Simplifying Web Application Development Using - Mean Stack Technologies

Bakwa D. Dunka[#1], Edim A. Emmanuel [*2], Dantala O. Oyerinde [@3]
[#]*Senior Programmer, ICT Directorate: Websites and e-Services, University of Jos, Plateau State, Nigeria.*
[*] *Senior Lecturer, Department of Computer Science, University of Calabar, Cross Rivers State, Nigeria.*
[@] *Lecturer, Department of Computer Science, University of Jos, Plateau State, Nigeria.*

Abstract: Traditional stacks have been the historical programming practice for most web applications developed around the globe, However, the trend is now changing. The advantages they provide can no longer meet the increasing need of the users. The focus of this study is to design and implement and email system as part of a telemedicine application using some emerging JavaScript technologies that include MongoDB, ExpressJS, Node.JS and Angular.JS. The benefits provided by these individual components are on the rise and can function as a single stack. This paper is relevant in studying the theoretical taxonomy, internal working principles and implementation of the M.E.A.N stack components. Bringing these components to work together through one common language of internal data communication (JSON) to build modern day web application is presented in this study. The test result shows that the technology provided a platform for easy user interactions and improved communication. The result of this study has proven the much-needed advantage of speed and flexibility of this technology when compared to other traditional stack such as LAMP or .NET.

Index Terms: MongoDB, Express.JS, Angular.JS, Node.JS, JSON, MEAN, REST API, mongoose, web development.

I. INTRODUCTION

M.E.A.N, simply referred to as “mean stack” or just “mean”, is a collective name for four pieces of software as presented by Haviv (2015) and Alfred (2014): MongoDB, ExpressJS, AngularJS, and Node.JS. Beyond the acronym however, JavaScript is the core of these four components and the back bone of most modern application development. Mean is a full stack, JavaScript technology that tends to accelerate the building process of web and mobile applications for modern developers. It is robust and can easily be maintained. It should however be noted that AngularJS can be replaced with any of the frontend engine of its kind such as EJS or REACT.JS. JADE is the default template engine for AngularJS in mean. Each component in MEAN stack speaks the language of JavaScript Object Notation (JSON). Thus, there is no translation needed from the client side, server side and JSON document Object stored. The benefits of same language across the entire stack is beneficial for time saving, increase in productivity (Chris, 2015), and less memory usage which is critical in every code execution process. It is also easy to update data speedily and simultaneously in MEAN because every component understands JSON including the MongoDB database which understands the binary translation of JSON i.e. BSON. With MongoDB, you can store and query the BSON without altering the structure and meaning of the information. One outstanding advantage of using MongoDB in this study is scalability. Fig. 1. describes JSON language across the M.E.A.N Stack. The Node Environment runs JavaScript to carry out operations and make request to the MongoDB database server; and the Web server (Node.JS) sends the generated HTML page to the frontend.

Frontend <--------->JSON<-----------> Backend <--------> JSON<-------->Database

AngularJS<--------->JSON<-------->Node.JS/Express.JS<------>BSON<------>MongoDB (NoSQL)

Fig. 1: JSON Across the mean stack.

The web is not a monolithic technology but rather a combination of web frameworks, programming languages, constructs and libraries. The experience of the pass 25 years revealed that the LAMP stack, the .NET stack and a rich variety of other frameworks and tools were used in developing various kinds of web applications, However, the main problem with these stacks is that each tier demands a knowledge base that usually exceeds the abilities of a single developer, this create a drawback such as forcing development teams to continue to expand more with different expertise. This usually leads to less productivity and exposure to unexpected risks. For example the Linux/Windows Apache MySQL and PHP (L/WAMP) stack is a registered stack consisting of either Linux or Windows Operating System (OS), a Web server (Apache), a database (RDBMS - MySQL), and the programming language for generating dynamic HTML web pages such as PHP, PERL or Python. However, these technologies put together as a single platform for web development were not originally designed to work together (David, 2002).

This study centers on Mean stack as a complete JavaScript technology stack that is built by different teams forming a substantial community of developers and advocates that supports the continuous development of each component to higher levels along with corresponding documentation (Onodi, 2016).

The technology consists of a No-SQL database, a web server framework, a web client framework, and a server platform. The strength of the Mean stack relies on a centralized used of JavaScript as the basic programming component (Elrom, 2016). MEAN stack is an open source collaboration tools that binds the development of the three-tier web MVC architecture and utilizes JavaScript as the programming language across the three layers. Node.JS performs the function of apache but with greater simplicity. Instead of deploying application to a standalone webserver, Node.JS is included in the application and run with the current version of the web server and with the help of some run time dependencies. Chris (2015) categorized the application into two parts, the client side and the server side. The client side consists of the frontend, which receives information from the server side. The server side model consists of Node.JS to provide the resources to the service requester.

In a nutshell, Node.JS has been continuously developed and improved upon. The server side platform allows developers to create their own web server as well as build fast, scalable applications with JavaScript as described by Perrenourd (2015). Node.JS runs on the V8 chrome engine, an open source JavaScript engine developed by Google Corporation. Node.JS is a hybrid of 40% JavaScript and 60% C++. Node.JS relies on event-driven architecture and non-blocking I/O model which helps to increase performance and optimizes throughput and scalability. This is of course in disparity to a thread-based concurrency model. Node.JS can manage several network connections simultaneously thereby making it the best choice for data intensive applications and development of real-time applications.

This paper presents a theoretical taxonomy and scenario for considering the implementation of the MEAN stack technology in the development of the frontend and backend of web applications. It also presents the benefits that developers have to gain when compared to other stacks such as the LAMP stack. In the study, an emailing application was built and deployed to the Heroku (PaaS) platform and the application demonstrates the strength of MEAN stack technology for development of fast, efficient and scalable web application.

II. LITERATURE REVIEW

The various building blocks and design components of MEAN stack, that are used in the design of web applications and other studies that have demonstrated the implementation of this technology are presented next.

