International Journal of Advanced Smart Convergence Vol.5 No3 32-39 (2016)
http://dx.doi.org/10.7236/IJASC.2016.5.3.32

IJASC 16-1-3

Addressing Concurrency Design for HealthCare Web Service Gateway in Remote Healthcare Monitoring System

Lionel Nkenyereye, Jong-Wook Jang[†]

[†]Department of Computer Engineering, Dong-Eui University,
176 Eomgwangro, Busanjin-Gu, Busan, 614-714, Korea
e-mail: lionelnk82@gmail.com, jwjang@deu.ac.kr

Abstract

With the help of a small wearable device, patients reside in an isolated village need constant monitoring which may increase access to care and decrease healthcare delivery cost. As the number of patients' requests increases in simultaneously manner, the web service gateway located in the village hall encounters limitations for performing them successfully and concurrently. The gateway based RESTful technology responsible for handling patients' requests attests an internet latency in case a large number of them submit toward the gateway increases. In this paper, we propose the design tasks of the web service gateway for handling concurrency events. In the procedure of designing tasks, concurrency is best understood by employing multiple levels of abstraction. The way that is eminently to accomplish concurrency is to build an object-oriented environment with support for messages passing between concurrent objects. We also investigate the performance of event-driven architecture for building web service gateway using node.js. The experiments results show that server-side JavaScript with Node.js and MongoDB as database is 40% faster than Apache Sling. With Node.js developers can build a high-performance, asynchronous, event-driven healthcare hub server to handle an increasing number of concurrent connections for Remote Healthcare Monitoring System in an isolated village with no access to local medical care.

Key words: *concurrent application; asynchronous I/O; server-side JavaScript; web service gateway; Internet of Things; remote healthcare monitoring system; Node.js; web services.*

1. Introduction

The idea of devices connected with each other leads to an enormous systems based Internet of Things (IoT) such as IoT-related healthcare systems. The implementation of these IoT-related healthcare is based on the definition of the IoT as a network of devices connected with each other in order to capture and share useful data through wireless connectivity and Internet access capabilities that connect to a central processing and controlling server in the cloud[1].

Manuscript Received: July 17, 2016 / Revised: July 28, 2016 / Accepted: August 05, 2016
Corresponding Author: jwjang@deu.ac.kr
Tel:+821056000845, Fax: +82518901724
Department of Computer Engineering, Dong-Eui University,
176 Eomgwangro, Busanjin-Gu, Busan, 614-714, Korea

Remote Healthcare Monitoring (RHM) is a technology to enable monitoring of patients reside in an isolated village with no access to effective health monitoring of conventional clinical settings[2]. However, most of RHM technologies follow a general architecture that consists of four components[2]. First component is variety of sensors on a device that is enabled by wireless communications to measure physiological parameters. The second component is a local data storage on the coordinator mobile device (smartphone) at patient's site that interfaces between sensors and centralized data repository and healthcare monitoring hub server. The coordinator device is equipped by wireless connectivity such as Wi-Fi or 3G/4G LTE enabled internet to send collected physiological data to healthcare providers. The third component is a centralized repository to store data sent from sensors, local data storage, diagnostic application, and healthcare providers. The last component is the diagnostic application software that send a real-time alert to the patient based on the analysis of collected data by triggering a series of actions previously agreed upon by the patient and medical practitioners. In this paper, we use the term healthcare web service gateway for referring to the third and fourth component. The Fig.1. shows the architecture for RHM system.

The healthcare web service gateway of the RHM system aims to handle a large number of requests from patients in a concurrency manner at the point to handle treatment recommendations efficiently. Concurrency is the tendency of things that happen at the same time in a system. Concurrency is particularly important in RHM system that have to react to outside events in real-time and that often have hard deadlines to meet[3]. As the number of patients' requests increases in simultaneously manner, the healthcare web service gateway located at the village hall encounters limitations for performing them successfully. Therefore, this RHM application fails to detect deterioration of patient's health, slows down predefined actions previously agreed upon by the patient and the doctor. As consequences, number of emergency department visits increases, hospitalization and duration of hospital stays increase healthcare delivery costs. In addition, patient cannot receive real-time alerts which are a series of actions that take place automatically without the help of doctors. These alerts remind the patient to take some precautions for instance. This situation can cause the death of patient when the doctor cannot receive new data and the patient's history from the healthcare hub server. Furthmore, the patient should not benefit any immediate medical attention.

