Mobile Testing as a Service

Pranjal Sharma
Department of Computer Engineering
pranjal.sharma@sjsu.edu

Nehal Sharma
Department of Computer Engineering
nehal.sharma@sjsu.edu

Amit Kamboj
Department of Computer Engineering
amit.kamboj@sjsu.edu

Rachit Saxena
Department of Computer Engineering
rachit.saxena@sjsu.edu

Abstract:
With the unprecedent growth in the usage of mobile phones, it becomes important to maintain the quality of mobile applications by conducting rigorous testing. This paper discusses the comprehensive design, implementation and validation of the functional components of a crowd sourced mobile testing as a service platform. Elastic provisioning of mobile device emulators, running test scripts in cloud on Android Virtual Devices using Appium server, Tester and Project Management Community, Google map view of testers locations, cost metrics based on resource consumption and bug severity and PayPal and Chatkit Pusher Integrations are the salient features of our application, which have been elaborated upon in this paper. We also carried out stress testing using JMeter to determine the load our application can handle (in terms of serving requests).

Keywords: Mobile Testing, MTaaS, Android Virtual Device, Emulators, Cloud Testing

1. Introduction:

As smart phones and mobile devices have become more popular, the demand for mobile applications has increased at a rapid pace. Considering this ever-growing demand, efficient and cost-effective testing methodologies are of prime importance for quality assurance of mobile applications. However, testing of mobile and web applications for multiple platforms, browsers and devices is extremely costly and tedious because of rapid evolution of mobile devices, technologies and platforms. Traditional testing methodologies of mobile applications in laboratory incur higher costs and difficulty arises in large-scale usability testing. Therefore, online crowd sourced testing has evolved as a favorable choice amongst the members of the software development community. It can be effectively used to circulate test scripts to a group of online testers across geographical boundaries to accomplish on-demand testing of mobile applications in a cloud infrastructure. Its advantages include reduced testing costs and large-scale concurrent testing.

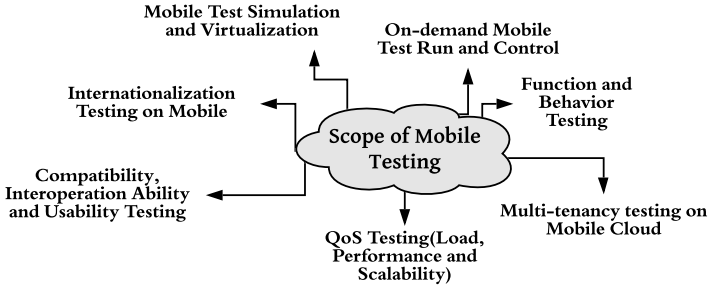

Figure 1: Scope of Mobile Testing

Our project, Mobile Testing as a Service, has been designed with the primary motive of facilitating the validation process of mobile software with included benefits of resource sharing and reduced costs for computation resources, memory and network infrastructures. This paper is structured in the following manner. The related works and their proposed solutions are presented in the immediate section. Our cloud based infrastructure design and the functional components are explained in Section 3. Section 4 talks about the database design and indexing algorithm and Section 5 discusses explores the scalability and load balancing strategies that we have used in our system, followed by the GUI design and flowchart in Section 6.Lastly, we evaluated our system performance and its results are presented in Section7.Conclusions, future enhancements and reference form the last sections of the paper.

2. Related Work:

Before beginning our work, our team spent a significant amount of time in comprehensive research about the on-going work in the related field of Mobile Testing as a service. As we mentioned in the earlier part of the paper, testing of mobile applications is complex and cost intensive because of the rapid technology upgradations, variety of

devices, device sizes and browser compatibility that need to be considered. A few papers that we came across such as Barbier and Ridene, 2011; Gao et al, 2013 discuss these challenges in detail. The next few papers by Gao et al., 2014 and Tao and Gao, 2014 discuss the technical ins and out about the Cloud TaaS infrastructure as an emerging business service model that aims at testing projects in the cloud environment. Various papers have also discussed the issues, challenges, needs and concerns for testing applications in a cloud environment(Aktouf, 2015; Gao et al., 2011, 2014;).These published works helped us in identifying the challenges, concerns and needs that currently exist in the MTaas domain so that we could work on overcoming at least a few of them in our system.

3. Cloud based System Infrastructure and Components:

In our group project, we have designed a mobile test service platform for testing intelligent mobile applications to ensure excellent and non-buggy end-user experience.