I. MongoDB

Submit In 2009 Dwight Merriman and Eliot developed an open source V8-based database called MongoDB. MongoDB derived its name from humongous scalability. The NoSQL database is more scalable than other conventional databases and uses JSON-like data model called BSON with dynamic schemas. MongoDB gained a lot of attraction as a result of providing developers the needed flexibility when dealing with complex data and at the same time scaling queries much more than the Relational Database Management Systems (RDBMS) (Rick, 2010). MongoDB cuts down large swaths of transformation logic. It is a leading NoSQL document oriented database with persistence storage strategy. BSON is used as a medium to transfer data easily from database to server to client (Ihrig & Bretz, 2015). It also expresses its model within codes. MongoDB is meant for fast and optimized updating using aggressive caching writes technique and by overwriting updated records where possible (Dirolf, 2010). However, non-readability is its main drawback.

mongoLab (mlab) was used to expound the management of MongoDB databases on the cloud platform. This profound improvement has proven greater level of control more than the relational database management systems (RDBMS) with related impedance mismatch problems. MongoDB is widely used in the cloud environments with no native libraries or drivers and offers more benefits over traditional databases. MongoDB are schema-less and works with object representations as oppose to storing those object in several tables with complex representations or schema, MongoDB like most NoSQL Databases support clustering and scaling application load. Its REST-enabled interfaces over Hypertext Transfer Protocol (HTTP) supports high-availability of data. The Mongoose model serves as a connection between MongoDB and NodeJS, this provides the required data structure and at the same time maintain flexibility in data storage and retrieval. Mongoose can be described as an Object Data Modeling (ODM) library for MongoDB.

II. ExpressJS

ExpressJS is a being a lightweight web application framework that helps developers in organizing web applications in MVC architecture on the server side. It manages everything from routing to handling requests and views. Express.JS makes it easier to write secure, modular and fast Node.JS applications with higher efficiency, reliability and less duplications. Express.JS has memory storage for sessions and organizes web applications by hiding most of the inner workings details of Node.JS. Its features include: modular HTML template engines, extending the response object to support various data format outputs, routing system, improved project structure, proper configuration of applications and breaking its logic into different modules. This framework was developed by T.J Holowaychuk within the JavaScript ecosystem to preserve node.JS and to facilitate building faster Single Page Applications (SPA).

III. Angular.JS

Angular.JS or just Angular is a frontend framework used in writing single page applications (SPA) with single page loading. It continued to streamline from several page loading and the use of Ajax call methods. Two-way data binding between views and models are made possible with Angular. This is useful for developers in terms of timelines, and placing tasks on a. template is made much easier with HTML (Adam and Colin, 2014).

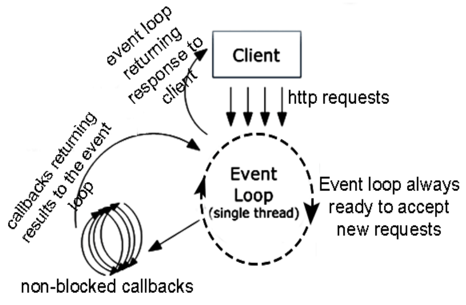

event loop returning response to client
Client
http requests
callbacks returning results to the event loop
Event Loop (single thread)
Event loop always ready to accept new requests
non-blocked callbacks

Fig. 3. Non -blocking capability of Node.JS adapted from Mathieu (2017).

IV. Node.JS

Node.JS is not a framework, rather, it is a very low level program just like C programming language that is used in writing client side and server applications that requires speedy and efficient delivery such as real-time and server applications. Node.JS uses Google Chrome’s super-fast highly optimized V8 execution engine in JIT (Just in Time) compilation fashion to execute JS code by transforming them into machine language and optimizes through complicated methods such as code inlining, copy elision etc. Some characteristics of node.JS include: single thread, exhibiting parallel execution of codes without interference of deadlock, handling concurrent request up to ten thousand connections and more, Node.js exhibits non-blocking to optimizes throughputs and has a main loop that listens to events, which then triggers a callback function (Kenneth, 2015). Its code algorithm is considered crucial to prevent users from wasting valuable time by allowing users do other task while waiting for longer tasks to complete execution. NodeJS maintain a great community and rapid growing history. It has rich libraries contained in its Node Package Manager (NPM) for sharing and publishing of open-source resources by the community (Fhala & Chrispinus, 2017). The diagram (Fig. 2) highlights two important capabilities of the development component. Node.JS is single threaded, asynchronous and uses event-driven architecture to run the server. This attribute is one of the main reasons that make Node.JS perform extremely fast, accurate and efficient in resource usage. Other well-known web servers such as Apache, Ngnix and IIS, uses the blocking I/O model. This model handle requests and uses multi threads to avoid concurrency issues. Therefore when a new connection is established, it will be served with a separate thread associated with an amount of RAM. Blocking I/O increases complexity and overhead for the server as it causes threads to wait for I/O while the server processes and retrieves the data.

Leveraging on the power of JavaScript, Node uses a single execution thread and asynchronous I/O to handle all the requests. Because of asynchronous or non-blocking I/O architecture, there is no delay for I/O or related context switching processes. Instead of giving each connection its own thread, Node ties all the connections to the same thread and maintains an event loop. The loop looks for new events and then delegates them to certain handler functions, While

NodeJS continues running other processes and invokes these handler functions to complete the execution of these other events.

During this study, an application was developed and in it, Express.JS was used to receive request via socket.IO when connected through socket.IO. This means the users will open their browser where the app is located such as

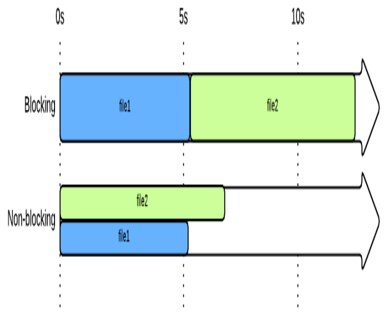


Fig. 2: Event Loop capability of Node.JS (Paul, 2014).

http://localhost:3000, then it will send and load the *index.jade*. In this file, a JavaScript code connects to the server, using HTTP and listen through socket.IO or WebSockets in general. The client carries out 2 types of connections: a connection to the HTTP server to load the default User Interface (UI) page and a real time connection to open a tunnel where the socket.IO communicates bilaterally to the application. The *httpServer* module in the application loads the http server API. Once loaded, an http server is created to listens to port 3000. The function of *http.createServer(request, response)* is to emit an event and the event is trigged whenever there is a request. Since, node.JS is asynchronous, this means that it cannot return the result of the request unless a function is provided when the request arrives. In this case the response is a JADE file embedded within app.js with its content. Observe that an *httpServer* module is created by simply adding the code. *exports.Function = function*. This will allow the user to access the module in Node.JS and keeps the code clean.

V.Representational State Transfer (REST) API

REST is an architectural style which offers good representation, visibility, reliability, scalability and performance through the following: 1) The use of Hypertext Transfer Protocol (HTTP) method 2) Being stateless 3) Exposing directory structure-like URLs and 4) Transfer of data through XML or JavaScript Object Notation (JSON) (Dickey , 2015).