When we talk about concurrency, two approaches are proposed[4]. The first approach is to have multiple threads in a single process. The second approach is to have multiple single-threated processes. Using separate processes for concurrency also has an additional advantage. The advantage is to run the separates processes on distinct machines connected over the network. Though this increases the communication cost. The alternative approach is to run multiple threads in a single process. Each thread runs independently of the others, and each thread may run a different sequences of instructions. The disadvantage of the multiple threads approach is that all the threads in a single process share the same resources. In case the shared resource fails, the active threads include in the single process are destroyed as well.

In this paper, we propose the design tasks of the RHM application for handling concurrency tasks. In the procedure of designing tasks, concurrency is best understood by employing multiple levels of abstraction[5]. First, the functional requirement of the RHM system must be well understood in term of its desired behavior. Next, possible function for concurrency should be checked into. This is best done using the abstraction of threads. The way that is eminently to accomplish concurrency is to build an object-oriented environment with support for messages passing between concurrent objects and minimize or eliminate the use of multiple threads of control with a single process [6]. In software development, Node.js[7,8], a cross-platform runtime environment for developing server-side web service features technologies for designing RHM system.

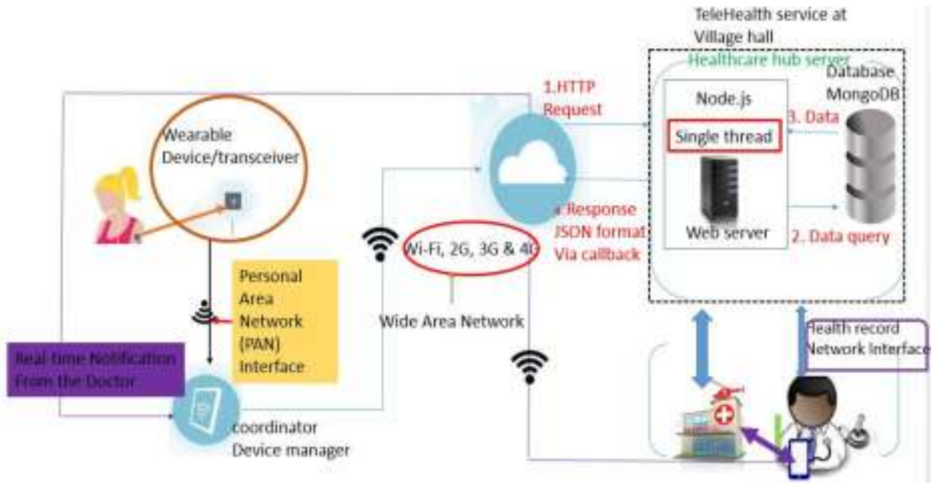

TeleHealth service at Village hall
Healthcare hub server
Database MongoDB
Node.js
Single thread
Web server
1.HTTP Request
3. Data
2. Data query
4.Response JSON format Via callback
Wearable Device/transceiver
Wi-Fi, 2G, 3G & 4G
Wide Area Network
Personal Area Network (PAN) Interface
Real-time Notification From the Doctor
coordinator Device manager
Health record Network Interface

Figure 1. Architecture of Remote healthcare monitoring system in an isolated location with no access to local medical care.

The main contributions of this work are as follows:

- Design tasks that make sense to construct concurrent system for RHM application operates on the healthcare web service gateway located at the village hall. The design of components that constitute the healthcare minitoring gateway based web service are also described.
- To investigate the performance of healthcare web service gateway that includes backend server, web programming framework and database able to support a large and increasing number of concurrent patients' requests. The performance metrics are throughput, response time and error rate to compare server-side javaScript implementation using Google's V8 javaScript engine and Rhino.

The rest of this paper is organized as follows. First, we discuss background of the technology related to the study in Section II. We elaborate a general system design model for building healthcare web service gateway in Section III. In Section IV, we brief the simulation, experimentation and analysis of results that state the performance of server-side javaScript to handle concurrency connection in remote healthcare monitoring system. A conclusion in Section V sums up our performance of Node.js and the advantages should bring when deploying applications which require high-performance, asynchronous, event-driven network server with lowly resource requirements.

2. Background and Technology Related to the Study

2.1 Internet of Things

Internet of Things has been developed to efficiently interconnect billions of devices that can sense, communicate and share information, all interconnected over public or private internet Protocol (IP) networks [1]. IoT has recently become more relevant to the everyday world because of the growth of mobile devices, embedded and ubiquitous communication, cloud computing and data analytics. With more physical objects and smart devices connected, the impact and the value that IoT brings to our daily services have grown exponentially. For instance, powered by these smart devices connected over the Internet, users can take the best routes to drive to work or choose their favorite restaurant.

