Best Cloudlet Switcher (BCS) for Computation Offloading

Cloud-computing Project - 2016

University of Arkansas at Little Rock, USA

By

Uddhav Prasad Gautam (upgautam@ualr.edu)

Laxima Niure Kandel (lxniurekande@ualr.edu)

**Abstract**

*Mobile devices are resource poor whereas cloud is resource rich. Mobile Cloud Computing is the integration of two so that the resource deficient mobile devices can take advantage of infinite resources provided by the cloud. This is achieved by computation offloading. As cloud are geographically located far-way, this could lead to higher latency. Therefore, Cloudlets are implemented at the edge-network for the mobile devices to offload computation and thus decrease latency. Choosing the best cloudlet for computation offloading is one of the challenges faced by mobile devices. To tackle this problem, we propose a mobile application called Best Cloud Switcher (BCS). BCS assumes that the mobile user is stationary when the cloud application is launched. Cloud provider like Amazon can deploy several Amazon cloudlets in different locations and in order for the mobile devices to use these cloudlets (as the user can move from one location to another), user information should be created and centrally stored and authenticated. If user information are stored everywhere in the cloudlet controller, there may happen Co-residency attacks. Once the authentication is done by the authentication server in the cloud, the user automatically gets connected to the nearby cloudlet. Here the server requires the user to turn on the GPS service in his phone so that based on the GPS location it can delegate control to cloudlet controller so that the mobile user application can be referred to  the best cloudlet server for computation offload.*

**Keywords**: Cloudlet, Cloudlet Controller, Mobile Cloud Computing, Co-residency attacks

1. Introduction

Best Cloudlet Switcher (BCS) is an effective automated best cloudlet server referrer tool for any mobile devices to perform computation offloading. After mobile devices (users) get authenticated, their gateway gets changed to Cloudlet Controller’s IP. Now, every packets routes to Cloudlet Controller. When any computation comes from BCS app, cloudlet controller takes that as a computation. It checks the best cloudlet server available right now to forward that computation query to that cloudlet server. Cloudlet server computes the result and user finally sees the output in his mobile screen.

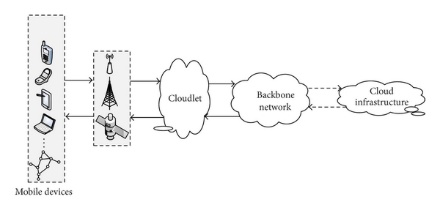
* 1. Mobile Cloud Computing

Fig: Cloudlet Architecture [1]

Mobile devices are portable, hand-held, generally small sized, low computing power devices. User prefers them because they are easy to carry. However, mobile devices are limited in resources like storage and RAM. Due to convenience of use, the user wants to run applications that are compute intensive like face recognition, language translation etc. in his mobile device. This quickly drains the battery of mobile device which is not desirable at all. To avoid this problem, computation can be offloaded to cloud. Cloud has huge computation and storage capability. But, since the cloud is located far away in the core network, offloading to cloud increases latency which is a very crucial parameter for augmented related applications. The intuitive idea is to deploy small clouds nearby the user called Cloudlet so that the application do not face the latency problem. Thus, Cloudlet is mobility-enhanced small datacenter network implemented at the edge-network in order to help mobile devices for the computation. Computation offload results could be obtained faster and thus the application achieve smoother display. Cloudlet devices are not mobile devices. They are computation powerful, always power connected devices located in between mobile devices and cloud. They may be connected to the Cloud through wired network. It should support wide-range of computations, should have minimal restrictions on Operating System, programming languages etc.

One of the challenges faced by the mobile device is how to get connected to best cloudlet. In our project we propose to build an application called Best Cloudlet Switcher (BCS) for an effective automated best cloudlet server connection for any mobile devices for computation offloading. After mobile devices (users) get authenticated, their gateway gets changed to Cloudlet Controller’s IP. Now, every packets routes to Cloudlet Controller. When any computation comes from BCS app, cloudlet controller takes that as a computation. It checks the best cloudlet server available in the cloudlet for that particular time to forward that computation query to that cloudlet server. Cloudlet server performs computation and passes the result to user mobile screen. All these is done seamlessly, the latency to offload and get the result is so low that the user has the feeling as if the computation is done in the mobile device itself.

* 1. BCS working scenario

The first step involved is Authentication. The Cloud user will sign up with the cloud from the Android App. Sign up validation will be checked both at the user side and the server side. As we know, client-side validation for the fast feedback to user if he inputs let’s say email incorrect format, while server-side validation assures whatever inputted is according to server’s standard and not malicious code. Once the user sign ups successfully, his login information will be saved in the Authentication database at the server side. Future sign in will be validated by comparison with the stored information at the database. This database will also contain the cloudlet GPS locations. The cloud company will enter the GPS location when they set up a particular cloudlet in any geographical area. The main purpose of the BCS is to provide the user with the best cloudlet. To fulfill this purpose, the server will require the user’s GPS location. Hence, the BCS requires the user to turn on the GPS location service, if not, in order for them to be able to sign in. The authentication server (MainServer here) receives the log in details along with the user’s GPS location, the details are validated with the information in the database. If the information is not matched, the server will send ‘login failed’ message to the user and the user will try again. The user will be allowed to try certain number of times ex. 10 attempts of trial before his account will be locked. The user then has to go through the account unlocking process. However, if the user details are matched, based on the user GPS location, it will find the best cloudlet (actually the nearest cloudlet) to the user and will provide the cloudlet controller gateway (R1) to the server and then server will then forward the message to the user. Meanwhile, the user is listening for the controller gateway. After it receives the R1, the user will then be connected to the cloudlet controller and now can offload any computations to it. The controller will check for the best server (or VM) which can provide higher service to the user and the user will offload its compute intensive applications to this server. The server will do the appropriate computation and send back the result to the user. This result will be displayed on the mobile screen of the user. It is important to note that we are assuming that the mobile device is stationary for our initial work. Our future work will involve a scenario when the mobile devices are moving.

