, viewHeightHalf, circlePaint);

}

**else** **if**(Ct.length()<(19\*2))

{

String s1 = Ct.substring(0, 18-1);

String s2 = Ct.substring(18);

canvas.drawText(s1, viewWidthHalf, viewHeightHalf-120, circlePaint);

canvas.drawText(s2, viewWidthHalf, viewHeightHalf, circlePaint);

}

**else** **if**(Ct.length()<(19\*3))

{

String s1 = Ct.substring(0, 18-1);

String s2 = Ct.substring(18,(19\*2)-1);

String s3 = Ct.substring((19\*2));

canvas.drawText(s1, viewWidthHalf, viewHeightHalf-120, circlePaint);

canvas.drawText(s2, viewWidthHalf, viewHeightHalf, circlePaint);

canvas.drawText(s3, viewWidthHalf, viewHeightHalf+120, circlePaint);

}

**else**

{

**int** base\_cut = 18;//<Information>The default base minimum</Information>

**if**(Ct.length()<(19\*4))

{

circlePaint.setTextSize(40);

base\_cut = 20;

}

**else** **if**(Ct.length()<(19\*5))

{

circlePaint.setTextSize(30);

base\_cut = 22;

}

String s1 = Ct.substring(0, base\_cut-1);

String s2 = Ct.substring(base\_cut,((base\_cut)\*2)-1);

String s3 = Ct.substring((base\_cut)\*2);

canvas.drawText(s1, viewWidthHalf, viewHeightHalf-120, circlePaint);

canvas.drawText(s2, viewWidthHalf, viewHeightHalf, circlePaint);

canvas.drawText(s3, viewWidthHalf, viewHeightHalf+120, circlePaint);

}

The code to call the custom view is

<com.example.testapp.LView

android:id=*"@+id/custView"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"254dp"*

android:layout\_margin=*"5dp"*

custom:circleColor=*"#ff0099"*

custom:circleLabel=*"Hello"*

custom:labelColor=*"#ffff66"* />

1. com.examples.webServices
   1. HttpAsynTaskPoint.java: This class performs the execution the Services Asynchronously on a separate Background Worker thread.

# Future Enhancements and Current Standing of the Project

We have not been able to fully implement the cloud-point algorithm. As a future enhancement we intend to complete this.

Further the algorithm is very scalable and can be easily extended, in case of larger groups we can have multiple clouds that perform actions, it is possible that the clouds could also be function specific.

The image support also remains, as a future enhancement we could also provide image support and under the correct threading conditions also provide a form of group video call facilities.(If the image is linked to the camera)

# References

 Spring and server resources

[Www.mkyong.com](http://Www.mkyong.com)

[Www.tutorialspoint.com](http://Www.tutorialspoint.com)

Android resources

[Www.androidhive.com](http://Www.androidhive.com)

[Www.androiddeveloper.com](http://Www.androiddeveloper.com)

Java and MySql resources

[Www.Oracle.com](http://Www.Oracle.com)

Other references

[Www.stackOverflow.com](http://Www.stackOverflow.com)