2.2 Server-side JavaScript programming

When it comes to server-side JavaScript programming, developers have choices between Rhino javaScript engine and V8 based solutions like Node.js, SilkJS, and others[9]. Rhino[9] is a java-based JavaScript interpreter that gives JavaScript programs access to the entire Java API[9,10]. The Node.js is a platform for building event-driven networking programs. It is a version of Google's V8 JavaScript. It runs single thread that makes it to serve many clients concurrently. As shown in the Fig. 2., the event loop queues both new requests and blocking I/O requests. The single-thread executes an event loop by setting up a simple mapping of all requests. The event loop gradually dequeuing request from the queue, then processing the request, finally taking the next request or waiting for new requests submitted[8,9].

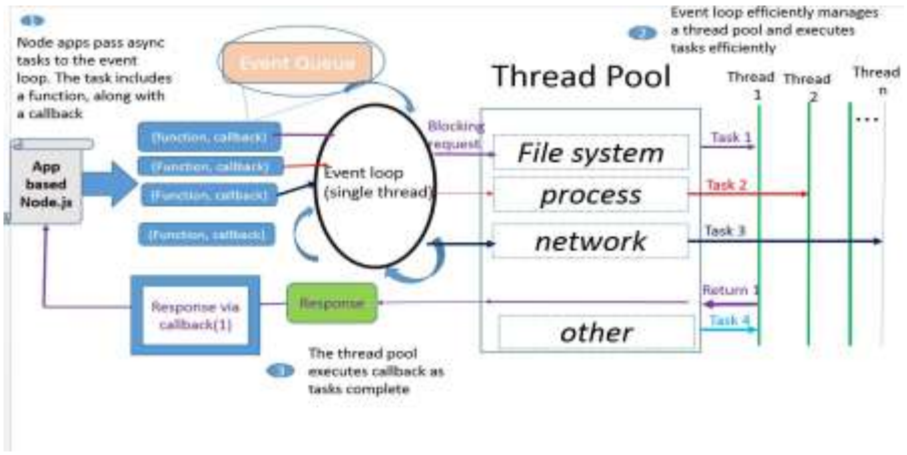

Node apps pass async tasks to the event loop. The task includes a function, along with a callback
Event Queue
Event loop efficiently manages a thread pool and executes tasks efficiently
Thread Pool
Thread 1
Thread 2
Thread n
App based Node.js
(function, callback)
Event loop (single thread)
Blocking request
File system
process
network
other
Task 1
Task 2
Task 3
Return 1
Task 4
Response via callback(1)
Response
The thread pool executes callback as tasks complete

Figure 2. Node.js is internally based around the concept of an event loop. There is an infinite loop running in a single process that continually monitors what event occurred and what callbacks need to be executed. It deals with queuing all events automatically, and keeps calling the appropriate callbacks as fast as it can [8].

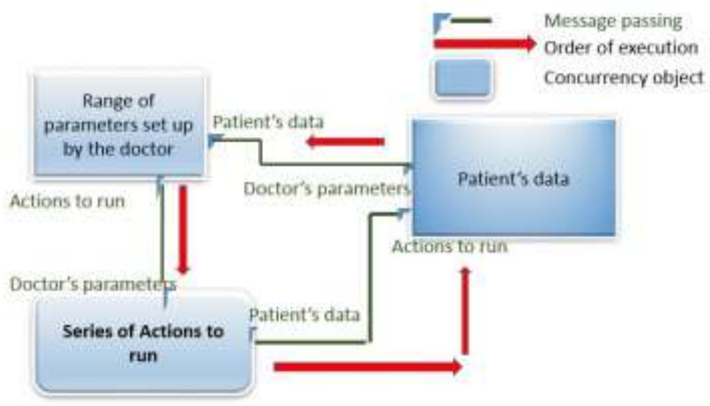


Figure 3. Design of concurrent objects for building RHM application on healthcare web service gateway.

Using Node.js allows to write scalable network applications that inherit concurrency in their design. This is accomplished by event-loop. To avoid blocking, all main Application Programming Interface (API)-calls that involve Input/output are asynchronous. Thus, instead of waiting the function to return, the event-loop can execute the next event in the queue, and when the API-call is finished it calls a callback function specified when the call was made. There is an infinite loop running in a single process that continually looks at what event occurred and what callbacks need to be executed. It deals with queuing all events automatically, and it keeps calling the appropriate callbacks as fast as it can. As a developer using Node.js, you do not really need to know this, though you just need to know that your callbacks will be called when events occur as quickly as possible the task completes[8](See Fig. 2.).

