

5-2016

# MEAN vs. LAMP Stack

Arpana Karanjit

St. Cloud State University, [kaar1301@stcloudstate.edu](mailto:kaar1301@stcloudstate.edu)

Follow this and additional works at: [https://repository.stcloudstate.edu/csit\\_etds](https://repository.stcloudstate.edu/csit_etds)

---

## Recommended Citation

Karanjit, Arpana, "MEAN vs. LAMP Stack" (2016). *Culminating Projects in Computer Science and Information Technology*. 11.  
[https://repository.stcloudstate.edu/csit\\_etds/11](https://repository.stcloudstate.edu/csit_etds/11)

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of theRepository at St. Cloud State. For more information, please contact [rswexelbaum@stcloudstate.edu](mailto:rswexelbaum@stcloudstate.edu).

# **MEAN vs. LAMP Stack**

by

Arpana Karanjit

A Starred Paper

Submitted to the Graduate Faculty

of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Masters of Science in Computer Science

Starred Paper Committee:

Jie Hu Meichsner, Chairperson

Andrew A. Anda

Dennis Guster

## **Acknowledgement**

I would like to express my sincere gratitude to my advisor Professor Jie Hu Meichsner for the continuous support, motivation, enthusiasm, and immense knowledge that she provided for my Starred Paper. Her guidance helped me in all the time of research and writing of this Starred Paper. I could not have imagined having a better advisor and mentor for my research.

Besides my advisor, I would like to thank the rest of my thesis committee members: Professor Andrew A. Anda and Professor Dennis Guster for their encouragement, insightful comments, and suggestions.

Last but not the least; I would like to thank my family: my parents and my husband for their continuous support and guidance to always do the best.

## **Abstract**

JavaScript has always been the scripting language for client-side programming that runs in the browser. The most crucial part in a web development project is choosing the right combinations of front-end framework, back-end server, and database environment. The main intent of this paper is to show the strength of the JavaScript-based framework, the MEAN stack (M for MongoDB, E for Express.js also termed Express, A for AngularJS or Angular and N for Node.js or Node) for building web applications. We compare the MEAN stack with the popular framework, the LAMP stack (L for Linux, A for Apache, M for MySQL and P for PHP), with respect to their components, strength, and environment configuration. We develop two similar applications built by MEAN and LAMP. We compare and analyze their respective real-time scenarios, data structure flexibilities, web performance, scalability, performance enhancements, and we perform load and data transfer tests.

## Table of Contents

	Page
List of Tables .....	v
List of Figures .....	vi
Chapter 1 Primers .....	1
1. Introduction.....	1
1.1. MEAN Stack.....	1
1.1.1. MongoDB .....	3
1.1.1.1. Features of MongoDB .....	4
1.1.1.2. MongoDB in MEAN Stack.....	4
1.1.1.3. MongoDB shell.....	6
1.1.2. Express.....	7
1.1.2.1. Features of Express .....	8
1.1.3. AngularJS .....	10
1.1.3.1. Advantages of AngularJS .....	11
1.1.3.2. Features of AngularJS.....	11
1.1.4. Node.....	14
1.1.4.1. Why Node? .....	15
1.1.4.2. NPM and Node packages.....	16

	Page
1.1.4.3. Node Modules.....	17
Socket.io.....	18
1.2. LAMP Stack .....	21
1.2.2. Apache web server .....	22
1.2.3. MySQL Database System .....	22
1.2.4. PHP Scripting Language.....	24
1.3. Architectures .....	25
1.3.1. MEAN Architecture.....	26
1.3.2. LAMP Architecture .....	27
Chapter 2 MEAN vs. LAMP .....	28
Chapter 3 Application Development Environment Setup .....	35
3.1. MEAN Stack Application Development .....	35
3.2. LAMP Stack Application Development.....	44
Chapter 4 Case Study.....	48
4.1. Tests and Results .....	51
4.1.1. Real-Time Web Test for Chat System.....	51
4.1.2. Extensibility.....	54
4.1.3. Performance and Load Testing.....	57
Chapter 5 Limitations and future enhancement .....	61
Chapter 6 Conclusion.....	63
References .....	66
Appendix.....	68

	Page
Source Code: .....	68
Chat System with MEAN: .....	68
Chat System with LAMP: .....	72
Address Book System with MEAN: .....	82
Address Book Systemwith LAMP: .....	86

## List of Tables

	Page
Table 1 Conceptual Overview .....	10
Table 2 Table schema of AddressList before update in LAMP stack .....	54
Table 3 Table schema of updated AddressList in LAMP stack.....	55
Table 4 Table schema of AddressDetail in LAMP stack.....	56
Table 5 Response time in MEAN and LAMP .....	58
Table 6 Data transferred in MEAN and LAMP with different concurrency .....	60



## List of Figures

	Page
Figure 1 Sample of MongoDB document .....	5
Figure 2 Screenshot of Mongo shell .....	6
Figure 3 JavaScript program in Mongo shell.....	7
Figure 4 Defining middleware .....	8
Figure 5 Error handling as Text Format .....	9
Figure 6 Error handling as HTTP response status .....	9
Figure 7 Error handling as JSON Format .....	9
Figure 8 Sample code of two-way data binding .....	12
Figure 9 Two-way Data Binding .....	12
Figure 10 Scope instance .....	13
Figure 11 View-model or scope.....	14
Figure 12 Example to load various Node modules .....	18
Figure 13 Socket.io start and connection .....	20
Figure 14 Socket.io in client .....	20
Figure 15 Architecture of MEAN .....	26
Figure 16 LAMP Architecture .....	27
Figure 17 Multi-threading server .....	30
Figure 18 Node.js server .....	31
Figure 19 Including Express in MEAN Application .....	37

	Page
Figure 20 Testing server running at port 3000.....	37
Figure 21 Server running at port 3000.....	37
Figure 22 Template declaration .....	38
Figure 23 AngularJS setup.....	39
Figure 24 Route Management.....	40
Figure 25 Client code for Route Management.....	40
Figure 26 Insert query in MongoDB.....	42
Figure 27 To display data of Collection with find() .....	43
Figure 28 To display data of Collection with find().pretty() .....	43
Figure 29 To check phpinfo () .....	45
Figure 30 PHP Module information displayed in http://localhost/test.php .....	46
Figure 31 Screenshot of Chat System in LAMP.....	49
Figure 32 Screenshot of Chat System in MEAN .....	49
Figure 33 Screenshot of Address Book System in LAMP stack .....	50
Figure 34 Screenshot of Address Book System in MEAN stack.....	51
Figure 35 Client request to get real-time update in LAMP stack .....	53
Figure 36 GET method to load old messages in MEAN stack .....	53
Figure 37 POST method to send message in MEAN stack .....	53
Figure 38 Schema of collection before update in MEAN stack .....	55
Figure 39 Schema of MEAN Stack after update .....	56
Figure 40 Response Time Graph with concurrency and 10000 Requests .....	58
Figure 41 Data Transferred Graph.....	60

## Chapter 1 Primers

### 1. Introduction

The increasing demand for performance and scalability in web technologies has motivated the development of many new technologies. Use of appropriate combinations of technologies in applications is the most crucial component of application development. In web application development, to achieve the highest benefits, one needs to focus on the right choice of technologies to be used in client-side, server-side, and database.

Our main objective is to show the strength of the JavaScript framework MEAN for developing web applications. Initially, we focus on basic components of both the MEAN stack and the LAMP stack i.e. introduction to individual components of each stack and their working architecture. In Chapter 2, we motivate why one should use the MEAN stack, and we present criteria including real-time application development, data structure flexibility, scalability, and data transformation, for choosing MEAN over LAMP. In Chapter 4, we describe the benefits of the MEAN stack versus the LAMP stack by comparing different metrics (response time, data transfer, and scalability) of two sample applications: a chat system and an address book system. We then focus on limitations and future enhancements of the MEAN stack, followed by our conclusion.

#### 1.1. MEAN Stack

“MEAN stack pulls together some of the ‘best of breed’ modern web technologies into a very powerful and flexible stack [3].” MongoDB is a [NoSQL](#) database system, Express is a web server framework for Node, AngularJS is a client-side scripting, and Node is a server platform.

These building blocks are developed by different teams and involve a substantial community of developers with proper documentation of each component.

The main strength of the MEAN stack lies in its centralization of JavaScript as the main programming language [1]. Each component of the MEAN stack is written in JavaScript, even the database stores data in JavaScript Object Notation (JSON) format which is the only script that JavaScript entirely understands [1]. So, JavaScript is not only used as client-side scripting language but also is used throughout the application i.e. in client-side, server-side and database [3]. Use of JavaScript as the main programming language both in client-side and server-side makes MEAN stack more powerful. It is beneficial to use the same language on the front-end and the back-end, as it makes the programming easier and reduces time consumed while building the application.

Using all JavaScript allows us to achieve following functionalities [4]:

1. Use JavaScript on the server-side (Node and Express).
2. Use JavaScript on the client-side (Angular).
3. Store JSON objects in MongoDB.
4. Use JSON objects to transfer data easily from database to server and to client.

In addition to providing a JavaScript based server platform, the MEAN stack provides the capability for the database to interpret JavaScript for data interchange (i.e. JSON). A single language across an entire stack increases productivity. Client-side developers who work in Angular can easily understand most of the code on the server-side. Server-side code turns out to

be more coherent to front-end developers and vice versa. This makes application development more straightforward and has appeared to decrease the development time.

The MEAN stack benefits from the qualities of Node, including non-blocking I/O, real-time application development, and server-side scripting. Applications which take the advantages of MEAN stack are chat system, real-time web applications like Facebook and Twitter which display instant status update without page refresh.

### **1.1.1. MongoDB**

MongoDB [5] is a powerful, adaptable, and scalable NoSQL document store model. In MongoDB, data is stored in the database in a JSON format named [BJSON](#), which stands for Binary JSON, rather than rows and columns like in relational database. MongoDB has a capacity to scale with various features that relational database provides like secondary indexes, sorting, and queries [6]. “It provides high performance, high availability, and automatic scaling [4].”

Since MongoDB is scalable, it is easy to implement and gives back-end storage for high traffic websites such as Facebook and Twitter where applications need to store huge comments and posts [6].

There are six major concepts of the MongoDB [5]:

1. Zero or more databases can be created within MongoDB instance.
2. A database can have zero or more collections, where a collection is similar to a table of a [RDBMS](#) (Relational Database Management System).
3. A collection can have zero or more documents, where documents are similar to rows of tables.

4. Documents can have one or more fields, where documents are similar to columns of tables.
5. Indexes of MongoDB work as a counterpart of RDBMS.
6. Cursors are used to point the result set of data.