1. Relevant works

CDN on-demand [2], Tactical Cloudlets [3], A survey of mobile cloud computing [4], Research on Mobile Cloud Computing [5], Mobile cloud computing: A survey [6] are studied thoroughly. CDN on-demand provides about choosing best CDN. We have implemented the same concept for the computation. Tactical Cloudlets is totally VM open stack based implementation while we are not using open stack in BCS. Other above cited papers are just for the exact understanding of Mobile based computing.

1. Methodology

Throughout the development and test, we use UALR’s cloud environment to configure our testing environment.

* 1. Structure of the Project

Here, we will briefly define the standards that we will be implementing throughout of this project. Project structure consists but are not limited to: communication standards, documentation standards, and change control procedure standards etc.

* + 1. Wired/Wireless Communication Standards

Mobile devices like Android smartphones, Android tablets etc. use their standard Wi-Fi adapter to be in the communication with internet access points. While immobile devices like: Desktop PC can use Wi-Fi and/or wired connection to reach into the internet. We have not gone through the details about Router, Firewall, Server’s Hardware configuration, Access point, Wire capacity and standards etc.

* + 1. Documentation Standards

Throughout the development until Beta version, we don’t use any standard documentation format. At the end, we will use IEEE conference format. But from the beginning we use IEEE referencing method for citation.

* + 1. Change and Control Procedure Standards

We use Git source code management system for our versioning of code. Documentations are shared for the review in the Github. While developing we code and test everything in test environment but when we are done we shift everything in the production environment.

As every software building process is based on the architecture it is used.