3. Design of Concurrency pattern on Healthcare web Service gateway for RHMS

Taken by concurrent components, neither blocks of software nor data structures nor procedures make very natural models for concurrent components but objects seem like a very natural way to design a concurrency software [6].

The Fig.3. illustrates objects as concurrent components for building RHM application on healthcare hub server. Consider an object model for our RHM system located at the village hall. A patient's status data object allows collection of patients 'data, then compares against a range of parameters set up by the doctor (physicians) object. Based on the results, the object model for our RHM send a message to series of actions to run object which executes a series of actions previously agreed upon by the patient and the doctor. Here execute actions by notifying the patient and updating patient's history data on doctor's (or physicians) tablet enable them to better assess the condition of the patients and reach more accurate diagnosis.

In order to support asynchronous communication, a current running instance of Node.js listens to all new incoming tasks. The running instance features event loop on which is provided with a message queue. The running instance refers to active object. The passive objects corresponds to the different tasks being run against the Node.js applications such as Range of parameters set up by the doctor, patient's data (Fig.3.). The passive objects include a function and a callback that corresponds to the response to be executed by the active object (thread pool) as soon the task encapsulates in the passive object is completed. The callback holds the state information of the passive objects that performed non-blocking and long tasks.

4. Building healthcare web service gateway using node.js and experimentation results

The Fig. 4 below illustrates how we can use a Node.js app deployed on edge to perform to write patient's data onto MongoDB database. Thus, MongoDB fits perfectly for Node.js applications. Therefore Node.js and MongoDB letting us write JavaScript for the backend and database layer . Furthermore, MongoDB is known for its schemaless nature gives a better way to match the constantly evolving data structures in remote on-line vehicle diagnostics applications. To start with, patient's data is transmitted via internet. This data can be converted into standard format for storage, and made available for both private patient's physician and health record network such as hospital.

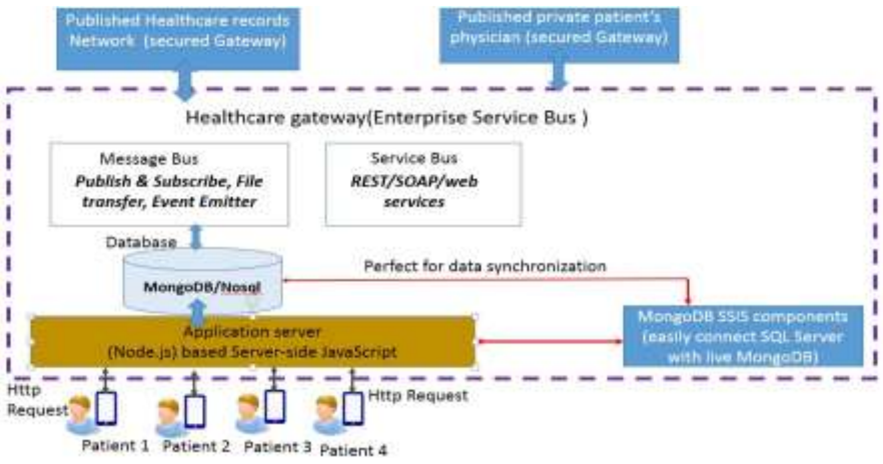

Published Healthcare records Network (secured Gateway)
Published private patient's physician (secured Gateway)
Healthcare gateway(Enterprise Service Bus)
Message Bus
Publish & Subscribe, File transfer, Event Emitter
Service Bus
REST/SOAP/web services
Database
MongoDB/Nosql
Perfect for data synchronization
Application server (Node.js) based Server-side JavaScript
MongoDB SSIS components (easily connect SQL Server with live MongoDB)
Http Request
Http Request
Patient 1
Patient 2
Patient 3
Patient 4

Figure 4 Design of the healthcare monitoring gateway features web services. It is based node.js, MongoDB, MongoDB SSIS components.