3.1 Basic user groups:

a) Freelance Testers: This group of users is able to sign up/into the application, update their profile, take-up a project for testing, view testing scripts for the project and receive alerts and notifications from other testers or project managers related to the project. The tester can test execute the testing script on a virtual emulator and submit the results to the project manager. The testers will receive incentives based on number of bugs covered and their severity upon completion of the testing process. The tester will also have to choose the correct emulator for testing the script depending on the language the script is in.

b**) Mobile Application Project Managers:** This group of users will be able to access the application using the sign up/in and update profile facility. The project managers can own and update (add/delete/set deadline) test project details and share related resources as required by the testers to test the application. Managing the project artifacts such as testing scripts, testing data, testing metrics is also an important duty being carried out by the project managers. The Project Manager can view his testers of a project based on their location. For this, we have integrated Google Maps API, which plots the testers using markers on the Google Map.

c) Operation Service Staff: This is the last group of users who are majorly responsible for accessing, tracking, monitoring user accounts (includes blocking), projects and the resources related to them.

3.2 Testing Community Service Feature: Another salient feature of our application is the 'Testing Community Service' functionality which would include Project Oriented Community Groups and Tester Oriented Community Groups. It would allow members to post messages for project managers in the community groups, project managers to post alert messages for freelance testers respectively and send one-to-one messages to one another using personal chat. For communicating via chat, our application makes use of Pusher Chatkit application (3rd party API). This API allows the tester/manager in our application to create a dedicated public/private chat room, add members to the room, delete members from the room, subscribe to messages from a particular room, show read receipts and also delete the room. It also allows users to send one to one messages to a member of the room directly (i.e., not publicly in the group).

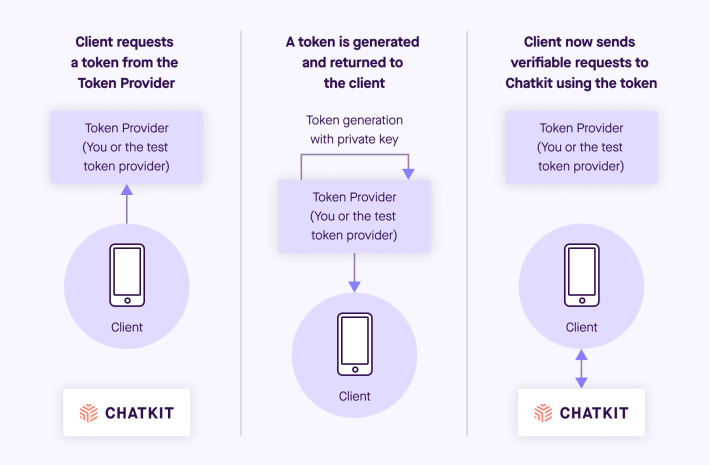

Figure 2: Pusher Chatkit APIs Usage

3.3 Test Runner:

Our application provides support to testers for executing, controlling, monitoring and analyzing test scripts. The system allows the manager to upload scripts for any platform. However, the execution can only be carried out for scripts on Android platform via Appium server. For this purpose, the tester can request an Android Virtual Device (AVD) on-

demand basis with the required memory, version, storage etc. Once the virtual device has been created, the tester can execute the script to discover bugs and eventually submit the bug report to the manager on the MTaaS platform vis the 'Submit Report' tab.

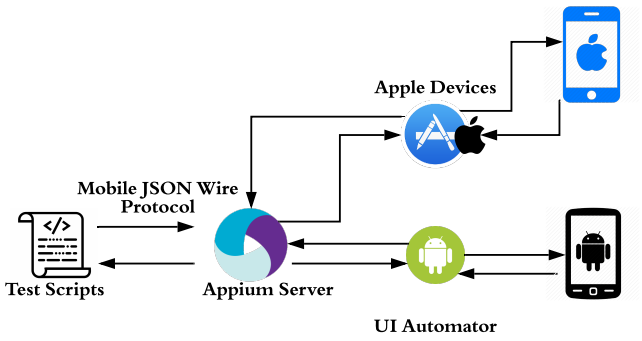

Figure 3: Execution of Test Scripts using AVD and Appium Server

As of now, only test script execution in Android platform is permitted, but we plan to integrate test script execution for other platforms as well in the future enhancements.

3.4 Dashboard:

The application allows the project manager and freelance testers to visualize statistics on their respective dashboards. The statistic charts take the data from the Database and use React ApexCharts.js library to implement the visualization. The Tester dashboard allows the tester to visualize the number of bugs that he has found for each project that he has been involved in. Also, he can view the number of projects he is currently working on. The Manager dashboard allows the manager to visualize the number of testers in each of his projects. The system administrator dashboard provides a. visual view of the resource utilization in terms of CPU cores, RAM and storage.

3.5 Billing and Incentives Metrics Module: The incentives paid to the freelance testers would be calculated cumulatively on the number of bugs found and their severity. The bugs severity standards would be predefined into low, medium and critical and the cost for the same would be determined by the project manager. Since these costs are defined by the project managers, they are specified in the project specification document of each project.

The project managers are also charged on the basis of pay-as-you-go model for the resources they have utilized for testing purposes. For this, the factors that we have taken into account include the hardware resources (CPU cores, RAM and storage), that are allocated on-demand basis. It is the role of the system administrator to manage payments from the manager.