#### 1.1.1.1. Features of MongoDB [5]

MongoDB provides many of the database functionalities that relational databases do. Apart from inserting, selecting, updating and deleting data, the following are some additional features of MongoDB:

- **Indexing:** MongoDB supports generic secondary indexes, permitting a variety of fast queries, and gives unique, compound, and full-text indexing capabilities as well.
- **Aggregation:** MongoDB supports an “aggregation pipeline” that permits us to build complex aggregations from simple pieces and allows the database to streamline it.
- **Special collection types:** MongoDB supports time-to-live accumulations for data that ought to terminate at a certain time, such as sessions. It supports fixed-size collections, which are useful for holding recent data, such as logs.
- **File storage:** MongoDB supports an easy-to-use protocol for storing large files and file metadata.

#### 1.1.1.2. MongoDB in MEAN Stack

In MEAN stack, MongoDB is the data store for applications. The node server supports a variety of databases so the MongoDB can be replaced by any other database. Reasons why the MongoDB fits well in the MEAN stack are listed below [4]:

- **Document orientation:** MongoDB is document-oriented and stores data in the JSON format which is very similar to what is dealt in server-side and client-side scripts. It doesn't require the need to transfer data from rows to objects and vice versa. For example, to create a document consisting of contact information and a contact address, it's not necessary to create two separate documents for basic information and address details. Instead, they can be bundled together inside the same object under a different key-value pair as shown in Figure 1.

```
{
  Id: "12324"
  Name: "Bob Smith"
  Address: {
    Address: "1140 Humboldt Ave S, apt # 102"
    City: "Minneapolis"
    State: "MN"
  }
}
```

**Figure 1 Sample of MongoDB document**

- **High performance:** MongoDB is known for its high performance and is capable of dealing with the heavy traffic of users in an application. Corresponding to the current use of an application, the back-end needs to be configured to handle heavy loads. The MongoDB is highly capable of achieving this.
- **High availability [7]:** A replica set of the MongoDB maintains the same data set and provides redundancy and high availability keeping high performance intact.
- **High scalability [8]:** MongoDB provides high scalability in performance by considering its various key dimensions like hardware, application pattern, schema design, and indexing [5]. MongoDB was intended to scale. Its document-oriented model makes it simpler for it to part up information over various servers. MongoDB

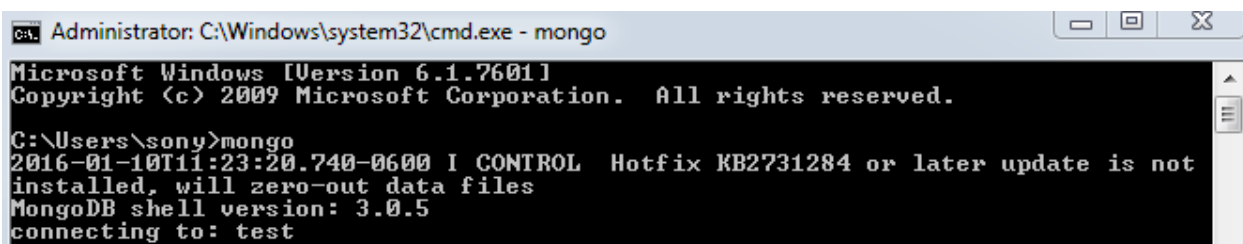
consequently deals with adjusting information and loads over a group, redistributing documents naturally and directing user request to appropriate machines. At the point when a group requires more limits, new machines can be included and MongoDB will make sense of how the current information ought to be spread to them.

- **No SQL injection [9]:** Database security is one of the most critical aspects of information security. Access to an enterprise database grants an attacker control over critical data. Back-end security is the most critical part that all application developers need to focus on. With SQL injection, hackers can get control over critical data via malicious queries. So, developers need to focus on SQL injection while using any back-end database. In MongoDB, the data stored are JSON objects and are manipulated via JSON queries not SQL strings. So, MongoDB is not susceptible to SQL injection.

### 1.1.1.3. MongoDB shell

MongoDB accompanies a JavaScript shell that permits interaction with a MongoDB instance from the command line. With the shell, various queries can be performed on the data including creating new documents, inserting new objects, finding data, and updating.

To run the Mongo shell, run mongo command and it will display version of Mongo shell, and will be connected to test database as shown in Figure 2.



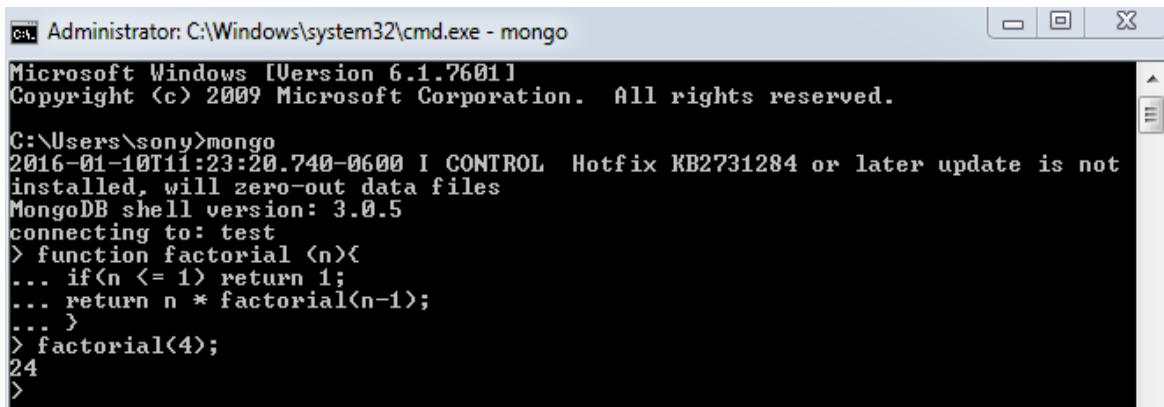
```
Administrator: C:\Windows\system32\cmd.exe - mongo
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\sony>mongo
2016-01-10T11:23:20.740-0600 I CONTROL Hotfix KB2731284 or later update is not
installed, will zero-out data files
MongoDB shell version: 3.0.5
connecting to: test
```

**Figure 2 Screenshot of Mongo shell**



The shell can be taken as JavaScript interpreter and is capable of running JavaScript programs as illustrated in Figure 3 showing that Mongo shell is capable of running factorial function calculating the result of `factorial(4)`.



```
Administrator: C:\Windows\system32\cmd.exe - mongo
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\sony>mongo
2016-01-10T11:23:20.740-0600 I CONTROL Hotfix KB2731284 or later update is not
installed, will zero-out data files
MongoDB shell version: 3.0.5
connecting to: test
> function factorial (n){
... if(n <= 1) return 1;
... return n * factorial(n-1);
... }
> factorial(4);
24
>
```

**Figure 3 JavaScript program in Mongo shell**

### 1.1.2. Express

Express is a lightweight framework based on Node. It sorts out web applications on the server-side. Express is defined as "an insignificant and adaptable Node web application structure" [4].

Express utilizes the Node HTTP module and connects components which are termed middleware. Middleware is a pattern which permits code reuse inside the applications and even imparts it to others as NPM modules. The key meaning of middleware is a capacity with three parameters: request (or req), response (or res), and next [10].

Figure 4 lists the sample code that shows us how to define middleware.

```
var middleware = function (req, res, next) {  
    // code for req and res  
    next();  
};
```

**Figure 4 Defining middleware [10]**

Express is easy to configure, implement, control and provides several key components to handle web requests. Express helps in building web applications and simple HTTP servers [4]. Since Express handles every server part and hides most of the inner working of Node, developers don't need to pay attention to additional server technology. Developers are allowed to pick whichever libraries they require for a specific task which furnishes them with adaptability and high customization. Some corporations which are using Express are MySpace, LinkedIn, Segment.io and Klout [4]. In MEAN, Express works as a medium to transfer requests from client to database and sends back responses from database to client.

#### **1.1.2.1. Features of Express**

Express is a light-weight framework for Node; it has features such as server setup, route management, session, cookie management, cache management, and error handling.

Features of Express are explained in detail below:

- **Server Setup:** Express helps server to listen incoming requests and return responses. Express defines directory structures where a developer can set folders to set static files in a non-blocking way.
- **Route management:** Express makes it simple to define routes or URL management which is attached directly to the Node script on the server.

- **Error handling:** Error handling is used across the whole application, and Express provides an in-built support for handling various errors such as Error 401, 404, and 500. In Express, error responses can be sent in multiple formats as shown in Figure 5, Figure 6, and Figure 7.

```
//Text Format
app.use(function(err, req, res, next){
  res.send(404, 'Internal Server Error.');
```

**Figure 5 Error handling as Text Format**

```
//Error HTTP response status
app.use(function(err, req, res, next) {
  res.send(500);
})
```

**Figure 6 Error handling as HTTP response status**

```
//JSON Format
app.use(function(err, req, res, next) {
  res.send(500, {status:500, message: 'internal error', type:'internal'});
})
```

**Figure 7 Error handling as JSON Format**

- **Easy integration:** Express can be implemented simply on the reverse proxy server, e.g. [Nginx](#), which allows integrating it into the existing secured system.
- **Cookies:** Express provides simple cookie management.
- **Session and cache management:** Express also enables session management and cache management [4].

### 1.1.3. AngularJS

AngularJS is a client-side JavaScript MVC system maintained by Google. It provides all the functionality to handle client information in the browser, control information on the client-side, and handle how components are shown in the browser view [11].

Table 1 explains a brief overview of AngularJS.

**Table 1 Conceptual Overview [3]**

S.No.	Concept	Description
1.	Template	Simply HTML codes
2.	Directives	Extended HTML with custom attributes and elements
3.	Model	Medium with which users interacts and which are shown in the views.
4.	View	It is the DOM(Document Object Model), which is viewed from the browser
5.	Controller	Business logic behind views
6.	Scope	Glue between a controller and a view.
7.	Filters	Formats the value of an expression when displayed to the user.
8.	Data Binding	Sync between model and view
9.	Module	Creating separate sections with controller, service, factory, filters and directive in order to organize app as its functionality

### 1.1.3.1. Advantages of AngularJS

Templating and two-way data binding are the major strengths of AngularJS which makes the framework easy to use and understand. Advantages of AngularJS are:

- **Data binding:** AngularJS has a very clean method for binding data to HTML elements, using its powerful scope mechanism.
- **Extensibility:** The AngularJS architecture allows us to easily extend almost every aspect of the language to provide custom implementations.
- **Clean:** AngularJS makes developers to force on writing clean, logical code.
- **Reusable code:** The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. The language often forces on doing so when creating custom services.
- **Support:** Google is investing a lot into its AngularJS project, which gives it an advantage over similar initiatives that have failed.
- **Compatibility:** AngularJS is based on JavaScript and has a close relationship with jQuery. This makes it easier to begin integrating AngularJS into development environment and reuse pieces of an existing code within the structure of the AngularJS framework [4].