The use of HTTP method means the use of HTTP verbs such as POST, GET, PUT to create, retrieve, update or change and delete data. http://hrtc.herokuapp.com/newuser.js?newuser=bob is an HTTP GET request.
HTTP being Stateless indicate that it does not support storing state information on the server rather it saves state information on the client side by using cookies. It will help reduce the load of manipulating the state of elements in the front end without putting so much burden on the server an also expose the directory structure-like URIs as illustrated below. Instead of intricate HTTP request such as: *http://hrtc.com/getfile.php?type=x&y=items&pid=20*, HTTP uses *http://hrtc.com/files/y/items/20* to describe the URL path. Also, in working with the backend to send JSON objects, developers could manipulate these objects in the presentation layer without hitting the servers, unless new data is required. RESTful service has been in existence for more than fifteen years even though it was not completely utilized until the last few years and seems like it is just as emerging.

One characteristics of a RESTful resource is HTTP whose specification must be recognized by a valid URI distinctively. For example, following links to navigate from one service to the other, the links that provides REST support not just for XML but also for JSON, this gives it an edge over the well-known SOAP web services which has a lot of functionality attachments and metaphor such as WSDL, UDDI, WSIL. These mechanisms perform the function of describing interfaces attributes such as: security, policies, endpoints location, descriptions and aggregation of services which works fine (Bojinov, 2015). However, description of service for RESTful service is optional; utilizing the service has nothing to do with its description and may not necessarily require such needed descriptions. Since REST uses standard HTTP it has the advantages of being much simpler. REST offers better performance and scalability. Corporations such as Yahoo, Amazon, Flickr and eBay all use REST support. SOAP was formerly used by most services such as Google. Today, REST dominates their web services architecture.

VI. JavaScript Object Notation

JSON is used to transmit serialized data over a network connection, such as the interchange of data between a web server and a web application. Serialization means transforming objects or data structures in a format that is well suited for storage in the memory buffer or file, and then transmit over a network connection (W3resource, 2015). This data can be retrieved any time. JSON stores semi-structured data due to its nature. It is also lightweight and open standard data interchange format that is both human and machine-readable. JSON has wide acceptance from the community of developer. It was tagged on RFC4627 on IETF specified standard. JSON files is stored with the. json extension.

VII. Benefits of Using MEAN Stack in building Web Applications

So many reasons has made MEAN to have an edge in developing web application. They include: 1) Unlike relational Database management system (RDBMS), MongoDB is a NoSQL BSON styled document oriented database which requires no conversion or relational models, it offers greater flexibility with accommodating layer for storing data dynamically because it was built for the cloud (Grover, 2015). It eliminates impedance mismatch problem encountered with the rigid structure of RDBMS (Peter, 2017) and has incredible performance implications when running queries. 2) MEAN works with JavaScript technologies and provides complete frontend development with less hassle, unlike the frontend of LAMP that can work with many languages and components than the backend. 3) Node.JS is a server that runs the MEAN application. It is an event-driven I/O server-side JavaScript environment that performs more than Apache, IIS and Ngnix.in terms of data-heavy and real-time task. 4) LAMP stack and related stacks restrict the operating system to a variant of its operating system. The MEAN stack has no such restrictions. MEAN allows any operating system which can run node.JS 5) Mean is entirely open source and has great community support 6) MEAN uses JSON as data interchange format, and using the same language all through this stack makes it consistent.

VIII. The Downside of using LAMP

The LAMP stack has produced very useful web applications for more than 25 years. However, LAMP stack has its shortcoming; Today's technology has provided faster and more robust alternatives to web server than Apache. Resource requirements such as memory and CPU consumption increases while serving lots of web page requests concurrently. This is due to the need to spawn new processes needing new threads that must compete for access to such memory and CPU. It will also refuse new connections when traffic reaches the limit of processes set by the administrator (Walker, 2014). MySQL has problem with handling recursive functions. Hence, there is the need to be connected with more recent technology. Also, writing secured readable, reusable PHP code can sometimes be challenging.

III. RESEARCH METHODOLOGY

This study was carried out using the agile method. That include the following processes: planning/listening, design, coding, testing and implementation. These processes are briefly discussed next.

Planning / Listening – This phase was conducted in order to determine the basic high level requirements needed for the design of the internal emailing as part of a caregiver-patient telemedicine application. User priorities were considered and refinement were made to come up with the user specifications.

Design – In the design phase, suitable designs were created with simple MVC architecture for the application. The process of refactoring was applied in the design phase until a desirable design was achieved. The specifications were streamlined based on the use of Node.JS for the server side run-time environment, Express.JS and Node.JS framework for the server-side, MongoDB for the database and JADE template engine for the frontend design since they are fast and supports JSON. The MVC modular design was also implemented.

Coding – The coding phase was crucial in building the application based on the M.E.A.N stack RESTive API. The advantages of the coding style include the speed, scalability and simplicity of coding.

Testing – The process of testing was carried out at various stages during design and implementation, to ascertain that the developed application conforms to the requirements that were set during the planning phase.

Implementation – User evaluation was carried out to measure the system performance with respect to its aim and objectives.

IX. The System Architecture

The system architecture for the MEAN stack presents the frontend, backend and data store. In Fig. 4, Users interact with the application through its user interface, developed using JADE view engine. Other alternative to JADE could be EJS or Angular.JS etc. The restful methods such as get(), put(), post(), delete() were used for communication between Express.JS routing API and the application. All input and output are transmitted in the JSON data format in all phases. Node.JS & Express.JS are part of the backend; they contain all the logic for different models and controllers for the emailing system. MongoDB uses the object relation mapper (ORM) to collect or interchange whatever data are needed from the data store.

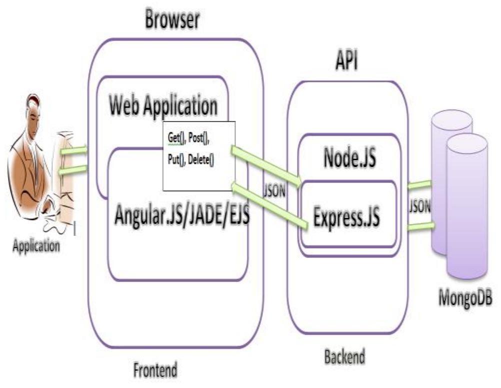


Fig. 4. Structure of the MEAN stack application.