Client	Requested Resource	OS	RAM(MB)	RAM Usage Hours	CPU Cores	CPU Hours	Storage (GB)	Storage Hours	Cost($)
Client A	Mobile Device	Android	512	2.35	4	9	4	12	(512*2.35*2)+(4*9*4.50)+(4*12*3.25)=$2724.40
Client B	Mobile Device	Android	128	1.45	2	6	8	4	(128*1.45*2)+(6*2*4.50)+(8*4*3.25)=$529,20
Client C	Mobile Device	iOS	256	0.3	1	2	2	5.5	(256*0.3*2)+(1*2*4.50)+(2*5.5*3.25)=$198.46

RAM cost per hr	CPU core cost per hr	Storage cost per hr
$2	$4.50	$3.25

Figure 4: Cost Calculation and charges for Resources

Transactions are handled by our application using PayPal Sandbox API integration. The flowchart for the same is given below:

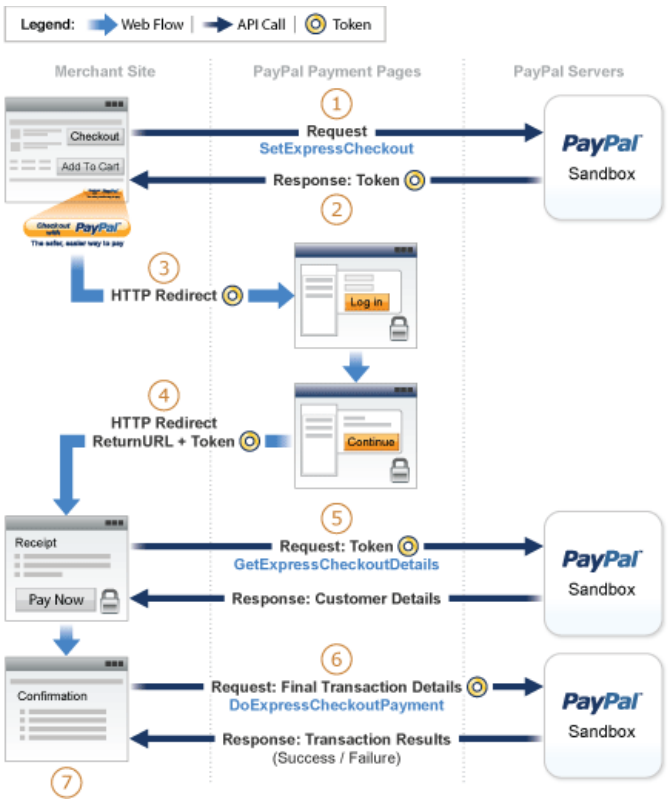

Figure 5: Express Checkout flow using the PayPal sandbox as the API server

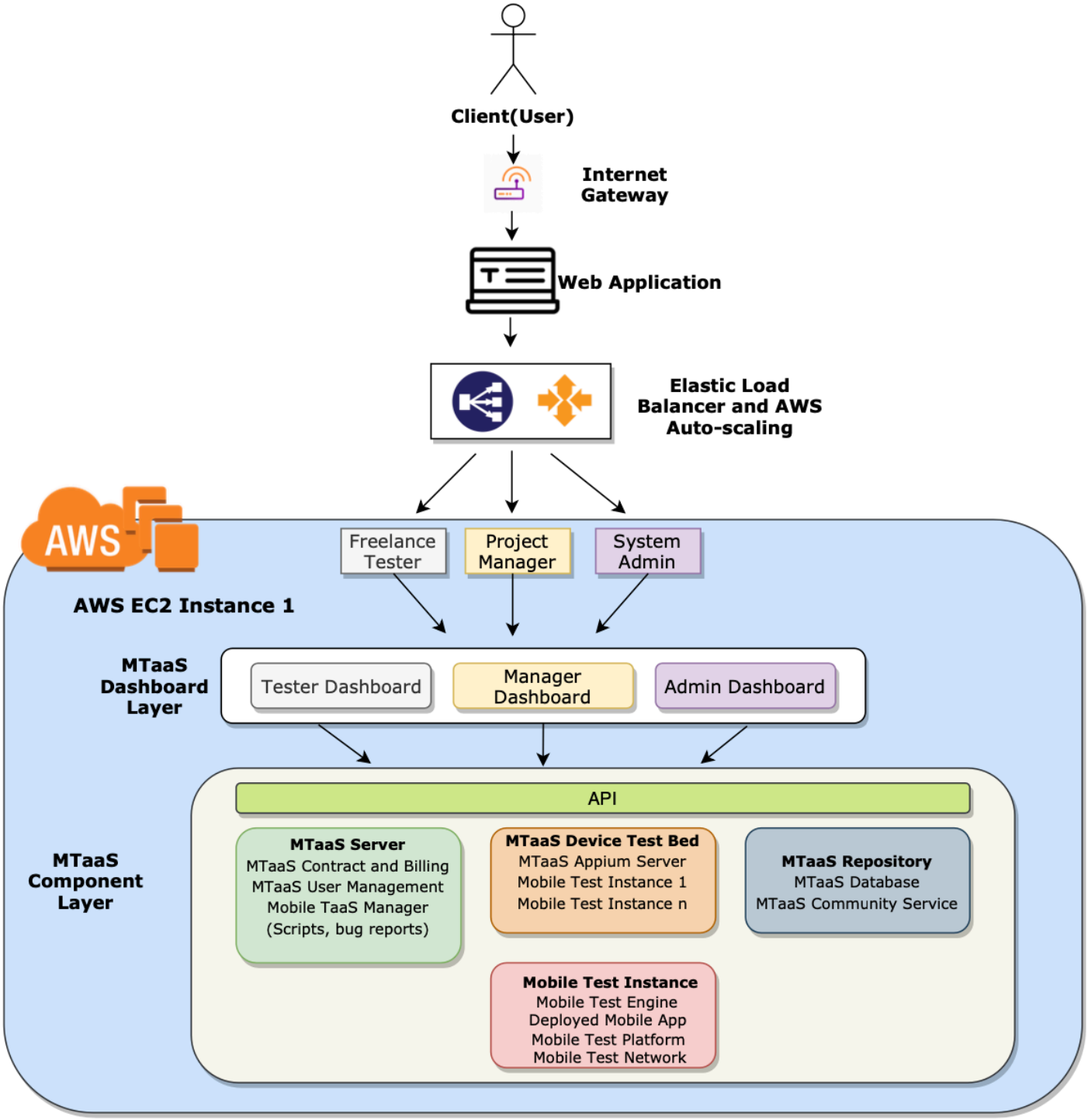

Figure 6: System Component Diagram

4. Data Design and Implementation:

For our application, we have used NoSQL database, MongoDB to store all the data. Our choice of NoSQL MongoDB over relational database MySQL was governed by the following factors:

(i)MongoDB is highly scalable and offers high performance coupled with automatic scaling and in-built sharding mechanism, which takes care of distributing data across multiple machines.

(ii)It is schema-less and can store unstructured and semi-structured data, which is appropriate for storing data such as chats, which may contain audio and image files also. Also supports geo-spatial indexes to provide support for querying geographical location based data.

(iii)MongoDB supports 'write-lock' mechanism and provides dynamic consistency to ensure the success for each write operation by blocking all other operations.

Hence, as cloud technologies are gaining prominence and popularity, we decided to go with MongoDB Atlas for managing and storing the data in our application. Given below is a schematic representation of the MongoDB collections used in our application.