### 1.1.3.2. Features of AngularJS

Two-way data binding and the MVC pattern is the main feature of AngularJS as described below:

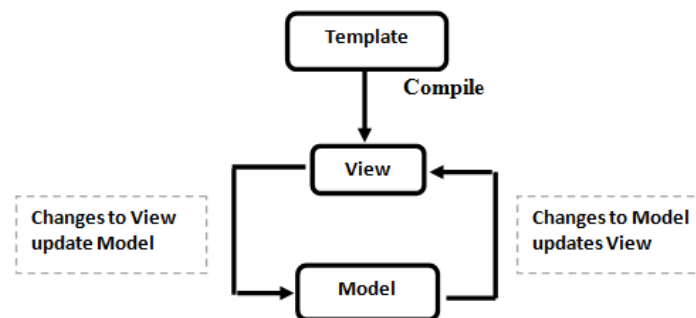
- **Two-way Data binding:** AngularJS supports two-way data binding which means any change in its template updates the model without a refresh. Similarly, AngularJS is smart enough to track any change in model without updating template or DOM. This is illustrated in Figure 9.

```
<body ng-app>  
  Name: <input type="text" ng-model="name">  
  <h1>Hello, {{name}}!</h1>  
</body>
```

**Figure 8 Sample code of two-way data binding**

In the code shown in Figure 8, any value typed in the text box will make a change in `<h1>` tag and will display the name instantly without any event call or button click.

AngularJS is distinctive in light of the fact that any perspective changes activated by a client are quickly reflected in the model, and any adjustments in the model are efficiently updated to a format.



**Figure 9 Two-way Data Binding [4]**

- **MVC pattern of AngularJS**
  - **Model:** AngularJS models are regular JavaScript objects which are not extensions of any base class or construct objects. In AngularJS, either regular JavaScript classes or objects can be used as the model.

- **View:** It is just a DOM which is viewed from the browser. “Each time it encounters a directive, AngularJS executes its logic to turn directives into dynamic parts of the screen.” [11]
- **Controller:** Controllers are standard JavaScript capacities. Controllers don't need to develop any structure for any classes nor call a specific AngularJS APIs to effectively perform their job. One of the major features of AngularJS Controller is to initialize a scope object.

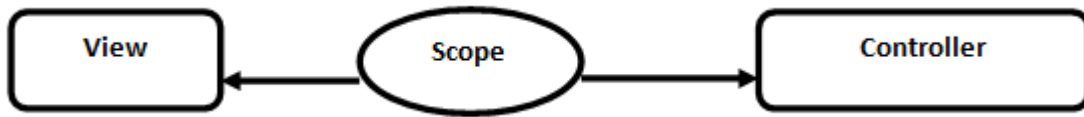
**Scope:** A scope object or view-model acts as glue between view and controller as shown in Figure 11. By allotting properties to scope instances, new values can be made accessible to the layout for rendering. Front-end logic can be exposed by defining function on a scope instance. Figure 10 illustrates the example code for defining scope in the controller.

```
var nameCtrl = function ($scope) {  
    $scope.getName = function() {  
        return $scope.name;  
    };  
}
```

**Figure 10 Scope instance**

Then use it in a template as shown in the following code:

```
<h1> Hello, {{getName()}}! </h1>
```



**Figure 11 View-model or scope [11]**

- **Dependency Injection:** An AngularJS application is a collection of several different modules that all come together to build an application. A dependency in the code occurs when one object depends on another. Dependency injection is a method by which an object can be given the dependencies that it requires to run. [4]

#### 1.1.4. Node

Node [16] is a development framework originally developed in 2009 by Ryan Dahl, which is built on Google's [V8 JavaScript engine](#). "Node is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node uses an event-driven and non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." [17]

Node is a server-side scripting language which can be used on server-side, client-side, and can even be a web server. Before Node came into existence, JavaScript was simply used for user interaction as a client-side script. In order to communicate with server, it required the support from other server-side scripting language like PHP/Perl. Node can act independently on its own on the server-side. Node even helps to reduce development time and resources and also tightens the interaction between the web server and the script because web server, client-side code, and server-side code all can be in the same language. The web server can run on a Node platform as a Node module.



In the MEAN stack, Node acts as server-side platform which is similar to the Apache system running with a server scripting language for creating applications [4].

#### 1.1.4.1. Why Node?

There are many reasons for developers to use Node. They are listed as below [12]:

- **JavaScript end-to-end:** Node allows writing code both in client-side and server-side in JavaScript. It is difficult to decide whether to use application logic in client-side or server-side. With Node, JavaScript written in client-side can be easily used and adjusted in server code and vice versa. Additionally, both client developer and server developer can use the same common language which they will understand.
- **Event-driven scalability:** Node uses a single thread to handle multiple requests. Node processes multiple requests in same thread using its event model, unlike Apache which creates a new thread to handle any request. This makes Node web server more scalable compared to Apache which follows multithreading mechanism to respond to multiple requests.
- **Extensibility:** Node has many users and extremely dynamic community. Individuals are giving new modules to amplify Node usefulness constantly. Additionally, it is extremely easy to introduce and incorporate new modules in Node; where individual can extend a Node task to incorporate new functionality.
- **Fast implementation:** Node can be installed and setups easily in few minutes and have a working web server. Many manual and documentation are available in the Internet.

### 1.1.4.2. NPM and Node packages

One of the advantages of Node is its built-in tool, Node Package Manager ([NPM](#)). NPM comes packaged with Node and helps to pull in various bundles of modules required in the application [4]. NPM resolves the dependencies and version conflict in various modules. The main purpose of NPM is to ease the use and installation of modules which are publicly available, reusable components via an online repository. NPM acts as an infrastructure in Node to manage packages. Various packages that are available via NPM can be found on its official website.

To check if NPM is already installed or not, use:

```
npm -v
```

If NPM is already installed, then it should display the version of NPM installed.

To install any module via NPM use:

```
npm install <module_name>
```

Installed packages are stored inside `node_modules`. All the details of packages like version, dependencies, and other details can be found in the `package.json` file which are auto updated after each `npm install`.

The following are some of the packages which can be easily installed via NPM:

- **[Express](#)**: A light weight framework for Node. Express can be installed by using this command:

```
npm install express
```

- [Mongoose](#): Package to interact with MongoDB. Mongoose can be installed by using this command:

```
npm install mongoose
```

- [MongoJS](#): Module to interact with MongoDB. MongoJS can be installed by using this command:

```
npm install mongojs
```

- [GruntJS](#): Package that helps to automate various tasks. GruntJS can be installed by using this command:

```
npm install gruntjs
```

- [PassportJS](#): Package for the verification of various social administrations. PassportJS can be installed using this command:

```
npm install passportjs
```

- [Socket.io](#): Package for building real-time web socket applications. Socket.io can be installed by using this command:

```
npm install socket.io
```

#### **1.1.4.3. Node Modules**

Node modules are add-ons for Node for adding extra features as needed in application development. Node modules are easily located and imported via NPM. Node modules can be either core modules, local modules, or third party modules. Core module comes with the default

installation of Node and is loaded when Node server starts. Third party modules are incorporated via NPM command.

To load any module, one needs to use 'require' function as follows:

```
var module = require ('module_name');
```

Figure 12 shows an example to load Node modules in the application:

```
var express = require('express');  
var http = require('http');  
var path = require('path');  
var mongoose = require('mongoose');  
var socketio = require('socket.io');
```

**Figure 12 Example to load various Node modules**

Some of the third party Node modules used in the application are Express, Socket.io, Mongoose and MongoJS. Details of Socket.io are explained as follows. An implementation and test result of Socket.io is given in Chapter 4.

## **Socket.io**

On the web application, in order to get real-time update, a user needs to perform some event-based actions such as button click, link click, or the whole-browser refresh. But in some applications like Twitter and Facebook, users will receive live post or tweets without performing any action, which is possible due to the implementation of real-time technology [12]. There are many mechanisms to achieve this, one of which is by using Socket.io [19]. Socket.io is a Node library for building real-time web applications such as Chat System, getting live feeds, and

polling. Its main aim is to make an application to get live updates without a need to refresh the browser.

Socket.io can be installed in Node application by issuing this command:

```
npm install socket.io
```

Socket.io is event-driven and has components for both a server-side and a client-side and these components are used to communicate on both sides. Socket.io has two kinds of messages:

- **System message:** Sent by server to the client, especially when the user connects or disconnects.
- **User message:** Sent by client to server.

### Socket.io on server

In order to setup Socket.io in server part, one needs to include Socket.io by including:

```
var socketio = require('socket.io');
```

To get started with the Socket.io function, `emit()` is used to send data to Socket.io client and `.on (<connection string>)` to listen for the incoming data as shown in Figure 13.

```

//Start Socket.io
var io = socketio.listen(server);
io.set('log level', 1);
//Socket on connect
io.sockets.on('connection', function (socket) {
  console.log('client connected');
  Message.find(function (err, allMessages) {
    if (err) {
      return console.error(err)
    };
    socket.emit('pastMessages', allMessages);
  })
});

```

**Figure 13 Socket.io start and connection**

## Socket.io on client

In order to setup Socket.io on the client the following steps are recommended:

- Create a Socket.io object.
- Use Socket.io object to send message to server and get response from the server as shown in Figure 14.

```

var socket = io.connect();
//recieve new messages from chat
socket.on('receiveMessage', function (data) {
  $scope.messages.unshift(data);
  $scope.$apply();
});
//load previous messages from chat
socket.on('pastMessages', function (data) {
  $scope.messages = data.reverse();
  $scope.$apply();
});

```

**Figure 14 Socket.io in client**

## **1.2. LAMP Stack**

LAMP is a web development platform comprising of efficient sets of open-source software; Linux as operating system, Apache as web server, MySQL as database system, and PHP/Perl as back-end scripting language. Each element in LAMP allows for seamless and smooth integration with each other which makes LAMP one of the powerful platforms for web application development. LAMP is one of the very old stacks preferred by many organizations. LAMP provides full control over server and remote access which helps to perform administrative operations via a Linux server from anywhere [13].

In the following sections, we discuss the details on the acronyms of LAMP stack.