The healthcare web service gateway implements the Enterprise Service Bus (ESB) based data integration. The ESB facilitate for constructing a messaging backbone which does protocol conversion, message format. The service bus interacts with Node.js should help for sending/receive and publish/subscribe messaging over RabbitMQ. The MongoDB SSIS (SQL Server Integration Service) components allows to easily connect SQL server with live MongoDB document database. The connection between SQL server and MongoDB is made possible through workflows. When it comes to use MongoDB, Data flow Components are used to synchronize with MongoDB data. To help healthcare record network and private's physician pursue patient's data without compromising patient confidentiality, we recommend secure and encrypted storage. The healthcare hub server should envisages encrypted storage of protected health information (PHI). All services involved in the healthcare hub server should use secure gateways.

4.1 Experimentation results

The test plan use JMeter[11] to capture throughput, response time results of Node.Js's single-thread event-loop against the traditional application based java on the server-side JavaScript interpreter using the Apache sling. The capacity and performance testing is required to show that a healthcare hub minitoring server which consists of backend server and database layers can run with acceptable responsiveness when a large number of concurrent users can access these backend server-side and database simultaneously.

For each of the Client-healthcare hub Server Architecture, the goal is not to test the web application but to listen to the http request sent from the mobile device in the same way an http request is submitted from a web browser. The Application under test is Instant Heart Rate[12]. It uses smartphone built-in camera to track color changes on the fingertip that are directly linked to your pulse. This android 's application is based on the mobile client server computing that has a module of uploading the accurate rate data stored on a temporary data store like SQLite[13] to the remote healthcare monitorting application server. The test environment for the first Client- healthcare hub server Architecture consists of logic tier that determines how data can be created , displayed, stored ,changed and data tier as shown on the Table 1.

Table 1 implantation's components for healthcare hub server

Open platform under test for server side application	Logic tier	Data tier
Node.js	Express.js[14]	MongoDB[14]
Apache Sling[15]	Java based server javaScript /Java Content Repository	MongoDB

The first tests against Node.js and the Apache sling simultaneously from 200 users up to 2000 users, 50 times. For each test, the "ramp-up" period is zero (0) which mean that all the users start sending the http request simultaneously.

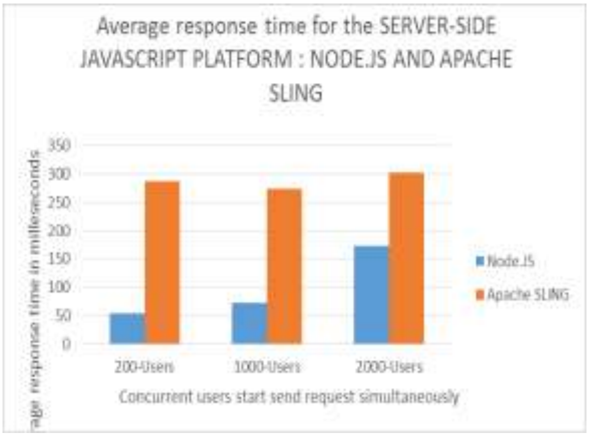
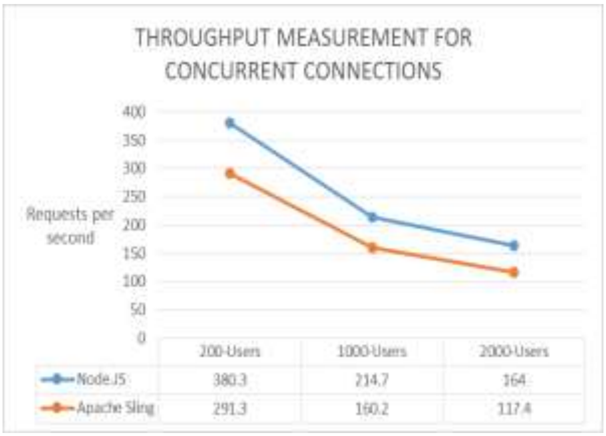


Figure. 5. Measurement of average response time



	200-Users	1000-Users	2000-Users
Node.JS	380.3	214.7	164
Apache Sling	291.3	160.2	117.4

Figure. 6. The measurement of throughput for Node.js against Apache sling

What we can see on Fig.4. is that the response time degrades as the number concurrent requests increases. For example, Node.JS-MongoDB was within response time of 54ms on average at 200 concurrent users, and 173ms on average at 2000 concurrent requests. We can see that for Apache-sling at the server side, the average response time has an almost linear correlation to the number of concurrent requests. This means that a thousandfold increases in concurrent users starts to a hundredfold increase in response time. The Fig. 6. shows that the throughput deteriorates for the two models as the number of users increases. The results show that for Node.js associated to MongoDB, the throughput was 380 requests per second at 200 concurrent users, and 164 requests per second at 2000 concurrent users. The throughput of Node.js based event-driven approach using V8 runtime engine have performed better than Apache-sling which use rhino as server-side javaScript based Java. With Apache-sling-MongoDB configuration, the throughput increases up to 1000 concurrent users, then deteriorates as the number of concurrent users increases. This leads to formulate that the number of concurrent users carried out by an Apache-sling at server-side for building web services is not relatively constant. Therefore, Node.JS is roughly 40% faster, for example 164 request per second against 117 requests per seconds for 2000 users that corresponds to one hundred thousand (100000) concurrent requests.