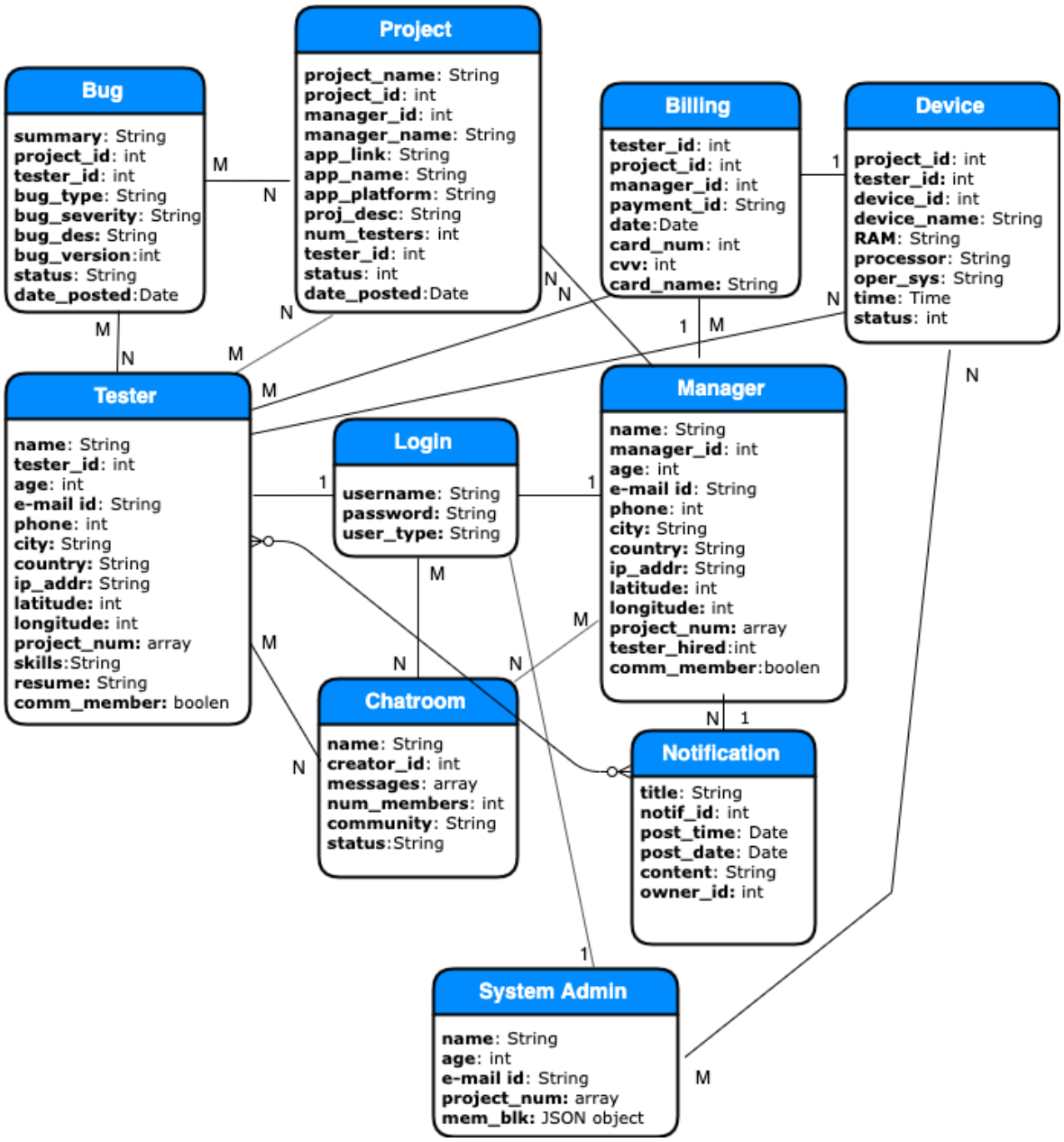

Figure 7: Database ER Diagram

4.1 Indexing Strategies used for MongoDB queries:

After researching about indexing mechanisms, our team realized that the most effective indexes take into consideration a variety of factors such as: the type of queries that may be expected on the system, the ratio of red to writes etc. Therefore, it is crucial to completely understand our system before

proceeding with any indexing strategy. We started out by listing all the types of queries that we expect the users to perform and their frequency so that we could create our index around those fields. In our application, most of the queries that will be performed by the freelance tester and project managers would be based on the project field. So, we designed an indexing scheme on the 'project_id' field, since it uniquely identifies each project. We created a single key index as follows, which greatly increased the query performance:

db.projects.createIndex({"project_id": 2345})

5. System GUI Design and Implementation:

For designing the Graphical User Interface of our system, we started out by understanding the requirements for the three categories of our system users i.e., freelance testers, project managers and system administrators. Since the roles are different for each user, each of them has different privileges and therefore, the GUI had to be designed accordingly so that each user has a view which is relevant to him/her.
For designing the user interface, initially we developed a low-fidelity prototype using Balsamiq (few screenshots shown below). After 2 repetitive iterations for prototyping, we actually started out with implementing the high-fidelity design of the application using ReactJS.

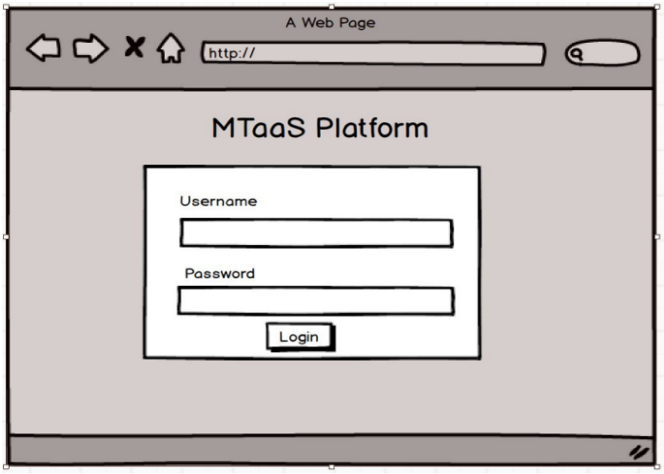

Figure 8: Low-fidelity GUI login screen prototype using Balsamiq

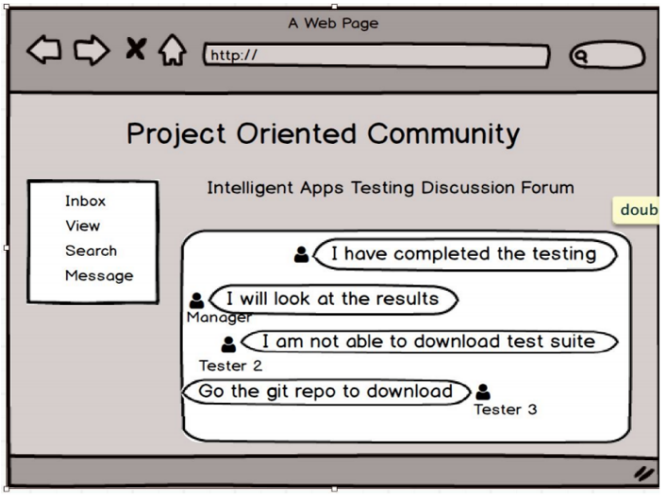

Figure 9: Low-fidelity GUI chatroom screen prototype using Balsamiq

The users would be interacting with the application using RESTful web service. Some of the RESTful services we have used in our application are given below: (this is not an exhaustive list and the actual application contains many more)