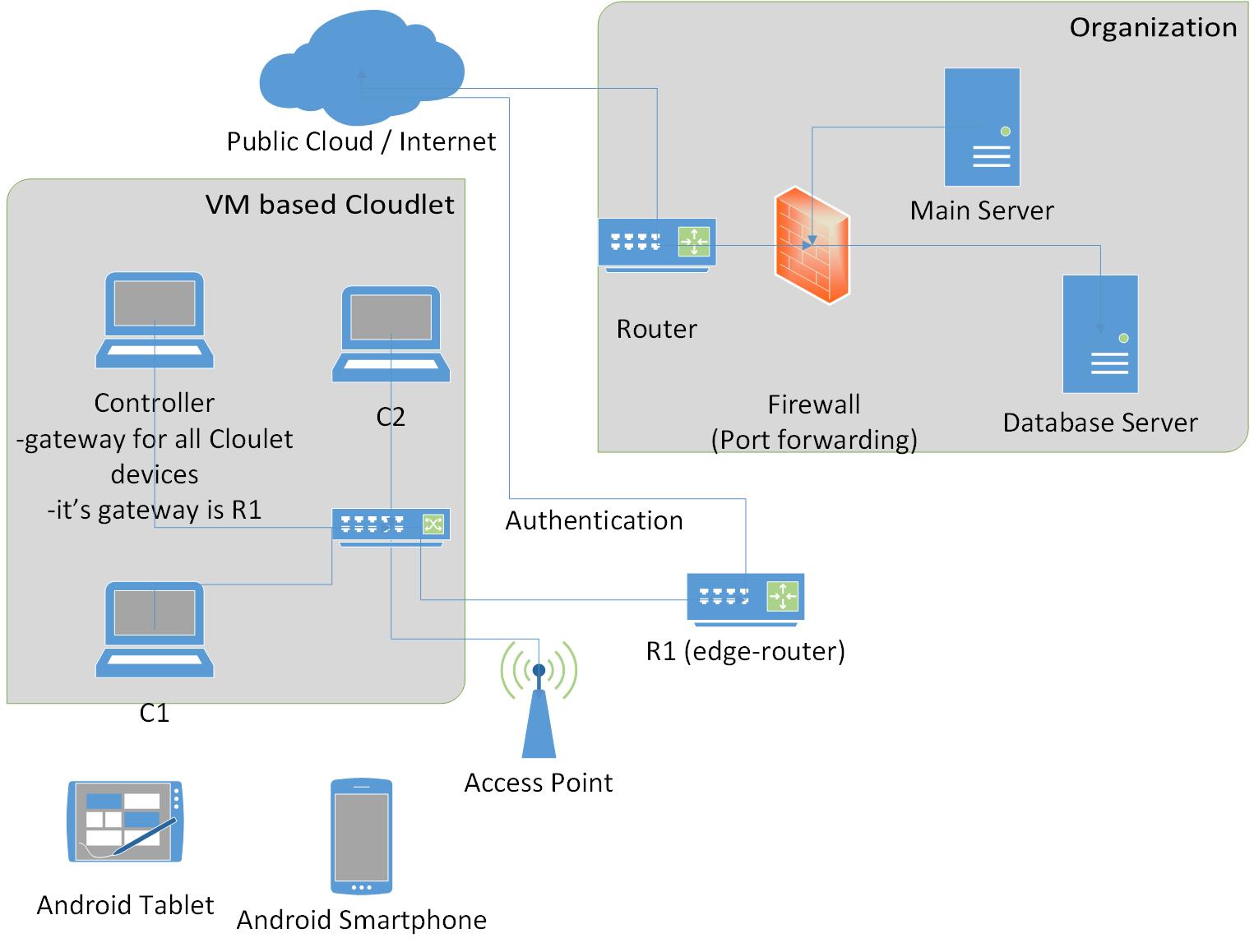


Fig: Network Architecture BCS

* 1. System Requirement Analysis

Based on the Network architecture above, we have defined requirements as below.

|  |
| --- |
| **Test Environment (UALR cloud)** |
| **Database Server**   1. MySQL  * Used to hold the data   **MainServer (or Authentication Server)**   1. Node.js  * Java script is used to authenticate user (employee, manager etc.) and other objects like Cloudlet Controller, Cloudlet Server.   **Developer Machine (Application Server)**   1. Android Studio  * To build BCS Android App.  1. Eclipse to build Node.js applications  * To build Node.js Applications  1. Cordova Mobile Framework  * To access to android native features like GPS.  1. Ionic Advanced Mobile Framework with Phone Gap  * To provide seamless reflection using Angular JS.  1. NPM package  * To install Node.js/Cordova modules  1. GIT  * To control versioning system of coding   **Cloudlet Controller**   1. BCS Cloudlet Server Switcher (Java script tool)  * Used to dynamically select best Cloudlet Server for users.   **Cloudlet Server**   1. Factorial tool to calculate factorial value   **Android Client**   1. BCS Android APP  * Used to offload the computation (Factorial here in BCS) to Cloudlet Server and displaying result in its GUI. |
| **Production Environment** |
| **Database Server**   1. MySQL  * Used to hold the data   **MainServer (or Authentication Server)**   1. Node.js  * Java script is used to authenticate user (employee, manager etc.) and other objects like Cloudlet Controller, Cloudlet Server.   **Cloudlet Controller**   1. BCS Cloudlet Server Switcher (Java script tool)  * Used to dynamically select best Cloudlet Server for users.   **Cloudlet Server**   1. Factorial tool to calculate factorial value   **Android Client**   1. BCS Android APP  * Used to offload the computation (Factorial here in BCS) to Cloudlet Server and displaying result in its GUI. |

Fig: Requirement Specifications BCS

* 1. System Design

As our computation offloading app is Android app. We discuss little about Android Mobile App Architecture.