5. Conclusion and future works

The healthcare web service gateway constitutes by Node.JS server javascript side and MongoDB is 40% faster that the Java based server side JavaScript solution using Apache sling with MongoDB database for implementing mobile client server computing related remote healthcare monitoring system. In this paper, the difference concurrency models between single-threaded event loop Node.js and multi-thread approach made difference. To test Node.js a higher concurrency level-where it is supposed to surpass multi-threading, other problems like increasing the number of requests occur. For future work, we will implement the RHM system using the node.js that collects patients' data in a real time, controlling treatment recommendations and intervention based on the analysis of collected data.

6. Acknowledgement

This research was supported by the Brain Busan 21 Project (2016), Nurimaru R&BD project (Busan IT Industry Promotion Agency) (2016), and Dong-Eui University Research Institute (2016).

This paper is a revised and expanded version of a paper entitled Performance Investigation of Event-Driven Server-Side for Internet of Things based Remote Monitoring Applications presented at 2nd International Integrated (web & Offline) Conference & Concert on Convergence, Saint Petersburg State University, August 7-14, 2016.

References

[1] Yvette EG, Ha Jin H, Haeng Kon K. Internet of Things (IoT) Framework for u-healthcare System. International Journal of Smart Home 2015; Vol.9, No.11; pp.323-330.
[2]. Niewolny D, White paper: How the Internet of Things Is Revolutionizing Healthcare. Freescale Semiconductor, Inc., Reg. U.S.; 2013.p.1-8.
[3]. Kang J, Yoo S and Oh D. Development of a Portable Embedded Patient Monitoring System.International Journal of Multimedia and Ubiquitous Engineering (IJMUE). 2013; Vol.8, No.6; pp.141-1504.
[4]Kalin M. Three Ways to web Server Concurrency. Linuxjournal web site.2012, p1-2, Available at http://www.linuxjournal.com/content/three-ways-web-server-concurrency(last access April 2016)
[5] PetitIV R.G, Gomaa Hassan. Analyzing Behavior of Concurrent Software Designs for Embedded Systems. Proceedings of the 10 th IEEE International Symposium on Object and Component-Oriented Real-time Distributed Computing.2007; p1-9
[6] ObjectTime Limited. Designing for Concurrency.Canada; 1998, pp1-28
[7] Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. In: Internet Computing, IEEE. 2010;Vol.14.No.6; p.80-83.
[8] Yuhao Z,Daniel R,Matthew H,Vijay,JR, Microarchitectural implications of event-driven server-side web applications.In: Proceedings of the 48th International Symposium on Microarchitecture;2015.pp. 762-774.
[9] David F.JavaScript: The Definitive Guide, sixth Edition. O'Reilly Media, Gravenstein North, Sabastopol. United States of America; 2011
[10] Artyom S. Continous Delivery of Apache Sling Applications. In: Ecolde Polytechnique Federale de Lausanne. Thesis presented for the Degree of Master of Science.Switzerland; 2014. Available on line at https://bdelacretaz.files.wordpress.com/2014/09/artyom-thesis.pdf.
[11] EmilyH,H., "Apache JMeter. A practical beginner's guide to automated testing and performance measurement for your websites,PACKT PUBLISHING,BIRMINGHAM-MUMBAI,2008, pp:1-138
[12] Shyam P. Building a Heart Rate monitor and dashboard for android, available on line at https://www.pubnub.com/blog/2015-09-17-tutorial-realtime-android-heart-rate-monitor-and-dashboard/
[13] Sunguk L. Creating and Using Databases for Android Applications. In: International Journal of Database Theory and Application. 2012, Vol.5, No.2;pp.99-106.
[14] Simon Holmes. Getting Mean With Mongo, Express, Angular and Node. MEAP Edition. Manning Early Access Program. Manning, 2015, p8-24.
[15].Apache Sling Architecture. https://sling.apache.org/documentation/the-sling-engine/architecture.html