X.System description

The emailing application was built from scratch using Mean stack. The system demonstrated the involvement of each component in the development process. Express.JS was particularly very helpful as it sets the foundation for the development. The development components were set-up in a logical MVC style before actual coding commenced. The process of re-factoring was applied in this research and performance functionality tests were carried during development using blazemeter cloud platform to determine the system performance based on load testing of the system. The MEAN architectural patterns for building the web application is based on the MVC pattern. A Model is seen as data or information store, the View as the layout of the user interfaces that renders data, and the Controller as logic that handles the control flow including any business logic needed to build the Model and pass the model data and the view for presentation to its end user (Edim and Bakwa, 2017). In the design of the web application, a route component was added between the controllers and the mandated user's browsers in order to coordinates the interactions with the controller. One common strategy was for us to have a RESTful interface which feeds a single page application (SPA); this was implemented through a REST API. In the application design, a HTTP request was routed to the appropriate Controller Action based on the URL. The Controller Action will process information then submit it in the request, and then returns the appropriate Model and View for rendering to the end user. MVC is described as a software architectural pattern for implementing the UI. It has three interconnected parts which separates the internal representations of information. The architecture of this system consists of data, logic and presentation layers similar to the database, server and client used in conventional application development. The architectural implementation pattern will handle the logic, visualization and data as presented in Fig. 5.

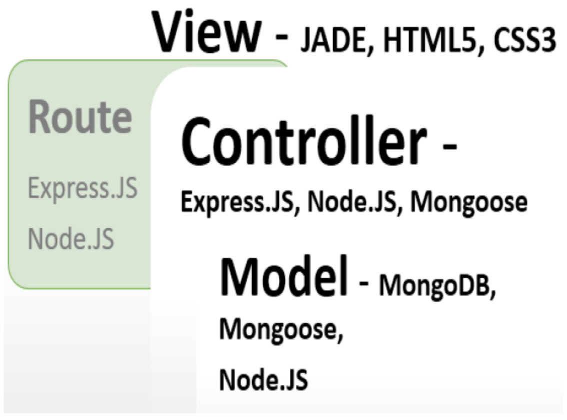


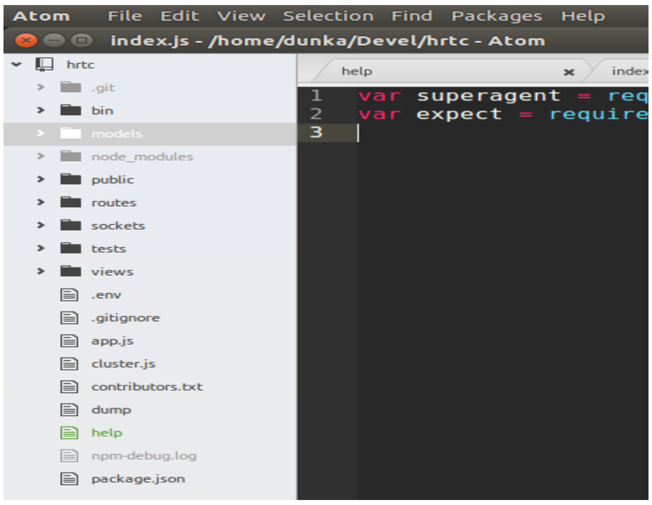
Fig.5. M.E.AN MVC Architectural pattern (Edim and Bakwa, 2017).

XI. MongoDB, ExpressJS, AngularJS, NodeJS (MEAN Stack) Implementation.

The installation of Node.JS and Express.JS was particularly important and set the foundation for the application. After setting up the development environment, the following commands were used on Ubuntu version 16.04 command line interface (CLI) for the task of installing Express.JS

```
npm update -g express
npm update -g express-generator
express hrtc
cd hrtc
npm install
```

The commands exemplify the task of installing and updating express.JS and its scaffolding generator in a global manner. This command is instrumental in creating the project directories including basic dependencies as shown in Fig. 6.


Atom File Edit View Selection Find Packages Help
index.js - /home/dunka/Devel/hrtc - Atom
help
index

```
hrtc
  .git
  bin
  models
  node_modules
  public
  routes
  sockets
  tests
  views
  .env
  .gitignore
  app.js
  cluster.js
  contributors.txt
  dump
  help
  npm-debug.log
  package.json
```

```
var superagent = req
var expect = require
```

Fig. 6. The directory structure of the system

The files and directories to the left of the diagram (Fig.6) are contained within a main directory, “hrtc”. This directory contains other sub-directories which include the model, views, routes, public, sockets, test, node_modules, app.js file, package.json file, cluster.js file, dump file and npm-debug.log files. The server-side consists of Node.JS, MongoDB and Express.JS. The initial creation of the server configures the application models, application routing and socket events relating to the application. The implementation of these significant components is described next.

XII. 1) app.js Start-up file

The *app.js* file was used to initialize or start-up the application since it glues every piece of the application together. *app.js* serves as the starting point of the application. Applications normally listen to request through a port. app.js initiates the port to which the application listens to, and also define Node.JS signaling server. It also contains dependencies that define handler functions for different RESTful paths and activities.

XIII. 2) Package.json

This file is used to store and remember the application’s internal dependencies and their respective versions. The implementation of package.json was achieved by executing the command “*npm install –d*”. This command installs all the dependencies listed in the Node Package Manager (NPM) as follows

```
{
"name": "my app",
"version": "0.0.0",
"private": true,
"scripts": {
"start": "node ./bin/www",
"test": "mocha ./tests/index.js"
},
"dependencies": {
"async": "^0.9.0",
"bcrypt": "^0.8.1",
"body-parser": "~1.12.0",
"colors": "^1.0.3",
"connect-redis": "^2.2.0",
"cookie-parser": "~1.3.4",
"csurf": "^1.7.0",
"debug": "~2.1.1",
"mongoose": "3.8.24",
"mongoose-paginate": "^3.1.3",
"morgan": "~1.5.1",
"multer": "^0.1.8",
"nodemailer": "^1.3.2",
"postmark": "^1.0.0",
"redis": "^0.12.1",
},
"devDependencies": {
"expect.js": "^0.3.1",
"superagent": "^0.21.0"
}
}
```

Routes handling

All generated routes are found in the "*route*" directory. The file *routes/index.js* contains the logic for a request against the main homepage. The routes/index.js file loads the user models. The *express.Router()* function is used to define groups of routes using *app.use* function. *express.Router* () function is a route used in processing requests. This validates parameters using *.param()* and use *app.route()* to access the routes and define multiple requests on a route.