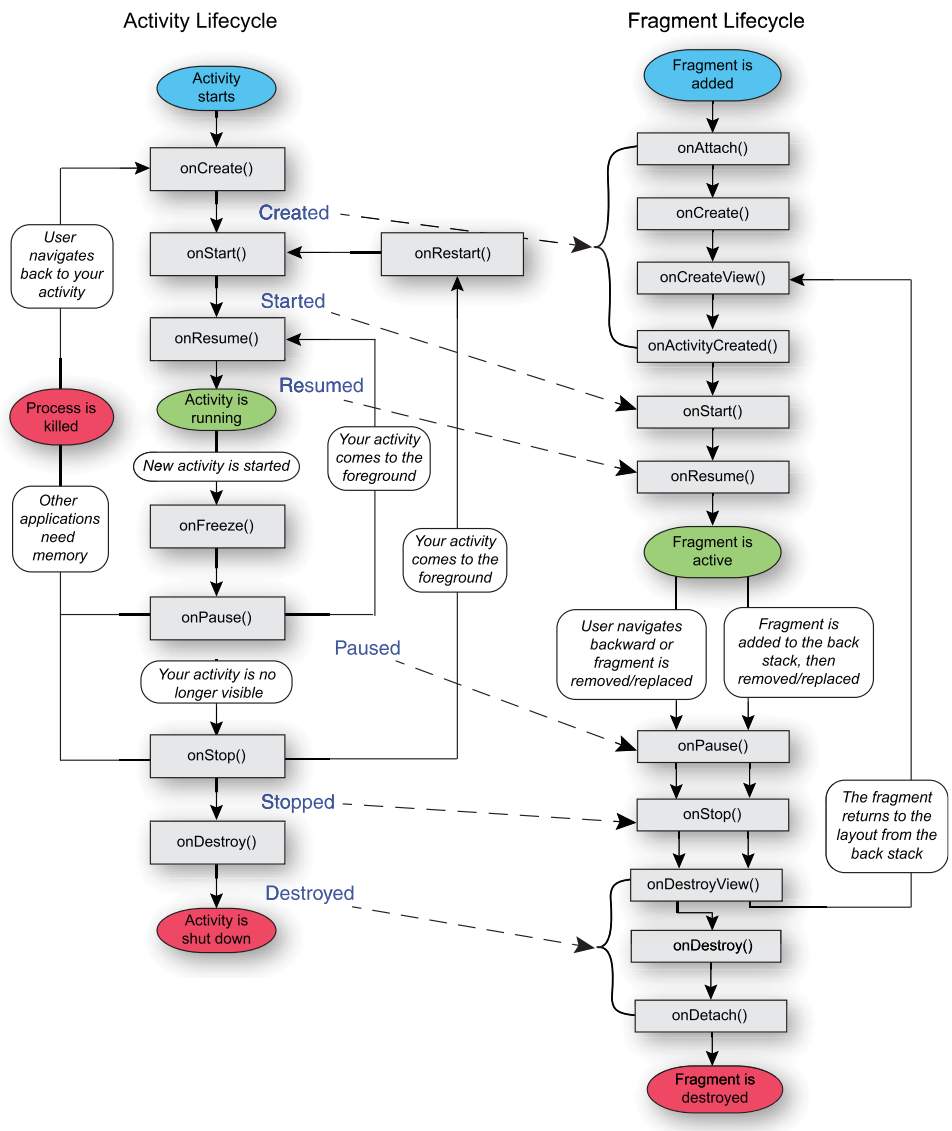


Fig: Android App Lifecycle including Fragment [1]

Every screen in an android application has XML (layout) and Java (Activity). Java file is called as Activity. It is a single screen in an application. The lifecycle of an Activity consists of:

**Don’t exist (Activity is not started)**

**Foreground**

**Background**

**Pause**

Each Activity class has seven methods:

**onCreate():** This call back creates an activity. Here the layout or interface is created for the user.

**onStart():** The Activity1 is visible to the user. Here the user is able to see sign-in and Sign-up page for BCS.

**OnResume():** This is required when the user starts interacting with the application. The BCS user clicks the sign-in or sign-up button. These buttons will be associated with event listeners which will take the user to Activity2 where the user will either sign-in or fill the sign-up page depending on what button the user clicked. Now the Activity2 is onResume and Activity1 is onPause.

**OnPause:** The paused Activity1 does not receive any input from the user and cannot execute any code. It is running but in the background.

**onStop():** This activity is called when the activity is no longer visible.

**onDestroy():** When this called the activity is destroyed by the system.

**onRestart():** This is called when the activity is started after stopping it.

Fragment represents a behavior/ portion of UI within an activity. Multiple fragments can be embedded in an Activity to create a multi-pane UI. A single Fragment can be reused across multiple activities. Fragment lifecycle is tied to and coordinated with the lifecycle of its containing Activity. At the same time, Fragment have their own lifecycle callback.