### **1.2.1. Linux Operating System**

Linux [20] is the operating system that runs the applications and computer operators to access the devices on the computer to perform the assigned task. It is specifically noted for its speed, insignificant hardware requirements, security, and remote administration. Linux is a fully featured operating system that is free to use. Another significant point of Linux is its ability to keep running with or without a Graphical User Interface (GUI), contingent upon user needs. Linux was started by a student named Linux Torvalds from Helsinki, Finland where he was working with [Minix](#)(a [UNIX](#) framework). Torvalds decided to develop an operating system which would surpass the Minix standards. He started his advancement in 1991, and his first public release was version 0.02. Improvement of Linux proceeds even now with updates released as significant changes are made to improved new version release [13].

### **1.2.2. Apache web server**

Apache [21] is an open-source web server developed by [Apache Software Foundation](#) (ASF). Apache work well with Linux operating system and allows virtual hosting which can run multiple websites on a single server. Also available for Windows OS, the Apache server suffers from decreases in performance due to Microsoft's memory managements, as well as architectural differences, so it is preferred with Linux environment.

#### **1.2.2.1. Features of Apache server [13]**

Features of Apache server are listed below:

1. Enhanced logging.
2. Bandwidth throttling.
3. Directory access protection.
4. Common Gateway support.
5. Secure sockets layer (SSL) support.
6. Built-in models.

### **1.2.3. MySQL Database System**

MySQL is a relational database management system (RDBMs) invented by IBM researcher Edgar Frank Codd in 1970. RDBMs allows users to represent advanced relationships between data and compute their relationships with the speed, so users can go from design to implementation easily, and can develop web applications to access terabytes of data and serve thousands of web users per second [14].



MySQL is a powerful and robust database manager that enables to store and retrieve data with a scripting language such as PHP. Different types of data such as Boolean operator, text, integers, images, binary digits, ENUM or enumeration, binary large objects (BLOBs) and date can be stored. Database is one of the major components for building dynamic applications. The term “*dynamic site*” is derived from being able to utilize a single page of code to display different information based on a user’s interaction. This would be virtually impossible without the use of a database and a scripting language such as PHP to manipulate the data. MySQL is packed full of features for tasks such as data replication, table locking, query limiting, user accounts, multiple databases, persistent connections, and as of MySQL 5, stored procedures, triggers, and views [13].

#### **1.2.3.1. Features of MySQL [22]**

Following are the features of MySQL:

1. It is a database management system which helps in computing and managing large amount of data.
2. MySQL Databases are relational where data are stored in organized tables rather than one big file or storeroom.
3. It is open-source software where anyone is able to use, download and modify.
4. MySQL is fast, robust, scalable, reliable, and easy to use. It can run in any platform; desktop, laptop or any other application or web server.

5. MySQL server works in both client and server system that comprises of a multi-threaded SQL server and backings distinctive back-end, client programs and libraries and an extensive variety of APIs.
6. It provides a privilege and password system.

#### **1.2.4. PHP Scripting Language**

PHP [23] also termed PHP: Hypertext Preprocessor is a server-side scripting language or programming language for building dynamic and interactive website or web applications, which means it runs on a web server. A PHP script is processed by the PHP engine each time it runs. PHP was originally created by Rasmus Lerdorf in 1994; the PHP reference implementation is now produced by The PHP Group.

PHP can be embedded into HTML and enhance an application dynamically. A term dynamic means whenever a page is viewed, its content is loaded as per request. These requests can send either from GET Request via URL link, POST Request via Form submission. In contrast to this is Static Web page, which remains same each time it is displayed. PHP helps to build very user-friendly websites embedded with HTML and CSS. Developers can simply learn the code, apply the logic, and then create a dynamic website that can interact with users on many levels.

The process of running a PHP script on a web server is as follows:

1. A user sends the request via GET/POST/REQUEST method by clicking any link, URL address or form submission from web browser.

2. The web server recognizes that the request sent via PHP script and instructs the PHP engine to process and run the script.
3. PHP processes the request and response is sent back to the web browser which is visible to the user in the form of HTML page on their web browser [14].

There are three fundamental regions where PHP scripts are utilized.

- **Server-side scripting:** To enable server-side scripting, it needs [Common Gateway Interface](#) (CGI or server module), a web server and a web program. After the PHP installation, the user can run a web server. PHP program output will be viewed from web browser and PHP code will be viewed and written using any editors including Notepad++, Dreamweaver, and Sublime.
- **Command line scripting:** PHP script can run without server and browsers as well. This kind of utilization is perfect for scripts routinely executed utilizing [Cron](#) (on UNIX or Linux) or [Task Scheduler](#) (on Windows). These scripts can likewise be utilized for basic content preparing undertakings.
- **Writing desktop applications:** PHP is most likely not the absolute best language to make a desktop applications with graphical client interfaces, yet one might want to utilize some cutting-edge PHP highlights in a client-side applications one can likewise utilize [PHP-GTK](#) to compose such projects. It helps to compose cross-stage applications along these lines. PHP-GTK is an expansion to PHP.

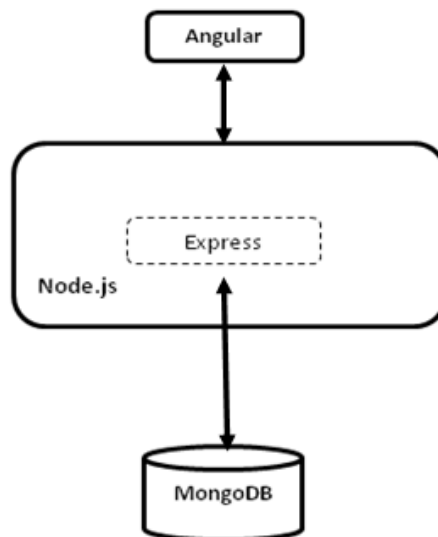
### 1.3. Architectures

This section covers the working procedure of MEAN stack and LAMP stack.

### 1.3.1. MEAN Architecture

Most web applications are implicit, a three-level architecture planning that comprises of three essential layers: data, logic, and view. In web applications, the application structure, for the most part, separates the database, server, and client. While in cutting edge web improvement, it is broken into database, server logic, client logic, and client UI (User Interface).

In the MVC architecture, the logic, data, and presentation are isolated into three sorts of objects, each taking care of its own tasks. The view handles the visual part, dealing with client interaction. The controller responds to system and client requests, making the model and view to change appropriately. The model handles information manipulation, responding to demands for data or changing its state as indicated by the controller's directions [1].



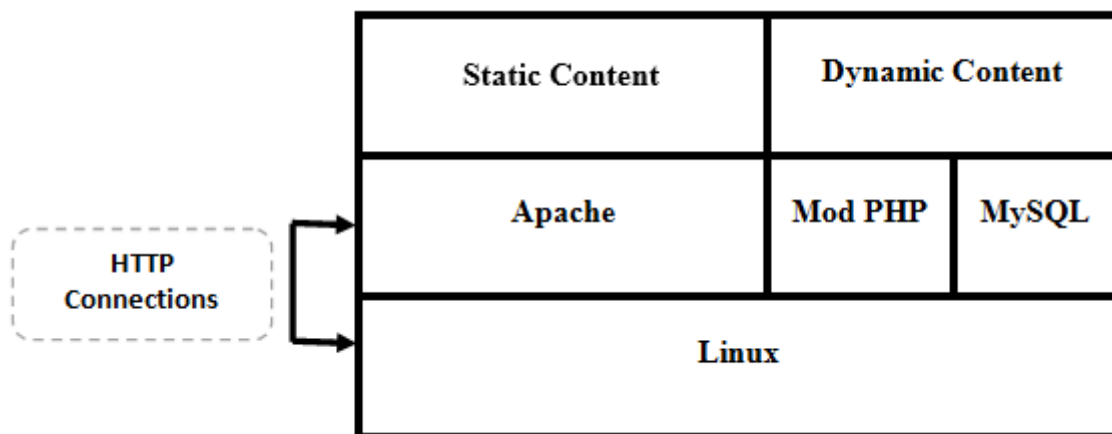
**Figure 15 Architecture of MEAN [1]**

Figure 15 shows the typical application architecture of a MEAN Stack. AngularJS, being a client side MVC, communicates to the Node server via Express. For any backend data request, Node send request via MongoDB native driver or Mongoose. The response from the server is

transferred to the client via Express. In this way, there is a transfer of request and response back and forth between the client and the server.

### 1.3.2. LAMP Architecture

LAMP is singularly focused on web applications. The architecture is very straightforward, as illustrated in Figure 16. Linux forwards HTTP connections to Apache, which serves static content directly from the Linux kernel [30]. Dynamic pages are forwarded by Apache to PHP, which runs the PHP code to design the page. Database queries are sent to MySQL through PHP. Administration is commonly handled through [phpMyAdmin](#), and every major enterprise management system can manage Apache and Linux.



**Figure 16 LAMP Architecture [30]**

## Chapter 2 MEAN vs. LAMP

The MEAN stack gains popularity due to the features of its components: Node, MongoDB, AngularJS, and Express. However, the MEAN stack should not be used for those applications which involve CPU intensive tasks. The LAMP stack is the preferred choice for such CPU intensive applications. But for simple [Single Page Application](#) (SPA), which requires high user interaction, display and need high scalability, the MEAN stack is a best fit. This chapter sheds light on the comparisons between the MEAN stack and the LAMP stack and describes some reasons for choosing the MEAN stack over the LAMP stack.

Some of the reasons for choosing MEAN over LAMP are discussed below:

- **Simplified Server Layer:** The LAMP stack needs various Apache configurations to setup server layer. One needs to have knowledge of various Apache configurations. LAMP stack requires different configuration files such as `httpd.conf`, `httpd.conf`, and `php.ini` and need to setup its configuration as per the needs of the application. The MEAN stack simplifies the server layer through the use of Node which provides the better environment for running the server. Node handles every task from app route requests to rewriting URLs or construct mapping with the use of just JavaScript code. Node avoids the need to have a configuration file for each task. Since Node provides everything in one layer, it simplifies server layer by reducing the chain of bugs created via interaction between multiple layers for the Apache server. LAMP needs a separate

web server for it to be hosted on, whereas in MEAN stack, it can be built locally using Node's built-in web server.

- **Isomorphic code:** JavaScript is the only language used in MEAN stack from server-side scripting for Node and Express to client-side scripting using AngularJS. Even for back-end, MongoDB uses JSON format to store documents. Queries can be written in JSON format in Node server and will be sent via Express.js to front-end AngularJS. This eliminates the need of different experts for front-end development, back-end development, and server management. JavaScript is the only language used across the MEAN stack unlike the LAMP stack which needs separate JavaScript experts for front-end development, PHP/Perl experts for back-end development, and server experts for server setup.
- **Scalability:** An application is scalable if it can expand as usage grows and also if it can perform better with post expansion. Scalability determines how far the application can run to meet the future requirements and expansion. In the LAMP stack, the Apache server handles each request via a multithreaded approach. A new thread is created whenever a connection is requested, to handle the request. An application might have to handle multiple requests so multiple thread is created. Depending on the server configuration, each application can process some limited number of requests. Apache will use one thread per request. If there is a situation where number of requests exceeds the server configuration, then a user may experience a connection timeout unless the ongoing processes are completed and freed up.  
  