Implementing *route* path for user define middleware

The Express method is called using; app = express(). Exress.JS uses the app object to perform different operations and services. For example listening to a socket connection on a path or on a specific host and port. Example of functions used by Express include app.get(), app denotes the express application object while get() is the method or function. It helps to start the request and response cycle of the appropriate middleware. The app.render() is used to render HTML view files using a call back. Express uses template view engines to render views. Similarly, all user defined routes are implemented in the application in app.js as shown in the code snippet below: All the registration forms are designed the same way using JADE frontend view. The *route* function was created to handle HTTP POST requests contained in the app.js file.

```
var routes = require('./routes/index');
users = require('./routes/users'),
auth = require('./routes/auth'),
calendar = require('./routes/calendar'),
chat = require('./routes/chat'),
events = require('./routes/events'),
message = require('./routes/message')
```

Implementation of frontend view template engine

Java Agent Development Environment (JADE) was used in the implementation of the frontend. It is used in writing the logic used for creating rich Single Page Applications (SPA) interface. JADE is responsible for the frontend development. It accesses all the data it needs through the Node.JS API. Node.JS hits MongoDB and returns JSON information to JADE based on the RESTful routing by express.JS. The API can extend more routes and functions to JADE frontend application.

XIV. Implementation of public and model directories

The public directory holds static files. It contains Cascading Style Sheets (CSS), img files and libs files. These files correspond to CSS styling, images files and JavaScript libraries files such as jQuery. The name also alludes to anything that the client uses which the server makes publicly available.

Implementation of data store models

The model directory contains all the ORM models or Schemas in mongoose. The first step in making a persistent data store is to configure the data models. To do this, a schema layer was added on top of MongoDB using a library called Mongoose. The models of the application were effectively implemented as a directory that enables data storage and exchange in JSON format. The models were built as separate files in the models directory. Thus, whenever there is the need to add a new model, it is added to this directory and the path is defined in the app.js file. Next the model is registered with the global mongoose object imported using the *require* function. This is needed so that it can interact with the database anywhere else. Mongoose is imported in app.js, relationships were created between different data models using the *ObjectId* type whose data type is a 12 byte *ObjectId*. This *ObjectId* is actually stored in MongoDB using binary value to uniquely identify the document in the collection. The ref property used in the entire schema tells Mongoose what type of objects the ID references and enables the application to retrieve both items simultaneously. The database model is described for one of the application modules (appointment module) and describe how it is embedded within the body of the codes snippet shown below;

```
var mongoose = require('mongoose'),
Schema = mongoose.Schema,
ObjectId = Schema.Types.ObjectId,
paginate = require('mongoose-paginate');
var AppointmentSchema = new Schema({
dateCreated: {type: Date, default: Date.now},
sender: {type: ObjectId, ref: 'User'},
receiver: {type: ObjectId, ref: 'User'},
…
var Appointment = mongoose.model('Appointment', AppointmentSchema);
module.exports = Appointment;
```

The code snippet describes the implementation of Mongoose connection and the use of *ObjectId* in *model/Appointment.js* file. MongoDB has a flexible schema that allows for variability between different documents in the same collection. That flexibility can be very powerful as databases grows over time. This is referred to as scalability. But that flexibility does not mean that safety checks should be ignored. A user is modeled by specifying the properties in a schema as shown below;

```
var uSchema = new mongoose.Schema ({name: {firstname: String, lastname: { type: String, trim: true }
},stud_age: { type: Number, min: 0  }});
```

XV. Node.JS internal implementation

Due to the blocking nature of I/O encountered with database queries, couple with connectivity issues and many scenarios in program execution associated with most programming languages, Node.JS was used to boycott these issues. Node.JS was designed to continually execute codes while responses are being awaited. Instructions keep running in an infinite while loop making it non-blocking or asynchronous. Node.JS is being improved for maximizing throughput and enhancing efficiency especially of real-time applications due to its asynchronous nature. The internal working principle of the application runs on a single thread as presented in Fig. 7.

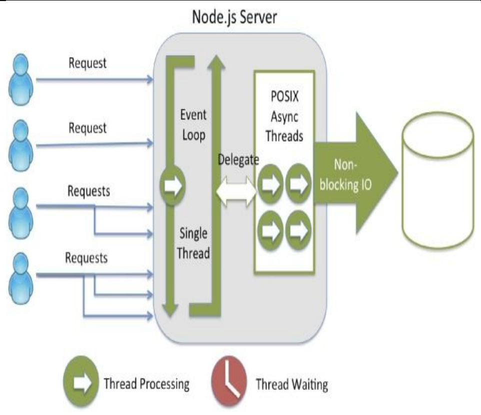


Fig. 7. Node.js single threaded - Non-blocking requests (Roth, 2014)

Unlike Java, PHP and other programming languages, Node.JS runs asynchronously and non-blocking rather than running on multiple threads. In Fig. 7, Request are handled in an asynchronous threaded fashion after the requests are delegated from the event loop unlike the server handling the requests on multi-threaded servers. With this concept, high performance is expected. The major problem of the general architecture of web applications is the maximum number of concurrent I/O connections that the server can handle and be stable. Node.js solves this problem by changing how a connection is made to the server. Node.JS solves the problem by creating a process that does not need another memory block to complement it instead of starting a new thread for each connection. It does not block outgoing calls directly to I/O. Node.JS servers can comfortably support tens of thousands of simultaneous connections because it does not allocate one thread - one connection model, but uses a model process per connection, creating only the memory that is required for each connection.

XVI. Connection to MongoDB

The Application database was implemented by installing the Node Package Manager (NPM) package driver for MongoDB called Mongooses. Mongoose is an Object Data Modelling (ODM) library that enables connection to be made to MongoDB and also provide a convenient modeling environment for mongoDB. Mongooses provide a fundamental connectivity and data manipulation logic for easy creation of schemas and simplifying Node.JS callback. The connection with MongoDB instance using Mongoose was fairly easy and requires the resource URL of the database only. The command *mongoose.connect* was used. This command helps to abstract certain complex processes before sending JSON data. The Node.js native driver was useful in handling the rest of the program in its callback.

XVII.Express.JS implementation

One of the basic components of MEAN stack is the ExpressJS framework. The functions offered by this framework are minimal but enormous and key in the central design of web application. ExpressJS performs both routing and middleware functions. ExpressJS has over 20 commonly used middleware, such as a logger, session support, cookie Parser among others. One major implementation of *app.use()* in this application is to plug-in to ExpressJS middleware by passing it as argument to the function. It will be responsible for Routing based on URL paths, managing sessions via cookies, Parsing incoming requests (example from JSON) to routes the request to a handler, otherwise known as the JavaScript callback function coded inline or in a separate module, (Madhanasekaran, 2015).

XVIII. Middleware Implementation

Being a middleware, its tasks have to do with having access to request object and response object. In implementing routes for Express.JS, two variables were passed to the handler function. *req*, which contains all the information about the request that was made to the server including data fields. *res*, will then respond to the client, as well as listening to the next middleware function denoted by the *next* variable. In this research middleware was used 1) as asynchronous functions that manipulate request and response objects using the function app.use() to pass the middleware intended 2) to make changes to the request (req) and the response (res) objects, 3) to end the request-response cycle and 4) to call the next middleware function in the stack using the *next* function. Applications developed using Express.JS can be embedded with different forms of middleware (Fig. 8) which include built in middleware, user defined middleware, application level middleware and error handling middleware.