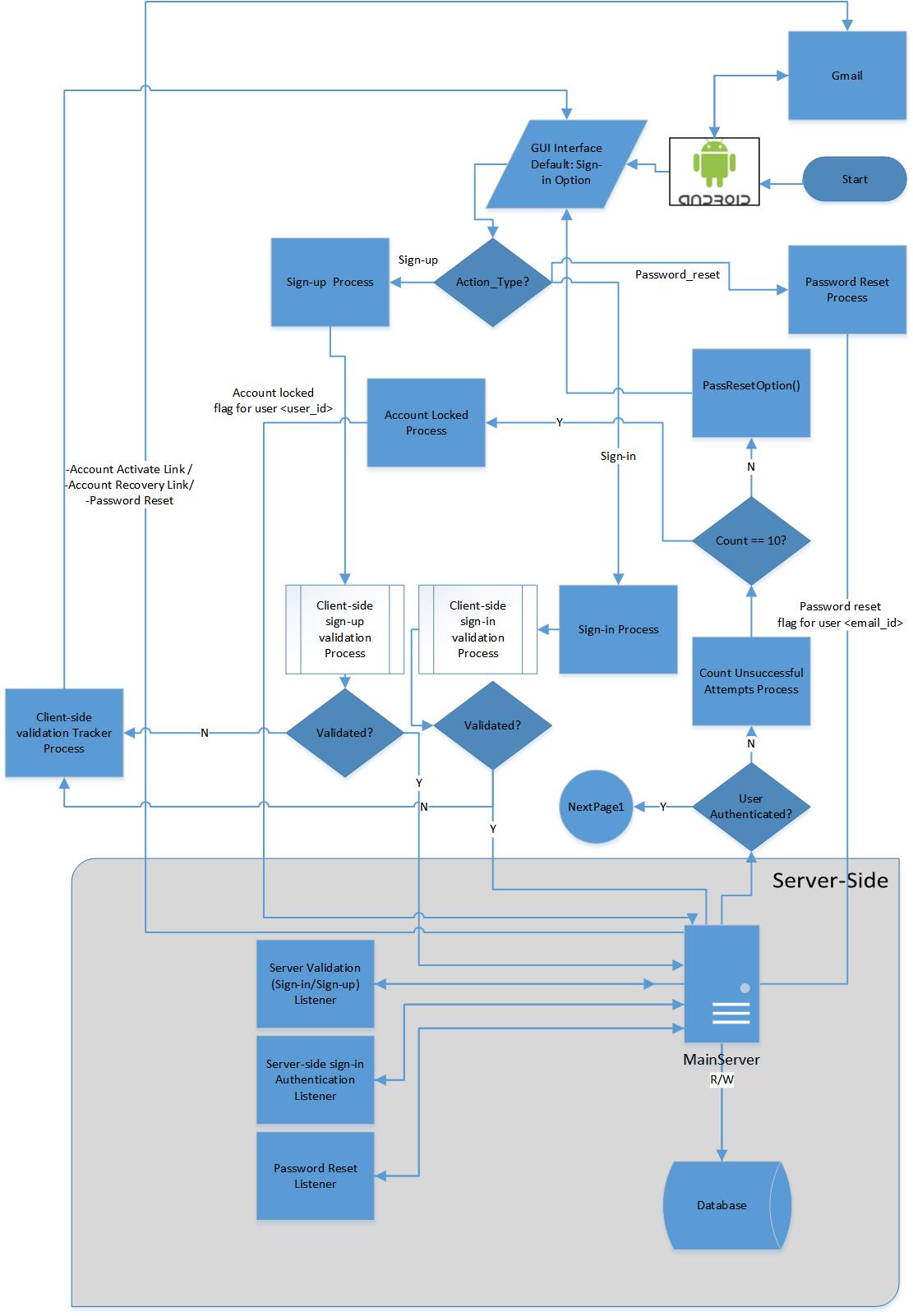
* 1. Flow-chart Diagram

Fig: Flow-chart User Authentication BCS

* + 1. Pseudo-Code User Authentication BCS

**Client**

LaunchApp();

Sign-inButtonActionListener(event onMouseClick) {

Sign-in(Sign-in Form Parameters) {

Boolean Sign-inValidation(); //returns TRUE if validated correctly

//do validation

If (noErrors) {

sendInfoThroughSocket();

return TRUE;

} }

Sign-upButtonActionListener(event onMouseClick) {

Sign-up(Sign-up Form Parameters) {

Boolean Sign-upValidation(); //returns TRUE if validated correctly

//do validation

If (noErrors) {

sendInfoThroughSocket();

return TRUE;

} }

If(Sign-in(Sign-in Form Parameters) == FALSE) {

passReset();

gotoGmailtoActivate();

**MainServer (Authenticating Server)**

SocketListener();

Sign-upDataListener();

If(Sign-upDataListener()) {

Boolean Sign-upValidation(); //returns TRUE if validated correctly

//do validation

If (noErrors) { //Means, according to server’s wish. That is no malicious codes etc.

ConnectAndStoreInfoToDatabase();

}}

Sign-inDataListener();

If(Sign-inDataListener()) {

Boolean Sign-ipValidation(); //returns TRUE if validated correctly

//do validation

If (noErrors) { //no Errors in validation

ConnecttoDatabase();

Authenticate();

If(Authenticate() == FALSE) {

COUNT++;

If(COUNT == 10) {

SendPassResetLinkToClient();

COUNT = 0;

} }

Else() { //User

COUNT = 0;

SendControllerInfoToClient();

AskControllerToSendControllerInforToClient();