1)/User/SignUp
Method Used: POST
Purpose: Sign up facility to the user based on the roles

2)/User/Login
Method Used: POST
Purpose: Sign in facility to the user based on the roles

3)/User/AccountUpd
Method Used: POST
Purpose: Allows User to update their information

4)/Tester/ViewProjects
Method Used: POST
Purpose: Retrieves a list of projects seeking testers

5)/Billing
Method Used: POST
Purpose: Lets the user view the billing based on pay-as-you-go model

6)/DeleteProject
Method Used: POST
Purpose: Lets the project manager delete the resources and postings related to a particular user

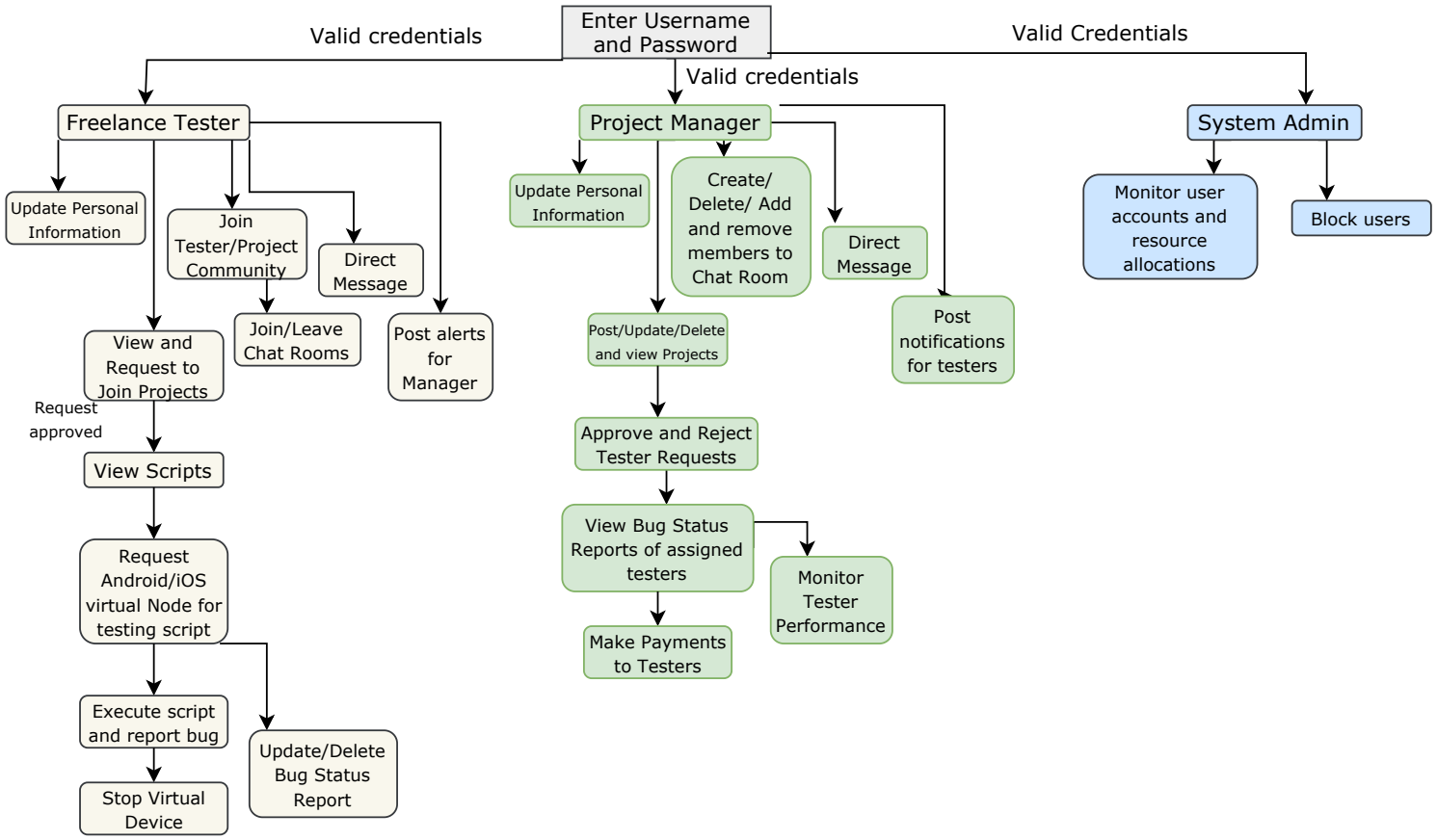

Figure 10: GUI Function Partition Diagram

6. System Services (Auto scaling and Load Balancing):

The resources i.e., Android Virtual Devices are allocated to the user on-demand basis when he makes a request for one, using the GUI.