One implementation of middleware shows that every HTTP request goes through a stack of middleware loaded through app.use() before a response is returned through one of the route handler. Before sending a request to the route that is being requested, it checks to see if the user has gone through the necessary checks to match the HTTP request for Cross-Origin Resource Sharing (CORS), Cross-Site Request Forgery (CSRF) or authentication middleware and any user defined middleware before sending them to HTTP responds that matches the route being requested by the user.

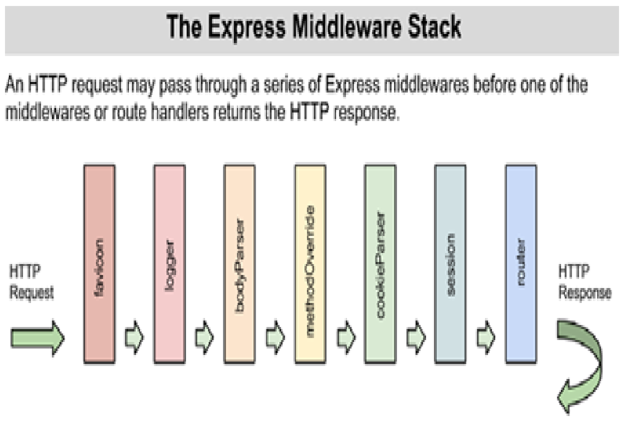

The Express Middleware Stack
An HTTP request may pass through a series of Express middlewares before one of the middlewares or route handlers returns the HTTP response.
HTTP Request
favicon
logger
bodyParser
methodOverride
cookieParser
session
router
HTTP Response

Fig. 8. Middleware stack (Madhanasekaran, 2015)

XIX. Implementation of require built-in middleware

The *require* function was implemented as a built-in Node.JS function to imports into another file or module. The path controllers cannot work without the *require* function. An implementation of *required()* function used in the app.js file are shown in the codes snippet below.

```
var express = require('express');
var path = require('path');
var socket = require('socket.io');
var mongoose = require('mongoose')
var logger = require('morgan');
var favicon = require('serve-favicon');
var bodyParser = require('body-parser');
var cookieParser = require('cookie-parser');
var redisStore = require('connect-redis')(session);
```

All the functions invoked by Express.JS routing layer before final request handler is made is described as a middleware. For example, in the code snippet above, the Serve favicon is a middleware for serving a favicon (icon before the URL). It takes care of cache control on the *public/*directory, this is an advantage over a normal favicon icon. *Morgan* is useful for logging requests and responses; *Cookie-parser* is a middleware for Express.JS that supports handling cookies. The request object will have a cookie object that can access the application. *Body-parser* is an Express.JS middleware that adds a body object to the request so that all POST parameter can be accessed. The *bodyParser.json()* gives the application the ability to parse JSON. This is important for JSON transfers, bodyParser.urlencoded({ extended: false }) allows the application to read data from URIs GET requests. It works with the query string module. The cookieParser() predominantly add the cookie object to every requests. The express.static(path.join(__dirname, 'public') tells the application to use the /public directory where images are stored and style sheets and scripts are used.
Express.JS has an environment-driven configuration, consisting of five functions which are driven by the NODE_ENV - environment variable; 1) app.configure(), 2) app.set(), 3) app.get(), 4) app.enable(), 5) app.disable() to define certain application-level settings that will enable the customization of the web application behavior such as enabling or disabling features.

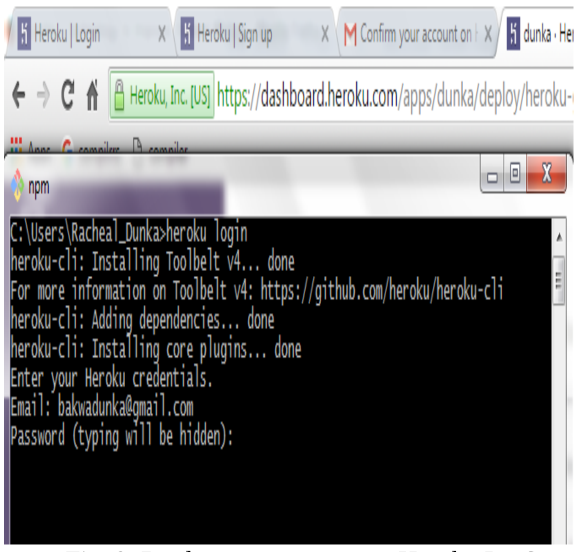
XX. Application Setup

This study makes use of Linux open source Operating System because it is widely used for development and production. The required tests were run on Ubuntu 16.04 Linux Machine. The MEAN stack application (email) was ran while performance was measured on real-time.

XXI. Deployment process on Heroku

One objective of this study is to deploy the application unto a flexible and easy to use platform, The Heroku platform as a service (PaaS) was used because it allows developers to focus on building, running, scaling application, storage, network management features. It helps reduce cost, better access to developer infrastructure and support for Quality of service. The following benefits were considered in choosing Heroku;

- Platform as a service (PaaS): In order to allow developers concentrate on the sole responsibility of development rather than concentrating on both development and hosing services, Heroku has emerged as a new technology to cater for every developer infrastructure such as hardware and software needs.
- Encouragement of developers driven deployment: Heroku uses tools that the developers are most likely aware of. A good programmer must be conversant with command line Interface (CLI), Integrated Development Environment (IDE). A similar environment is maintained by using Git CLI for passing commands for development, manipulating, monitoring and hosting purposes.
- Effortless scaling: Heroku allow developer to quickly scale application dynamically through their interface matrix system or through passing commands.
- Time and Cost: The provision of the hosting infrastructure has dramatically reduce the time and cost for developers.
- Multi-platform support: Heroku supports various platforms and has less to do with incompatibility issues. It means application developed with various languages can enjoy the same services as one developed using Node.JS
- Revision Version Control: One major function of Heroku is version control. It has forward and backward compatibility with various versions of the application. Commands were used on Heroku CLI to deploy the application on Heroku PaaS as illustrated in Fig. 9.