}}

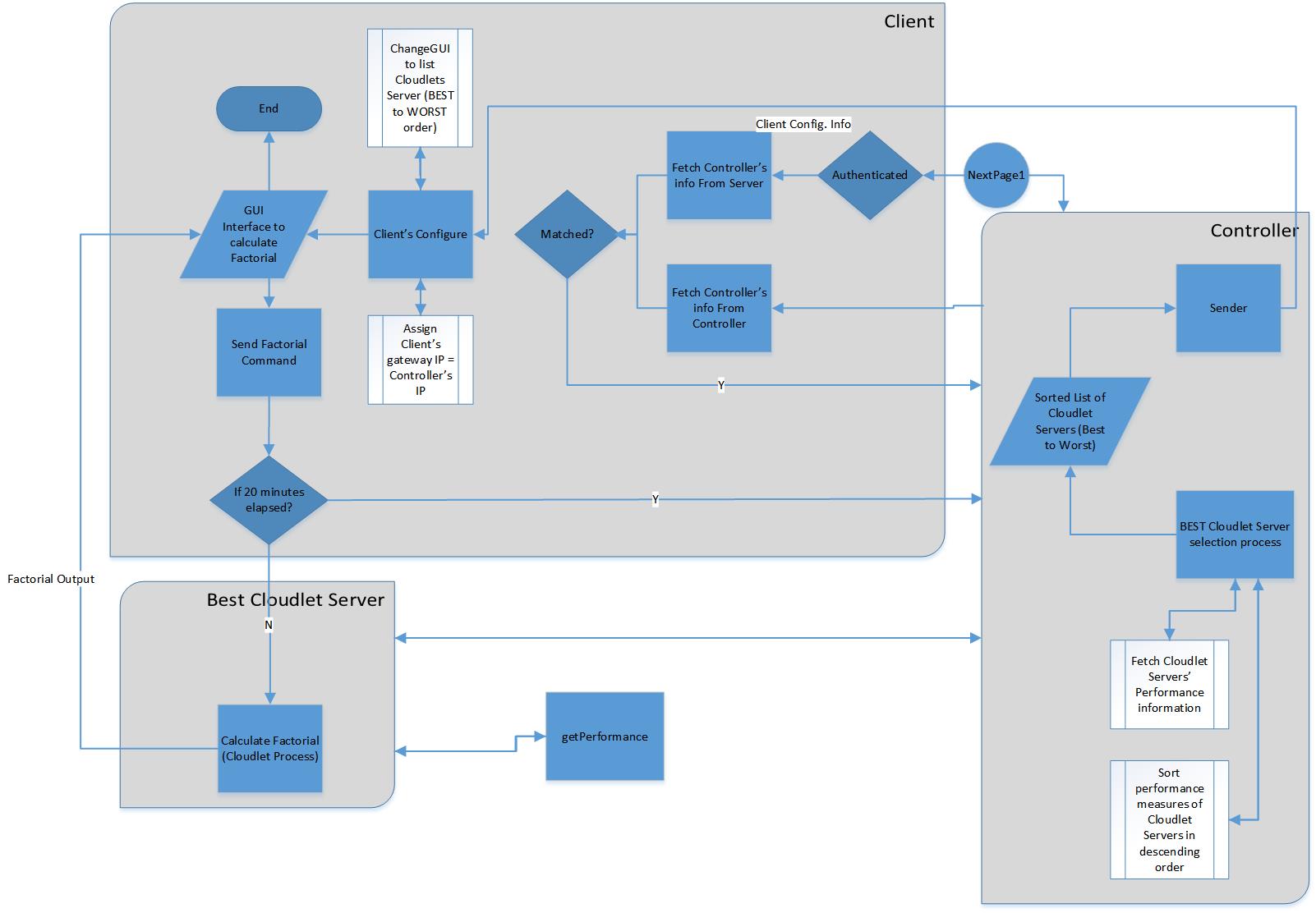


Fig: Flow-chart showing best Cloudlet Server is servicing computation to user

* + 1. Pseudo-Code of how Cloudlet Server services Clients

**Client**

Boolean MatchInfoFromControllerAndMainServer() {

//do matching

If(Matched) {

return TRUE;

}

Else {  
return FALSE;

}}

If(MatchInfoFromControllerAndMainServer() == TRUE) {

ChangeWifiGatewayAsCloudletControllerIP();

Check: AskControllertoProvideBestCloudletServer();

}

If(CloudletServerAvailable) {

If(CloudletServersUpdateFromControllerTime < 20 Minutes) {

SendComputationToBestCloudletServer();

ShowResultToUser();

}

Else {

goto Check;

}

**Cloudlet Server**

Boolean ComputationRequestListener();

If (ComputationRequestListener() == TRUE) {

//Perform Computation

//send computation to Client

}

**Cloudlet Controller**

FetchCloudletServersPerformanceInfoOnEveryAsk(); //each time Client asks, controller has to update Cloudlet Servers