For autoscaling, we have used EC2 instance auto scaling, which creates an AMI image of the instance and based on the autoscaling condition, it either spins up a new instance or closes down an existing instance. The condition that we have specified for our instance autoscaling are based on:

1)**Number of Requests:** When the number of requests increases more than 8000, spin up a new instance and shut down an existing instance, when the number of requests fall below 4000.These figures are based on the JMeter load testing, which is described in the following section

2) CPU Utilization: When new virtual devices are started for executing scripts, they require memory and utilize CPU. Hence for autoscaling, we spin a new instance when the CPU utilization crosses the 70% mark and shut down an instance when it is below 30%

For load balancing, we have used the AWS Classic Load Balancer, which directs the traffic to a new instance when it spins up.

7. System Performance Evaluation and Experiments:

Throughout the development cycle, we carried out rigorous testing mechanisms to ensure correct functioning and optimal performance of our application. We carried out validation testing during the development phase and at the end to evaluate whether our system satisfies the client's needs and business requirements. Following were the main components of validation testing cycle that we carried out: Unit Testing, Integration testing, System Testing and Acceptance Testing. For our project, we also carried out functional testing by designing test cases to cover all functionalities to effectively seal all loopholes. For testing the performance of our application, we used Apache JMeter to record the average response time for different number of users and requests sent. The results played a pivotal role in helping us determine the maximum number of requests our system could handle without registering a degrade in performance. We tried to generate client behavior for different APIs in our system and produced load by making several concurrent requests to the application to estimate how the performance changes as the load progresses from low to normal to high(stress test cases).Stress tests are beneficial in evaluating the bottlenecks and hence planning for allocating more resources when the traffic starts affecting the application performance. We carried out JMeter testing for 200, 500, 1000, 5000 and 100000 requests and noted the time taken by the server to serve the requests. The graph for the results(throughput) are shown below (green lines represent throughput):