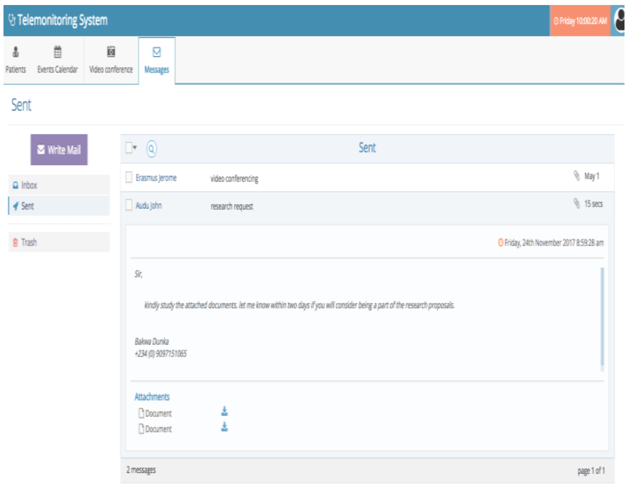
```
C:\Users\Racheal_Dunka>heroku login
heroku-cli: Installing Toolbelt v4... done
For more information on Toolbelt v4: https://github.com/heroku/heroku-cli
heroku-cli: Adding dependencies... done
heroku-cli: Installing core plugins... done
Enter your Heroku credentials.
Email: bakwadunka@gmail.com
Password (typing will be hidden):
```

Fig. 9. Deployment process on Heroku PaaS

IV. RESULTS AND DISCUSSIONS

XXII. User Interface Screen shots

This section gives an overview of the result and discussion of this study. The screen shots are obtained from the conversation between caregivers and patient sending and receiving emails. The system allows users access based on individual login credentials. Patients would be able to send medical information or data for analysis or treatment then treatment recommendations. The inbox has the following: write mails, Inbox, Sent and trash as seen with the normal e-mail. There is a provision to search for mails, delete mails, sort read and unread mails and view acceptance request. It has a nicely collapse and shrink for user convenience. Fig.10. Shows the user interfaces screen shot for e-mails. The advantages with this e-mail system is that it deviates from the usual design of polling for updates and page reloading thereby making the applications slow and resource intensive. This research addresses the need for a better design of the telemedicine system using recent technologies. This study is also an attempt to address most of the challenges and hence it will increase the overall interactions between caregivers and patients and the quality of care of patients. This new concept reveals the limitations which were observed with the traditional stack, while also improving the speed and scalability of applications. Therefore, the viability of integrating the MEAN stack components to render doctors-patients e-mailing system is a welcome development in the provision of healthcare.



Erasmus Jerome	video conferencing	May 1
Audu John	research request	15 secs

Friday, 24th November 2017 8:59:28 am

Sir,

kindly study the attached documents. let me know within two days if you will consider being a part of the research proposals.

Bakwa Dunka
+234 (0) 9097151065

Attachments
Document
Document

2 messages page 1 of 1

Fig. 10. User interfaces screen shot for e-mails

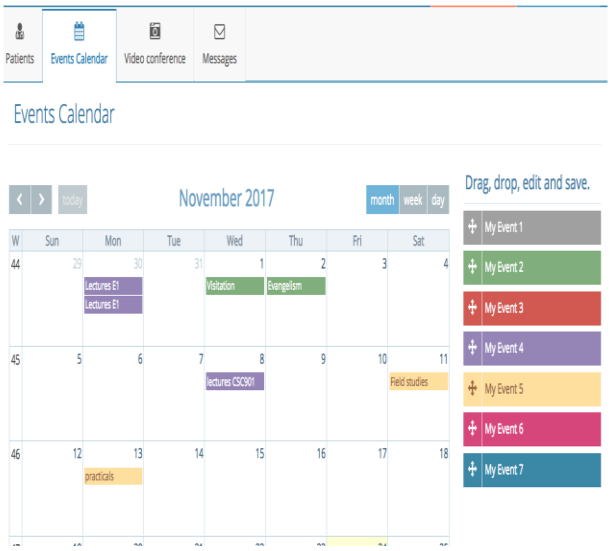
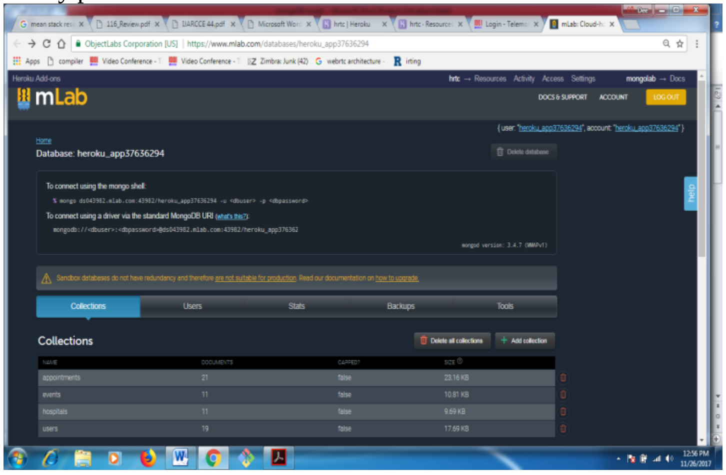


Fig. 11. User Interface screen shot for Event Scheduling

Fig. 11, Shows the user interface screen shot for scheduling personal events. This can be monthly, weekly or daily. It was implemented by using AngularJS and bootstrap. With the event scheduler, users can schedule their own events, manage their time more adequately and improve their productivity during office hours. The technology used for its implementation provided the much needed flexibility and speed of interactions. It also created room for more features or functions to be included in the scheduler and made it more interactive and intuitive. The events scheduler is easy to interact with and navigation from one level or interaction to the other. it is simple and easy to understand. This has been made possible as a result of the MEAN stack technology used.

V. DATABASE RESULTS

The mlab platform helps to expound the management of the MongoDB databases on the cloud. Its function is to fetch the database results and present them nicely in a human readable format. It is subscribed from the heroku dashboard. Result of the email collection is presented in Fig. 12. It shows that data is stored using blocks of JSON and BSON as a JavaScript interchange format. Unlike relational databases tables are called collections while rows are called documents or objects. Every document starts with an _id and present data in the form of "name": "value" pairs, the objects in the hospital collection include "Name", "Address", and "Active". Values themselves could be arrays of other object or nested objects. MongoDB does not enforce updating schema like in RDBMS which may sometimes be difficult. The output shows that MongoDB allows us to store objects in a very rich and dynamic way. This makes the presentation and understanding of database queries very easy and the requires information is easily past to the user.



https://www.mlab.com/databases/heroku_app37636294

{ user: "heroku_app37636294", account: "heroku_app37636294" }

Database: heroku_app37636294

To connect using the mongo shell:

```
% mongo ds043982.mlab.com:43982/heroku_app37636294 -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds043982.mlab.com:43982/heroku_app376362
```

```
mongod version: 3.4.7 (MMAPv1)
```

Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.