OrderCloudletServersBasedonHighPerformance() {

If(All clients performance == SAME) {

//do logical ranking

}

Else {

// do ranking based on performance values sorting

}

AssignClientBestCloudletServer(); //assign 1st Cloudlet Server

* 1. Use-case Diagram

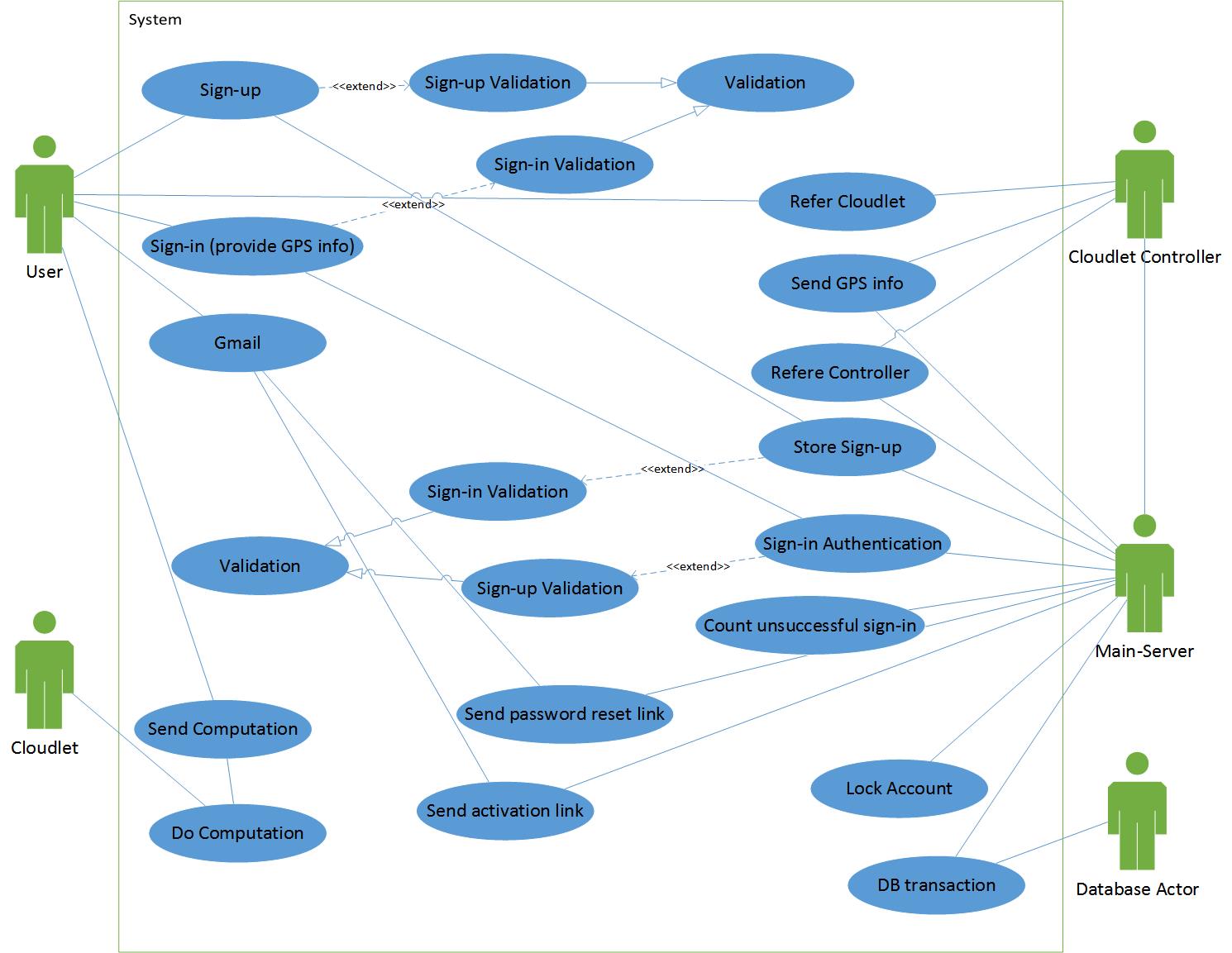


Fig: Use-case BCS

* 1. Sequence Diagram

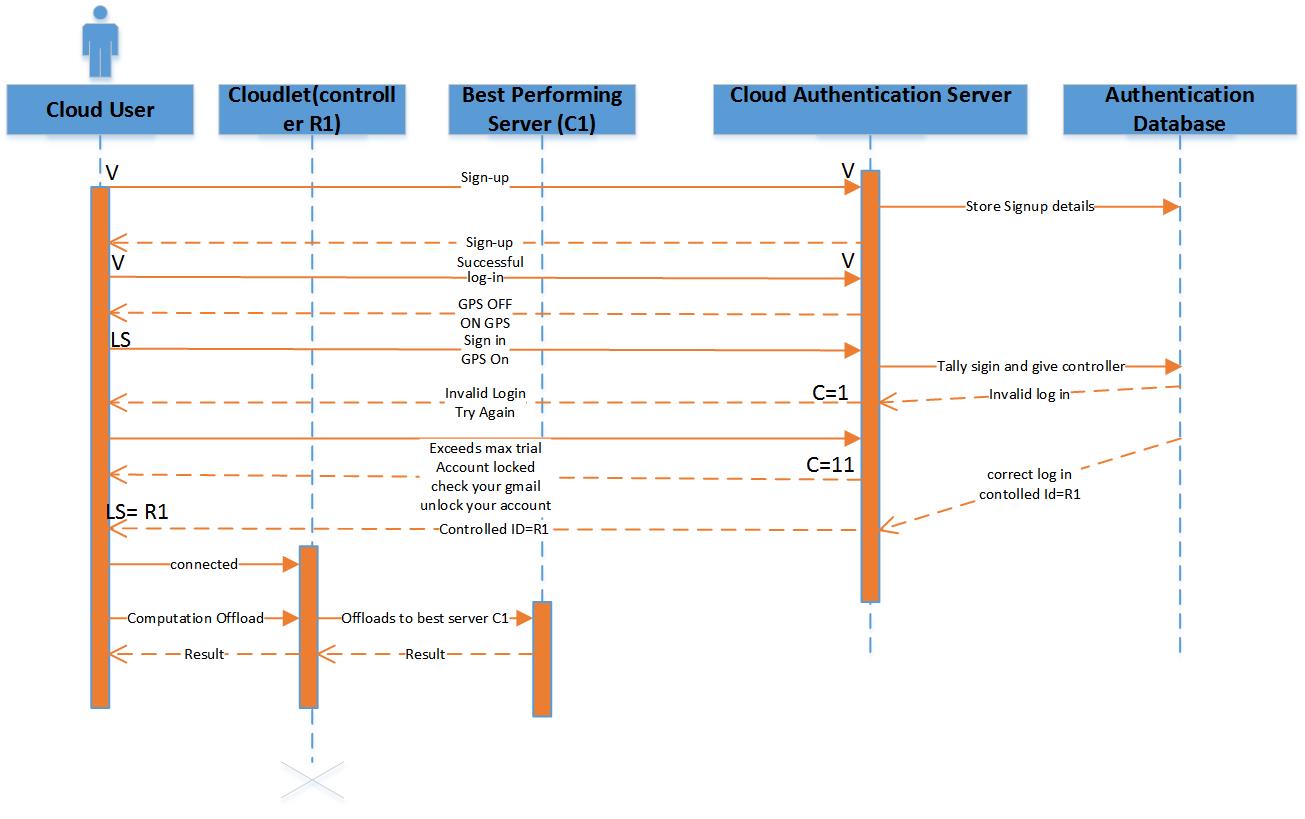


Fig: Sequence Diagram BCS

* 1. Activity Diagram

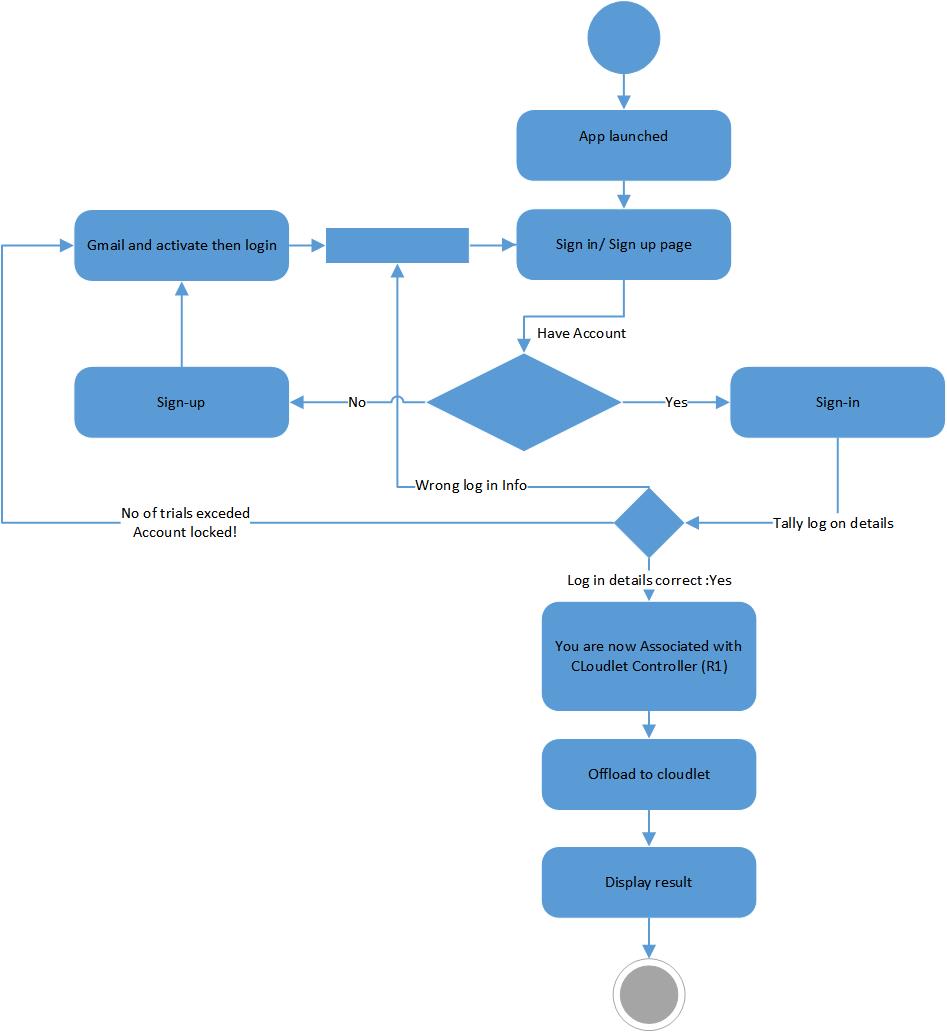


Fig: Activity Diagram BCS

Class Diagram

System Implementation

9) Gantt Chart

10) Conclusion

# References

|  |  |
| --- | --- |
| [1] | [Online]. Available: http://www.hindawi.com/journals/jam/2013/409539/fig2/. |
| [2] | [Online]. Available: https://www.internetsociety.org/sites/default/files/blogs-media/cdn-on-demand-affordable-ddos-defense-via-untrusted-clouds.pdf. |
| [3] | [Online]. Available: http://elijah.cs.cmu.edu/DOCS/lewis-milcom2014.pdf. |
| [4] | [Online]. Available: http://people.cs.vt.edu/~irchen/ps/Wang-wpc14.pdf. |
| [5] | [Online]. Available: http://arxiv.org/ftp/arxiv/papers/1206/1206.1118.pdf. |
| [6] | [Online]. Available: https://www.elsevier.com/\_\_data/assets/pdf\_file/0008/96947/Mobile-cloud-computing\_a-survey.pdf. |
| [7] | [Online]. Available: http://1.bp.blogspot.com/-Ey4LVVav72E/VcMqZbJvJNI/AAAAAAAAWqI/a9OmqVQ1Djs/s1600/activity-fragment\_lifecycle\_4.png. |
| [8] | [Online]. Available: http://zaam.se/wp-content/uploads/2014/07/android-activity-lifecycle.png. |