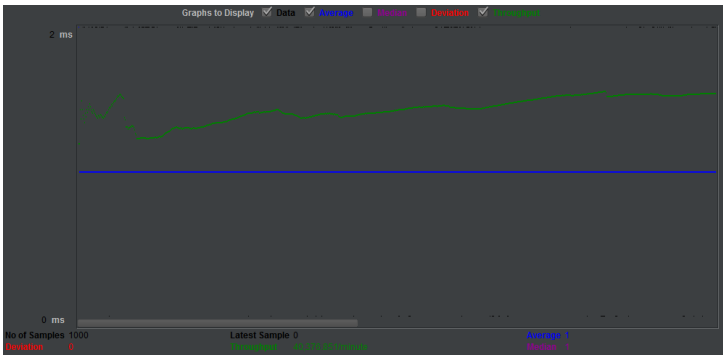

Figure 11: JMeter Graph for throughput for 1000 calls

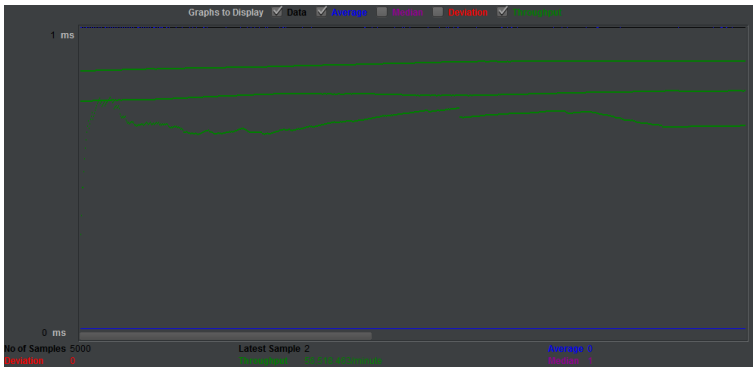

Figure 12: JMeter Graph for throughput for 5000 calls

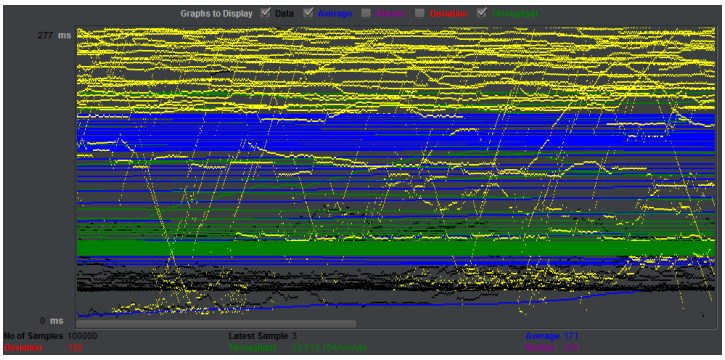

Figure 13: JMeter Graph for throughput for 100000 calls

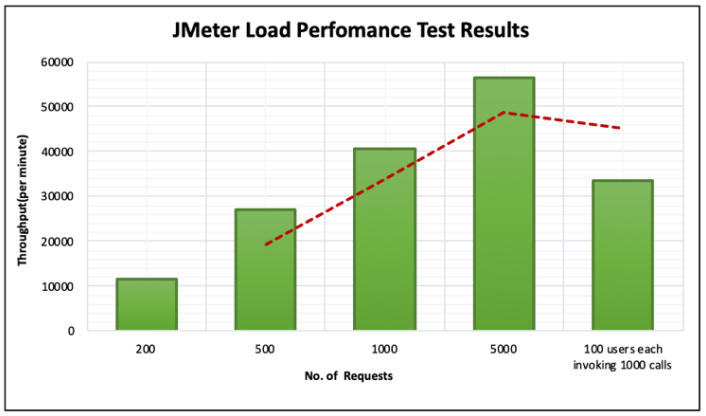

Figure 14: Load Performance Test Results

From the above graph, we can see that the throughput consistently increases as the number of request increase. However, the values register a slight decrease when the number of requests reach 100000.

8. Conclusion and Future Enhancements:

Through this paper, our team has successfully tried to explain our design and justify our system architecture choices for an MTaaS. It is a platform for crowd sourced testing of mobile applications which provides the facility to run test scripts on android virtual devices by allowing the tester to request virtual mobile nodes to simulate a testing environment. We have discussed the cloud-based system infrastructure, functional components, database design, load balancing and scaling mechanisms and the services offered by our system in great detail. To validate our system, we also conducted load performance testing and its results indicate that the server can handle user requests effectively under high load scenarios. Given below is a Gannt chart for our project tasks and completion milestones.

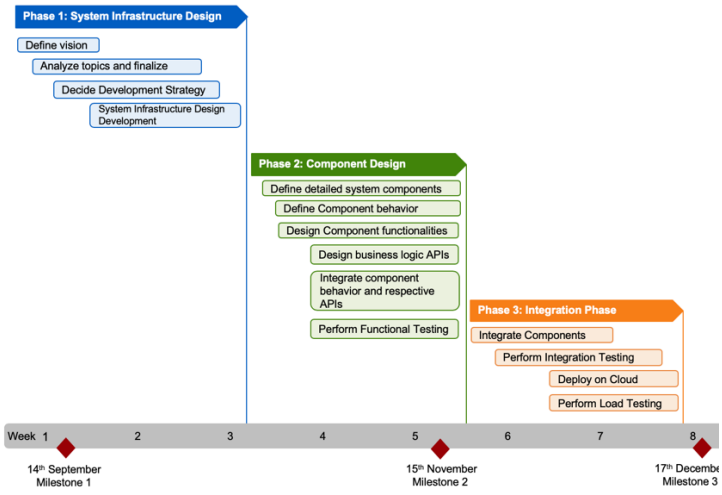

Figure 15: Gantt Chart

Our current application can successfully execute android test scripts by running Android Virtual Device through Appium Server. As the future pathway for our application, we intend to integrate automated test script execution support by emulating virtual iOS devices for iOS script testing as well. The cloud computing domain has evolved rapidly and our team is grateful to Dr. Gao for giving us an opportunity to work on a project which aims to deliver mobile testing as a service through crowd sourced testing at a low cost.

9. References:

1. Guilherme Antonio Borges, Romulo Reis de Oliveira, Tiago Coelho Ferreto, Claudio Fernando Resin Geyer, "A Novel Model to Computational Offloading on Autonomic Managers: a Mobile Test Bed", *High Performance Computing & Simulation (HPCS) 2018 International Conference on*, pp. 157-164, 2018
2. Amro Al-Said Ahmad, Pearl Brereton, Peter Andras, "A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods", *Software Quality Reliability and Security Companion (QRS-C) 2017 IEEE International Conference on*, pp. 555-562, 2017.
3. Jerry Gao et. El, "Mobile Testing-As-A-Service (MTaaS) –Infrastructures, Issues, Solutions and Needs", IEEE 15th International Symposium on High-Assurance Systems Engineering 2014, pp. 158-168
4. Zachary Parker, Scott Poe and Susan V. Vrbsky, "Comparing NoSQL MongoDB to MySQL DB", ACMSE '13, April 4-6, 2013, Savannah, GA, USA, pp.36-42
5. *Xamarin test cloud: Mobile app testing made easy*, 2016, [online] Available: https://xamarin.com/test-cloud/.
6. Jia Yao, Babak Maleki Shoja, Nasseh Tabrizi, *Cloud Computing – CLOUD 2019*, vol. 11513, pp. 303, 2019