In case of MEAN, Node utilizes an event-driven approach. Node is asynchronous and single threaded so events are executed freely. In MEAN, instead of waiting for the

event to happen, the program will pass an event to the event handler queue and continue with the main program stream. Once the event is ready, the program will be back to it with a callback function, execute the code, and then return to the main flow of the program.

- **Non-blocking architecture:** One feature that makes Node more powerful is its non-blocking architecture. In the application, blocking means that when one line of code is executing, then the rest of it is locked waiting for that current code to be finished. On the other hand, in a non-blocking architecture, Node executes each line of code and then via callback mechanism, it returns when the event actually happens. In LAMP, this issue of blocking is solved via multiple threads of execution whereas in MEAN, Node handles using non-blocking event loop in a single thread.

Figure 17 and Figure 18 show the detail processing in a multiple threaded server and the Node server.

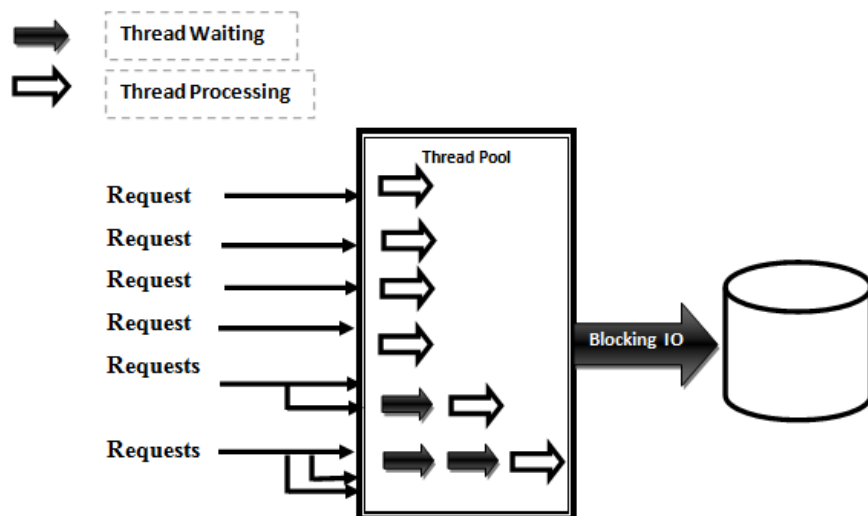


Figure 17 Multi-threading server [15]



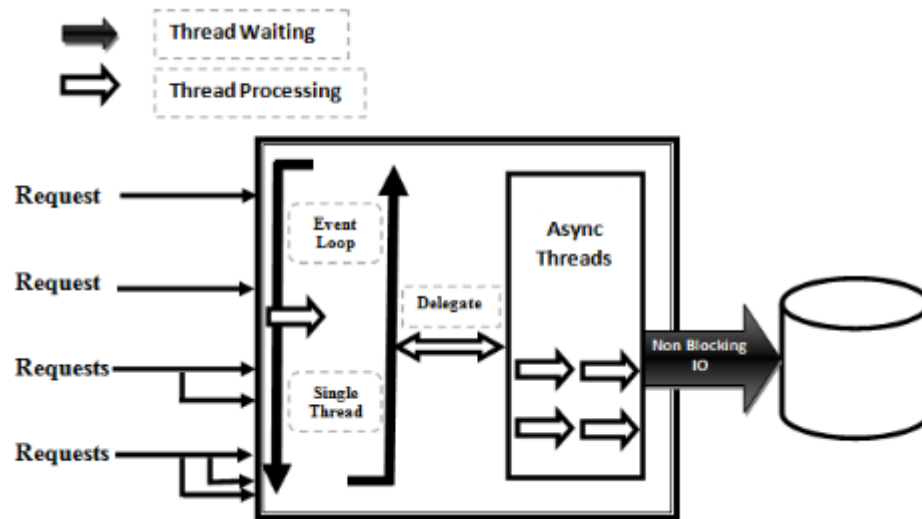


Figure 18 Node.js server [15]

- **Development time:** In MEAN, JavaScript is used in each component of the stack, so code is transparent. Transparency in code reduces the effort and the development time. Front-end developers will be able to understand back-end program and vice versa. In short, there is no need to seek different expertise for various components in MEAN, which is not the case in LAMP.
- **Data Transformation:** JSON, which is a data interchange format, makes a subset of the JavaScript language and does not have any additional feature that JavaScript already has [18]. In MEAN, AngularJS, MongoDB, Express, and Node interpret, in JSON format, for sending and receiving requests and responses between clients and servers. MEAN uses the same JSON format for information globally, which makes it easier and reduces time for formatting as it goes through each layer. The data streams perfectly among every layer without reformatting and editing. In case of the LAMP stack, MySQL has its own format for query results, and PHP has a code to import

MySQL data that helps in PHP processing, whereas in client manipulation there is a need to write extra code for it to be displayed in the browsers [4].

- **High Traffic:** Since MongoDB is scalable, it is easy to implement and has the capability to give great back-end storage for high traffic websites such as Facebook and Twitter where application needs to store voluminous comments and posts. MongoDB provides high scalability in performance by considering its various key dimensions including hardware, application pattern, schema design and indexing. MongoDB comes furnished with automatic sharing and full cluster support. MongoDB provides failover support and automatic replication. Its document-oriented model makes it simpler for it to partition information over various servers. MongoDB consequently deals with adjusting information and loads over a group, redistributing documents naturally, and directing user request to the right machines. At the point when a group requires more resources, new machines can be included, and MongoDB will allocate to them.
- **Extensibility:** The MEAN stack provides flexibility in changing the structure of the database. In any module, to add a new feature can be easily achieved via MongoDB documents. If it was in the case of LAMP, it needs a new table or new field in a table which can make huge change in application structure in coding and architecture.
- **Performance:** A number of benchmarks show that Node offers better performance due to its event-driven architecture. Many organizations including Wal-Mart, PayPal, Uber, and Yahoo, implement Node servers due to its better performance compared to other native server technologies, and it reduces the high mobile traffic. Inclusion of a Node server helps to reduce the response time drastically.

- **Real-time application development:** Many applications, such as news feed, weather update, game score live update, and share market, might need to populate data in real-time. In LAMP, these can be fulfilled by using [Ajax](#) functionality using third party libraries such as jQuery, [Backbone JS](#) and [knockoutJS](#). Although it is possible to get a real-time feed, there will be continuous requests sent to the server from the client. In case of the MEAN stack, these real-time updates can be retrieved using, the Node Module named Socket.io (See in Chapter 4, 4.1.1.). Many organizations are taking the benefits of real-time application by integrating Node into many of their operations.
- **SQL or Query Injection:** In any web application, SQL injection is a serious security issue which could damage the whole application database via user interaction such as user input POST, or GET URL. SQL Injection is a technique of affecting or hacking the system by malicious users which can inject SQL commands in the queries and could damage the system database [29]. In the LAMP stack, injected SQL query can alter SQL statement which can even delete any table or the whole database.

For example:

```
SELECT * FROM <Table name> WHERE <id> = <Value>
```

<Value> can be parameter passed from user input or GET URL. If the user tries to pass malicious parameters, then it may delete the table or drop database as shown below:

```
SELECT * FROM <Table name> WHERE <id> = 1; DROP TABLE
<Table name>
```

Developer must apply various functionality and logic to prevent the system from SQL injection. In the MEAN stack, client requests are assembled in the MongoDB queries,

which are built in a BSON object. The queries are not built in a string format so conventional SQL injection attack is not a problem and queries built in BSON format provides an injection free process [17]. Basically, in BSON format we pass key-value in the query as shown in example below:

```
BSON query = BSON ("name" << <Value>);
```

The query will have a value such as (name: "Adam"). If the user tried to pass various malicious character such as commas, colons, or brackets then the query would not match any document. And the malicious users won't be able to hack the system and make any alter or drop operation in the database.

## **Chapter 3 Application Development Environment Setup**

The MEAN stack and LAMP stack are two different web application development environments containing distinct components in their architecture (See in Chapter 1). These components comprise the building blocks of both MEAN and LAMP stack architecture. And these components need to be setup prior to the initiation of application development using either of the stacks (MEAN and LAMP). In this chapter, we focus on application development by setting up the environment and configuring the installation and integration of those individual components for the MEAN and LAMP stack.

### **3.1. MEAN Stack Application Development**

In MEAN, Node is used for the back-end server on which the entire application runs. Express acts as back-end framework for Node that provides a robust set of features for applications. Angular functions as front-end framework for application act as the interface for users. MongoDB is the database where the application data are stored. Node, Express, Angular and MongoDB, build the MEAN stack and together they can be used to build powerful application using only JavaScript on both the back-end and the front-end.

#### **3.1.1. Environment Setup**

MEAN can be installed in any platform: Linux, Windows and Mac OS X. Initially, all the components need to be installed, i.e. Node, MongoDB, AngularJS, and Express of the stack. The

installation of each component of MEAN is quite simple. The remaining steps are listed in a chronological order:

1. In order to setup a Node server, a folder needs to be created for the application where all the application files will be stored and created. This can be accomplished by issuing following DOS command:

```
mkdir <application_folder>
```

2. In the application folder, create a JavaScript file with any name like `app.js` or `server.js`. Then, open any terminal or command prompt and navigate to the application folder issuing following command:

```
cd <application_folder>
```

3. Install the necessary packages using NPM command. To install Express use the following command:

```
npm install express
```

This will download and install Express framework in the application by creating `node_modules` sub-directory in the root directory of the current application.

Include Express in the Node Server using `require` command so that the APIs of Express module can be used. This is accomplished by adding 2 lines of code in `server.js` file as shown in Figure 19.

```
var express = require('express');  
var app = express();
```

**Figure 19 Including Express in MEAN Application**

4. To test if server is running correctly in a defined port, include the portion of the code as shown in Figure 20.

```
app.get('/', function (req, res) {  
  var id = req.params.id;  
  res.send('This is from server');  
});  
app.listen(3000);  
console.log("Server running on port 3000");
```

**Figure 20 Testing server running at port 3000**

5. From the command prompt, type the command shown in Figure 21 to start the server:

```
D:\SCSU\arpana\ContactList>node server.js  
Server running on port 3000
```

**Figure 21 Server running at port 3000**

A successful setup of a Node server can be verified by observing the print outs in the console after issuing the server start command. For instance, in our case, the message “Server running on port 300” in the console verified a successful setup of the server in the application.