Collections

NAME	DOCUMENTS	CAPPED?	SIZE
appointments	21	false	23.16 KB
events	11	false	10.81 KB
hospitals	11	false	9.69 KB
users	19	false	17.69 KB

Fig. 12. Database interface from mLab.

VI. DISCUSSION

Due to little implementation of MongoDB, ExpressJS, AngularJS, Node.JS (MEAN stack) emerging technologies functioning as a single stack, the relevance of this study becomes obvious when components work together through one common language for internal data communication. This will forestall the benefits of using same language across the entire stack. MEAN allows for cost effectiveness in terms of developers expertise, speed, flexibility, support for M-V-C, scalability, open source and cloud deployment support. These benefits are higher when compared to other technologies. Node.js allows developer to easily deploy the applications directly on the server without the need for an external standalone server. This reduces the overheads that come with the client server architecture.

In this study, the practical experience in the design of the emailing subsystem using MEAN stack technologies has been presented. The new concept reveals the variations which have been observed with the stack of most common web application technologies such as LAMP and older technologies. These components when compared to other choices offer more convenience for interaction. A secured emailing system was developed, with many of the benefits explained. The system allows access based on login credential. If login is successful, users are allowed to carry on their related operations. The login phase is a uniform interface that comes to life by login to the URL: hrtc.herokuapp.com. The applications run on Firefox, Opera and Google Chrome browsers. The user interface results of this study have shown that users are able to communicate with each other from any location conveniently. The application provided the users flexibility, easy interactions and the desired quality of service. The study has shown that applications developed with the MEAN stack technology are easy to develop, implement and highly interactive.

VII. CONCLUSION

LAMP and its varieties have been a historical technology that has given birth to rich and wonderful application across the globe, and this cannot be undermined. However, in today's technology, JavaScript is a leader in web application development. The MEAN stack, named after its constituents is a new cutting-edge and robust technology that will soon overturn the web development platforms. The decision of using any stack depends on the need and priority of the programmer, but the MEAN stack has proven to be a reliable choice for fast, scalable and real-time applications. Node.JS asynchronous non-blocking concept made it easy for handling concurrency. MongoDB document oriented No-SQL database has proven high performance, flexibility and scalability in emerging technologies with the use of the JSON-BSON data format. Express.JS was built on top of Node.JS to provide an easy to use framework for developers. AngularJS was developed by Google to promote the MVC framework for building SPA and quickly build beautiful interactive user interfaces.

VIII. REFERENCE

[1]. Alfred, A. (2014). *Node.js: Introducing the MEAN Stack*.
[2]. Adam, B. and Colin J., I. (2014). Full Stack JavaScript Development with MEAN. Retrieved on November 20, 2017 at http://pepa.holla.cz/wp-content/uploads/2016/11/mean1.pdf
[3]. Bretz, A., & Ihrig, C. J. (2015). *Full stack JavaScript development with MEAN*.
[4]. Burns, N., & Grove, S. K. (2009). *The practice of nursing research : appraisal, synthesis, and generation of evidence*. St. Louis, Mo: Saunders Elsevier.
[5]. BUECHELER., C. (2015). CREATING A SIMPLE RESTFUL WEB APP WITH NODE.JS, EXPRESS, AND MONGODB. RERIEVED FROM: HTTP://CWBUECHELER.COM/WEB/TUTORIALS/2014/RESTFUL-WEB-APP-NODE-EXPRESS-MONGODB/
[6]. Bojinov., V. (2015). Design and implement comprehensive RESTful solutions in Node.js.
[7]. Dickey, J. (2015). *Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node.js*. San Francisco, CA: Peachpit Press.
[8]. Dirolf, M. (2010). Binary JSON. Retrieved from: http://bsonspec.org
[9]. Edim, A., E. and Bakwa, D., D (2017). A Peer-To-Peer Architecture For Real-Time Communication Using Webrtc. Journal of Multidisciplinary Engineering Science Studies (JMESS) ISSN: 2458-925 3(*4*).
[10]. Elrom, E. (2016). AngularJS SEO. *Pro MEAN Stack Development*, 197-219. doi:10.1007/978-1-4842-2044-3_8
[11]. Elrom, E. (2016). CSS, Bootstrap, & Responsive Design. *Pro MEAN Stack Development*, 131-164. doi:10.1007/978-1-4842-2044-3_6
[12]. Fhala, B., & Chrispinus, E. O. (2017). *Learning path: MEAN : create MEAN stack apps*.
[13]. Grover, R. (2015). *Building Apps with MEAN Stack: The Benefits of the MEAN Stack*.
[14]. Haviv, A. Q., Mejia, A., & Onodi, R. (2016). *Web application development with MEAN: Unlock the power of the MEAN stack by creating attractive and real-world projects : a course in three modules*.
[15]. Ihrig. J. C., and Bretiz, A. (2015). Full Stack Javascript Development with MEAN.
[16]. Kent Beck. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co, Inc, Boston, MA, USA.
[17]. Madhanasekaran., p.(2015). Node, Express.js and Mongoose : Part-II. [weblog post]. Retrieved Dec, 30 from http://developeriq.in/articles/2015/feb/09/node-expressjs-and-mongoose-part-ii/
[18]. Mathieu., N.(2017).Ultra-fast applications using Node.js. [Weblogpost]. Retrieved on Dec, 2017 from https://openclassrooms.com/courses/ultra-fast-applications-using-node-js/node-js-what-is-it-for-exactly.
[19]. Onodi, R. (2016). *MEAN blueprints: Unlock the power of the MEAN stack by creating attractive and real-world projects*.
[20]. Paul, S. (2014). Node.js – reasons to use, pros and cons, best practices![Weblog post] Retrieved oct, 2017 from http://voidcanvas.com/describing-node-js/
[21]. Perrenourd, M. (2015). *Learning web development with the MEAN stack*.
[22]. Peter. W, (2017). LAMP diehards take note: The flexible simplicity of MongoDB, Express, Angular, and Node.js is no joke. [Weblog Post]. Retrieved from https://www.infoworld.com/article/2937159/javascript/mean-vs-lamp-for-your-next-programming -project.html on November 20, 2017
[23]. Rick, O. (2010). SQL or NoSQL? [Web log post]. Retrieved on Dec, 2017 from https://rickosborne.org/blog/2010/02/sql-or-nosql/
[24]. Roth., I.(2014). What Makes Node.js Faster Than Java? [web log post - Strongloop by IBM]. Retrieved on Dec, 2017 on https://strongloop.com/strongblog/node-js-is-faster-than-java/
[25]. Walker., R. (2014). Ngnix VS Apache. [Weblog post - Anturis Blog] . Retrievd from on Dec 30th on https://anturis.com/blog/nginx-vs-apache/