6. After that from browser, navigate to `http://localhost:3000`, where the message sent from the server file should be displayed.
7. If there are any code changes in `server.js` file, then a server refresh is required. Type `Ctrl + C` to stop the running server, then start it again by issuing this command:

```
node server.js
```

### 3.1.2. Client Template Setup

A simple HTML page can be used for the layout of the template. First, an Express command need to be set to direct the template file that contains the layout of the application with HTML, JavaScript, images and Cascading Style Sheet (CSS) files.

To connect the server to HTML template file, write the following code shown in Figure 22.

```
app.use(express.static(__dirname + '/public'));  
app.use(bodyParser.json());
```

**Figure 22 Template declaration**

In this application, all the template files are stored under `public` folder for which a new directory named `public` is created in the root folder. Inside `public`, the template file `index.html` is created where basic HTML code for the view is coded.

- **AngularJS Setup:** In the template file, AngularJS is included by linking this script just after the closing `<body>` tag.

```
<script src = "angular.min.js"></script>
```

Add `ng-app` to the HTML tag to let the application know about the inclusion of the Angular module to make it ready to use all the features of Angular in the application.

The Angular controller can be added by adding `ng-controller` then adding the script link for that controller JS file. AngularJS controller files are also added inside the `public` folder.



In the AngularJS controller, code needs to be added to connect to the Angular module as shown in Figure 23.

```
var myApp = angular.module('myApp', []);
myApp.controller('AppCtrl', ['$scope', '$http',
function($scope, $http) {
    console.log("This is from Client");
}
]);
```

**Figure 23 AngularJS setup**

From the browser's developer tool, console messages can be viewed to verify if the controller is properly connected. From this controller, the `$scope` variable is used to communicate with view. Since AngularJS is a two-way binding, this scope acts as glue between view and model.

### 3.1.3. Route Management

In any application there might be multiple routes implying that the server should know which page to direct to when requests come from the client. And, this is managed by the request functionality of Express.

In server files, we add a section of code as shown in Figure 24.

```

app.get('/pagename', function (req, res) {
  console.log('This is get request from list page');
});
app.post('/pagename', function (req, res) {
  console.log('This is get request from list page');
});
app.put('/pagename/:id', function (req, res) {
  console.log('This is get request from list page');
});
app.delete('/pagename/:id' , function (req, res) {
  console.log('This is get request from list page');
});

```

**Figure 24 Route Management**

If the server displays a console message in the command prompt as shown in Figure 24, then a console message indicates that it has received the request from the client, which can be viewed in the terminal by issuing the command:

```
node server.js
```

In the controller, we add the section of code shown in Figure 25.

```

$http.get('/ pagename').success(function(response) {
  console.log("I got the data I requested");
});
$http.post('/pagename', $scope.contact).success(function(response) {
  console.log(response);
  refresh();
});
$http.delete('/pagename/' + id).success(function(response) {
  refresh();
});
$http.put('/pagename/' + $scope.contact._id, $scope.contact).success(
function(response) {
  refresh();
}
);

```

**Figure 25 Client code for Route Management**

Figure 25 shows post, get, delete and put request that helps in route management.

### 3.1.4. MongoDB Setup

MongoDB runs by issuing following command in the command prompt:

```
mongod
```

After running `mongod` in the command prompt, at the bottom, the display of the text “listen waiting for connection on port 27017” indicates that MongoDB is working correctly.

In order to perform queries from the command prompt, open another command prompt then run the following command:

```
mongo
```

After running `mongo` in command prompt, it will display following information:

- a. Version of MongoDB.
- b. Connecting to: test.

It implies that the database is connecting and is using a default database named test.

Some of the basic queries that can be used in MongoDB are listed below:

To show all databases use this command:

```
show dbs
```

To create or use the database, use this command:

```
use <database_name>
```

MongoDB will respond with switched to DB <database\_name>

To insert object in collection, query is passed as shown in Figure 26.

```
db.<collection_name>.insert([
  field1: 'data 1',
  field2: 'data2'
])
```

**Figure 26 Insert query in MongoDB**

Note that insertion can be done in a new collection, and we don't need to create a collection, because MongoDB will create it itself.

To view object of collection use this command:

```
db.<collection_name>.find()

db.<collection_name>.find().pretty()
```

We observed that, `_id: ObjectId()` is created by itself. It is a unique key added by the MongoDB to identify each object uniquely. In the second command, `.pretty()` helps to view the object in indented format as shown in Figure 27 and Figure 28.

```

> db.addressbook.find()
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca1"), "address" : "3108 OCCIDENTAL DR",
  "latitude" : 38.55042047, "longitude" : -121.3914158 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca2"), "address" : "2082 EXPEDITION WAY",
  "latitude" : 38.47350069, "longitude" : -121.4901858 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca3"), "address" : "4 PALEN CT", "latitude" : 38.65784584, "longitude" : -121.4621009 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca4"), "address" : "22 BECKFORD CT", "latitude" : 38.50677377, "longitude" : -121.4269508 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca5"), "address" : "3421 AUBURN BLVD", "latitude" : 38.6374478, "longitude" : -121.3846125 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca6"), "address" : "5301 BONNIEMAE WAY", "latitude" : 38.52697863, "longitude" : -121.4513383 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca7"), "address" : "2217 16TH AVE", "latitude" : 38.537173, "longitude" : -121.4875774 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca8"), "address" : "3547 P ST", "latitude" : 38.56433456, "longitude" : -121.4618826 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7cca9"), "address" : "3421 AUBURN BLVD", "latitude" : 38.6374478, "longitude" : -121.3846125 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7ccaa"), "address" : "1326 HELMSMAN WAY", "latitude" : 38.60960217, "longitude" : -121.4918375 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7ccab"), "address" : "2315 STOCKTON BLVD", "latitude" : 38.55426406, "longitude" : -121.4546045 }
{ "_id" : ObjectId("5692f13a16ab1b3601c7ccac"), "address" : "5112 63RD ST", "latitude" : 38.55426406, "longitude" : -121.4546045 }

```

Figure 27 To display data of Collection with `find()`

```

> use addressbook
switched to db addressbook
> db.addressbook.find().pretty()
{
  "_id" : ObjectId("5692f13a16ab1b3601c7cca1"),
  "address" : "3108 OCCIDENTAL DR",
  "latitude" : 38.55042047,
  "longitude" : -121.3914158
}
{
  "_id" : ObjectId("5692f13a16ab1b3601c7cca2"),
  "address" : "2082 EXPEDITION WAY",
  "latitude" : 38.47350069,
  "longitude" : -121.4901858
}
{
  "_id" : ObjectId("5692f13a16ab1b3601c7cca3"),
  "address" : "4 PALEN CT",
  "latitude" : 38.65784584,
  "longitude" : -121.4621009
}

```

Figure 28 To display data of Collection with `find().pretty()`

To use MongoDB in the stack, different packages can be used like MongoJS, or Mongoose, which can respectively be installed using command given below:

```
npm install mongojs
```

```
npm install mongoose
```

These commands will install the packages into `node_modules` folder.

In order to include MongoJS module in the server use `require` function as shown below:

```
var mongojs = require('mongojs');  
  
var db = mongojs('db_name', ['collection_name']);
```

So, at first it includes MongoJS module and then it will ask to provide MongoDB database and collection name.

## **3.2. LAMP Stack Application Development**

In this section, we focus on the environment setup and application development using the LAMP stack.

### **3.2.1. Environment Setup**

A virtual environment Linux platform (Ubuntu) was setup in a virtual machine using [VMware](#).

### **3.2.2. Apache Setup**

Apache server was installed in Ubuntu by issuing following command in terminal:

```
sudo apt-get install Apache2
```

Proper installation of Apache was confirmed by issuing one of following URLs in the browser:

```
http://localhost
```

or

`http://127.0.0.1`

### 3.2.3. PHP Setup

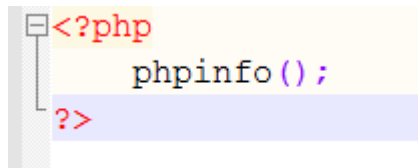
To setup PHP in Apache server, the following command was used:

```
sudo apt-get install php5 libApache2-mod-php5
```


After the installation of PHP, the Apache server was restarted using:

```
sudo /etc/init.d/Apache2 restart
```

Installation of PHP was checked by creating the `test.php` file in the root of Apache server as shown in Figure 29 then fetching it in the browser by entering this URL: `http://localhost/test.php`. The display of PHP Module information as shown in Figure 30 confirmed its correct installation in Apache server.

A code editor snippet showing PHP code. The first line is `<?php` in red. The second line is `phpinfo();` in black. The third line is `?>` in red. The code is enclosed in a light blue box with a vertical scrollbar on the left.

**Figure 29 To check phpinfo ()**

<div> <div>PHP Version 5.5.19</div>  </div>	
System	Windows NT SONY-PC 6.1 build 7601 (Windows 7 Home Basic Edition Service Pack 1) i586
Build Date	Nov 12 2014 12:29:42
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\x86\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20121113
PHP Extension	20121212

**Figure 30 PHP Module information displayed in <http://localhost/test.php>**

### 3.2.4. MySQL Connection

- To install the MySQL server in Ubuntu, the following command was used:

```
sudo apt-get install mysql-server
```

In the installation procedure, we need to configure username and password. The default username, i.e. `root` and default password, i.e. blank, was used.

- To check whether MySQL was installed correctly, type following command:

```
mysql
```

- To list existing databases, use this command:

```
show databases;
```



By default, MySQL have two databases: `information_schema` and `test`.

- To have privilege of super user, we need to assign username and password using this command:

```
mysql -u root -p
```

## Chapter 4 Case Study

Depending upon the application requirements, we can choose either the MEAN stack or the LAMP stack for application development. We focus on the fact that the MEAN stack is preferred over LAMP stack (See Chapter 2). In order to test and analyze the pros and cons of the MEAN and LAMP stacks, two sample applications, Chat System and Address Book System, were developed using both stacks followed by some benchmarking tests. Some of the criteria such as real-time application development, performance, database flexibility, and data transformation are demonstrated in this chapter. The Chat System demonstrates a real-time application development scenario. The Address Book System is used to examine the performance, data transformation, and database flexibility of the sample application. The performance of the application was measured with the help of two benchmarking tools: [Apache Bench](#) and [Siege](#).

The two applications are both built in LAMP stack and MEAN stack.

### Chat System

Chat System is an application that allows two or more users to communicate in real-time using a web interface. A similar Chat System was developed that allows multiple users to transfer and receive messages among each other in real-time via a web interface. In the developed Chat System shown in Figure 31 and Figure 32, a user is supposed to initiate or join the chat by entering a username. The chat messages and username of the Chat users were saved in the database for both the MEAN and LAMP stacks. The main motive to develop this system was to see how the Socket.io module of Node in the MEAN stack was able to replicate the

instant messaging in real-time, where as in LAMP Stack real-time communication is curtailed due to the frequent transfer of messages between users and server.

LAMP Chat Application:



**Figure 31 Screenshot of Chat System in LAMP**

MEAN Chat Application:



**Figure 32 Screenshot of Chat System in MEAN**

## Address Book System

Another application we developed for this paper is a simple CRUD (Create, Read, Update, and Delete) application having address, latitude and longitude data as in Figure 33 and Figure 34. Part of the reason for this application development was to see how easy or difficult it was to create a simple create, read, update and delete functionality in both the stacks. The main aim was to illustrate the data transfer rate in both the LAMP stack and the MEAN stack, database flexibility, and scalability. For the testing purpose, around 16K address records were inserted in both the stack applications.

### Address Book List By LAMP Stack

Address	Latitude	Longitude	Action		
<input type="text" value="Address"/>	<input type="text" value="Latitude"/>	<input type="text" value="Longitude"/>	<input type="button" value="Add Address"/>	<input type="button" value="Update"/>	<input type="button" value="Clear"/>
2082 EXPEDITION WAY	38.47350069	-121.4901858	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
4 PALEN CT	38.65784584	-121.4621009	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
22 BECKFORD CT	38.50677377	-121.4269508	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
3421 AUBURN BLVD	38.63744780	-121.3846125	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
5301 BONNIEMAE WAY	38.52697863	-121.4513383	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
2217 16TH AVE	38.53717300	-121.4875774	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
3547 P ST	38.56433456	-121.4618826	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
3421 AUBURN BLVD	38.63744780	-121.3846125	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
1326 HELMSMAN WAY	38.60960217	-121.4918375	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
2315 STOCKTON BLVD	38.55426406	-121.4546045	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	

**Figure 33 Screenshot of Address Book System in LAMP stack**

## Address Book List By MEAN stack

Address	Latitude	Longitude	Action		
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="Add address"/>	<input type="button" value="Update"/>	<input type="button" value="Clear"/>
3108 OCCIDENTAL DR	38.55042047	-121.3914158	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
2082 EXPEDITION WAY	38.47350069	-121.4901858	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
4 PALEN CT	38.65784584	-121.4621009	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
22 BECKFORD CT	38.50677377	-121.4269508	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
3421 AUBURN BLVD	38.6374478	-121.3846125	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
5301 BONNIEMAE WAY	38.52697863	-121.4513383	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	
2217 16TH AVE	38.537173	-121.4875774	<input type="button" value="Remove"/>	<input type="button" value="Edit"/>	

**Figure 34 Screenshot of Address Book System in MEAN stack**

### 4.1. Tests and Results

This section shows some of the criteria discussed in Chapter 2 to compare the test results between the MEAN and LAMP stacks.

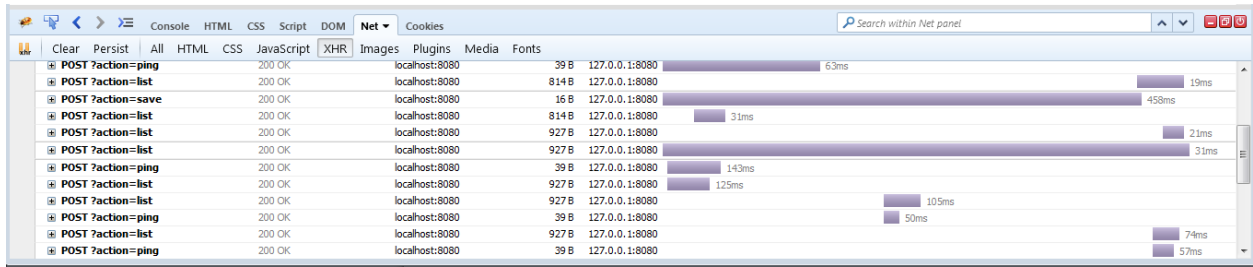
#### 4.1.1. Real-Time Web Test for Chat System

The Chat System needs to get real-time messages without the aid of user actions like a button click or a page refresh. In MEAN stack, we used Socket.io which is a Node module to get real-time message updates and in LAMP stack we used AJAX which needs to send request to server in regular interval to get the updated message. The number of requests sent from client to server i.e. web performance can be observed via add-ons or pre-existing features in browser's developer tool.

## Web performance

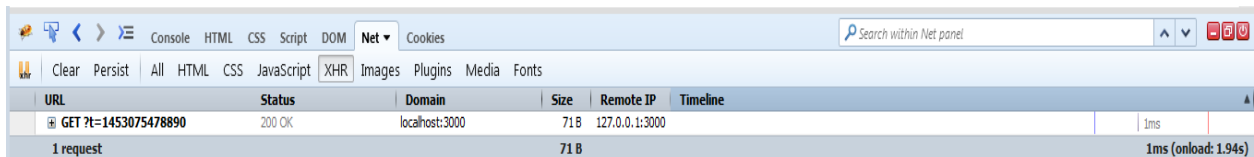
In most of the modern browsers like Firefox and Chrome, web performance can be measured either by using browser's developer tool or by installing add-on tools like Firebug (for Firefox) which helps to measure the performance of any web application [26]. Firebug is an extension for the Mozilla Firefox browser that allows debugging and inspecting HTML, DOM, and JavaScript, and for detecting performance of any web applications. From the *Net tab* of Firebug, we can retrieve information such as the number of requests with total size, cache, IP addresses, status, the type of each request, and the total timeline for each process.

For the Chat System built in the LAMP stack, in order to get real-time client-server interaction, a client needs to send a request to the server in a set of time intervals which can be viewed from the *Net Tab* using Firebug plugin in Firefox browser as shown in Figure 35. The *Net Tab* lists the URL of the request, status of the HTTP request, domain, remote IP address, size of request, and the timeline to complete that request. In order to get the live messages, the client needs to send request to the server in a regular time interval. We see consecutive number of POST methods with action as `list` and `ping` so that we get real-time messages.

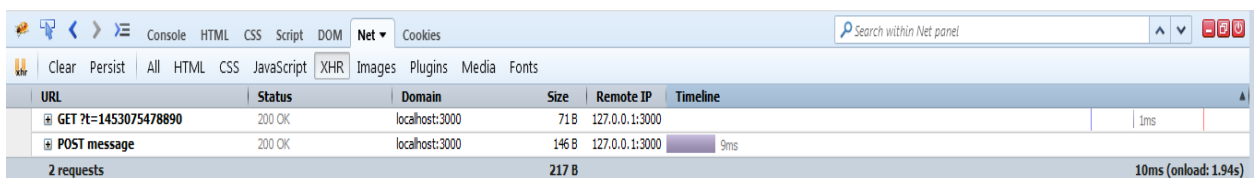


**Figure 35 Client request to get real-time update in LAMP stack**

Whereas in the case of the Chat System built in the MEAN stack, only two requests are sent, one for getting the old messages, and the other for posting message as shown in Figure 36 and Figure 37. The MEAN stack uses Socket.io, therefore the server is able to listen to the changes in the client to retrieve the live messages, as soon as the user sends the chat messages it will be automatically updated in the message list.



**Figure 36 GET method to load old messages in MEAN stack**



**Figure 37 POST method to send message in MEAN stack**

Using the Socket.io module of the MEAN stack (See Chapter 1), we can get real-time updates without refreshing the page or without any event trigger.

#### 4.1.2. Extensibility

Initially, the Address Book System stored an address, latitude and longitude. In order to check the scalability and data structure flexibility, additional information including city, zip code, and state were added. In the case of the LAMP stack, the addition of information is only possible by a change in data structure. For that reason, additional tables were created to store additional data and also to check how JOIN queries act in process execution. But in MEAN, there is no need to change the database data structure to store additional data. In the LAMP stack, after updated requirements, two tables; one for storing the city, state, and zip code (AddressDetail) and the other for storing latitude and longitude (LatLongDetail) were added. The main table (AddressList) was joined with AddressDetail.

The database structures of AddressDetail before update in the LAMP and MEAN stacks are shown in Table 2 and Figure 38.

**Table 2 Table schema of AddressList before update in LAMP stack**

Field	Type	Null	Key	Extra
Id	Int (10) unsigned	NO	PRIMARY	Auto increment
Address	Varchar (100)	NO	NULL	
Latitude	Varchar (35)	NO	NULL	
Longitude	Varchar (35)	NO	NULL	



```

addresslist = {
  _id: int,
  address: string,
  latitude: string,
  longitude: string
}

```

**Figure 38 Schema of collection before update in MEAN stack**

In the LAMP stack, for the new updated requirements, AddressList needs to add a foreign key to store addressDetailID and a new table addressDetail is needed as shown in Table 3 and Table 4.

**Table 3 Table schema of updated AddressList in LAMP stack**

Field	Type	Null	Key	Extra
Id	Int(10) unsigned	NO	PRIMARY	Auto increment
Address	Varchar (100)	NO	NULL	
Latitude	Varchar (35)	NO	NULL	
Longitude	Varchar (35)	NO	NULL	
addressDetailID	Int(10) unsigned	NO	FOREIGN	

**Table 4 Table schema of AddressDetail in LAMP stack**

Field	Type	Null	Key	Extra
Id	Int(10) unsigned	NO	PRIMARY	Auto increment
City	Varchar (100)	NO	NULL	
Zip	Varchar (20)	NO	NULL	
State	Varchar (100)	NO	NULL	

For the MEAN stack, the database schema doesn't need to be changed to fulfill the new requirement. In the same collection, nested JSON fields can be passed to represent the newly added data as shown in Figure 39.

```
addresslist = {  
  _id: int,  
  address: string,  
  latitude: string,  
  longitude: string,  
  addressDetail: {  
    city: string,  
    zip: string,  
    state: string  
  }  
}
```

**Figure 39 Schema of MEAN Stack after update**

### **4.1.3. Performance and Load Testing**

In any application development, performance and load balance are essential. For calculating the performance and load testing on the server, benchmarking tools such as Siege and Apache Bench were used. For testing the performance of both stacks, experiments were conducted using Apache Bench and Siege as described in the next section.

#### **4.1.3.1. Response time test using Apache Bench**

Apache Bench is a command line platform. In Windows OS, Apache Bench comes with default Apache server installation. And in Ubuntu, Apache Bench can be installed using following commands in the terminal:

```
sudo apt-get install apache2-utils
```

For this test, we used Apache Bench tool to test the response time of Address Book System developed in the MEAN stack running in the Node server and the LAMP stack running in Apache server with various levels of concurrency and numbers of requests. Concurrency is the measure of how many simultaneous user sessions are active on a web application at a given time. With an increase in concurrency, we can test how many concurrent user sessions a web application can support in terms of response time to perform any task.

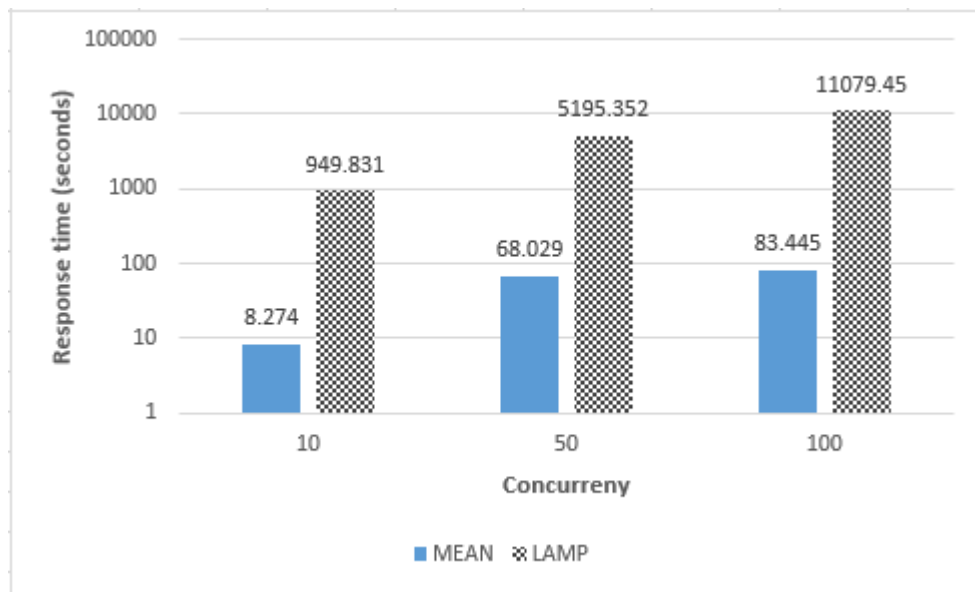
This is a sample command for benchmarking a server running locally on port 8080 with 5,000 total requests and 1,000 concurrent requests:

```
ab -n 5000 -c 1000 http://localhost:8080/
```

The MEAN stack running in Node server executed faster as indicated by the lower response time of MEAN compared to LAMP running in the Apache server. Several tests were performed with increases in concurrency and the number of request as shown in Table 5. The tests were performed for MEAN and LAMP with the concurrency set to 10, 50, and 100, and the number of request as 500, 1000, 5000, and 10000. Even with the increase in concurrency and number of requests, the response times seem to be lower in the MEAN stack than it was for LAMP stack as shown in Figure 40 which indicates that response time of the MEAN stack is almost 100 times faster than that of the LAMP stack.

**Table 5 Response time in MEAN and LAMP**

Concurrency	Number of Request							
	500		1000		5000		10000	
	MEAN	LAMP	MEAN	LAMP	MEAN	LAMP	MEAN	LAMP
10	11.821	1020.778	13.551	1011.168	14.109	1006.276	8.274	949.831
50	65.404	5923.339	64.154	5528.416	63.594	5178.956	68.029	5195.352
100	137.808	10675.21	134.208	11444.56	129.067	11040.79	83.445	11079.45



**Figure 40 Response Time Graph with concurrency and 10000 Requests**

#### 4.1.3.2. Data transfer test using Siege

For testing loads in both applications built in the MEAN and LAMP stacks, the Siege benchmarking tool was used. Siege is an HTTP benchmarking and load testing tool. The installation procedures for Siege in Windows and Linux are:

For installing Siege in Windows following two steps need to be performed. First, download the latest Siege zipped file. And then extract it to use it from the command line.

For installing Siege in Linux, following command needs to be issued from the terminal:

```
sudo apt-get install siege
```

Below given is a sample command for benchmarking a server running locally on port 8080 with 1,000 concurrent requests:

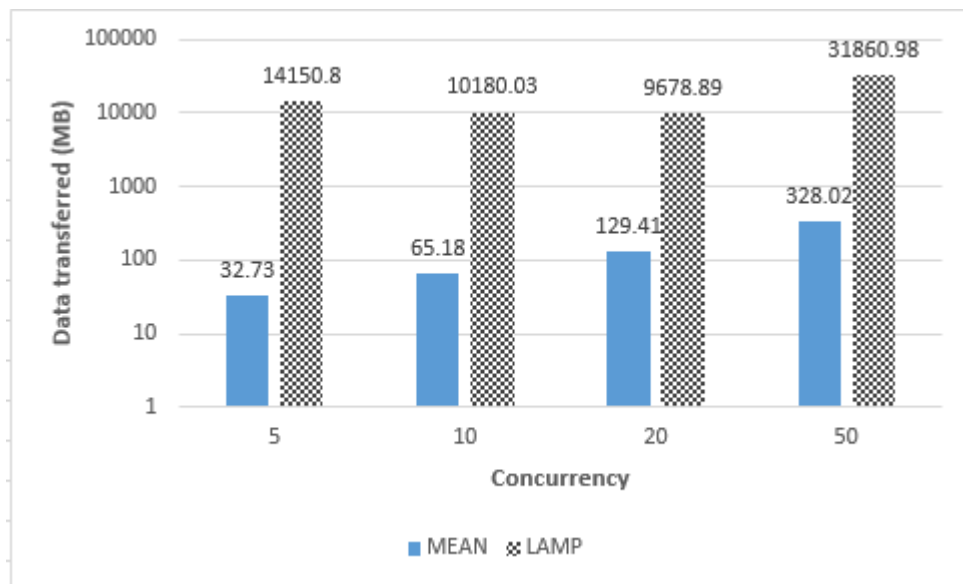
```
siege -c 1000 http://localhost:8080/
```

The test performed by Siege provides elapsed time, the amount of data transferred, response time, throughput, and transaction rate. These data were collected from Siege by passing the concurrency value and the URL of the application as the input. For illustrating the data transfer rate in Address Book System, Siege was used to measure the amount of data transferred in the MEAN and LAMP stack applications. The data transferred is the sum of the data transferred which includes the header information as well as the content. The result of the Siege test for data transferred for both the LAMP and MEAN stacks is shown in Table 6. The comparison graph of showing data transformation between the MEAN and LAMP stacks is shown in Figure 41. We see the difference in data transferred in both stacks. The MEAN stack is

only one of many language applications where only JSON is used to transfer data from client to server and vice versa. A MEAN application takes less data compared to a LAMP stack. In the LAMP stack, there is a need to convert data from JSON to PHP and vice versa to exchange data between client and server. Also, there is a need to convert data from PHP to SQL to send data between server and back-end database.

**Table 6 Data transferred in MEAN and LAMP with different concurrency**

	Concurrency (MB)			
	5	10	20	50
<b>MEAN</b>	32.73	65.18	129.41	328.02
<b>LAMP</b>	14150.8	10180.03	9678.89	31860.98



**Figure 41 Data Transferred Graph**

## **Chapter 5 Limitations and future enhancement**

In this paper, we focused on the introduction to the MEAN and LAMP stacks followed by the situations where MEAN stack excels relative to the LAMP stack and where the MEAN stack is therefore preferred over LAMP stack. We didn't mention those situations where the MEAN stack should not be used. An analysis of the shortcomings of the MEAN stack would provide us with a better understanding of MEAN and its applicability. Though, this paper demonstrated how the MEAN stack is good for real-time application development, and how it is scalable to cope with the requirement changes by two sample applications, there are limitations to our case study. For example, our case study doesn't show how MEAN reacts with CPU intensive applications. Therefore, this paper could benefit by introducing a case study that deals with CPU intensive tasks.

Following are some of the cases where the MEAN stack would not work efficiently and in those situations, the LAMP stack might be a best fit [28].

### **Maintaining relational data and transactions:**

The MEAN stack uses MongoDB for data storage which is good for data flexibility and scalability. But, MongoDB is not good in applications that need to maintain relational data and transactions. Scenarios which needs to maintain relational data and transactions are the application that needs to store a deposit into a bank account, entering the purchase details in the ecommerce application, or application that keeps the track of loans. For maintaining relational data and transactions, the LAMP stack best fits. It would be a good test to see how MEAN

performs with relational databases because not all the circumstances suffice with NoSQL databases like MongoDB. A comparison of the LAMP stack with the MEAN stack using relational databases should show how relatively stable and reliable MEAN is, when it is integrated with non-NoSQL databases. We can perform various test cases between applications, built in the LAMP stack and MEAN stack but using MySQL instead of MongoDB.

### **Access and storage pattern:**

MongoDB is a NoSQL document-oriented database where the JSON format is used to store documents. Using the property of JSON, we can flexibly derive documents using MongoDB queries. MongoDB is not designed to use joins; we need to store documents in nested JSON format. Although this makes the retrieval of record fast since no joins are used, it can be difficult to perform partial updates on nested or aggregated entities. A test can be made by comparing the application which needs partial updates built in the LAMP and MEAN stacks.

### **Concurrency model:**

In the MEAN stack, Node uses a single-threaded event loop via JavaScript's callback functionality. And these JavaScript callbacks are queued for execution. In case of LAMP stack, it uses multi-threaded approach for performing tasks. We may have different scenarios where a thread or actor model makes sense. So, we can develop those applications which needs multiple threads and see the difference using the LAMP stack and the MEAN stack.



## Chapter 6 Conclusion

We introduced and discussed two kinds of stack technologies for developing web applications: the MEAN stack and the LAMP stack. The MEAN stack is a software bundle combining MongoDB as the NoSQL database, Express as a light-weight framework of Node for the server-side scripting, Angular as client-side MVC platform for client-side scripting, and Node as the server built with JavaScript code. The LAMP stack is the combination of Linux as the operating system, Apache as the web server, PHP/Perl for server-side scripting and MySQL as the database system. In the MEAN stack, JavaScript is the only programming language both for client-side and server-side. In the LAMP stack, a different scripting language is used respectively for client-side scripting and server-side scripting.

The increasing popularity of using JavaScript as both a client-side and server-side scripting language and for database connectivity has made MEAN one of the leading web application frameworks. In MEAN, we use JavaScript for client-side scripting, server-side scripting, database queries, and as a server. With MEAN, we don't need to decide on the combination of technology to be used for front-end to back-end scripting and database connectivity. MEAN is the blend of JavaScript components which presents itself as a complete programming language. The MEAN stack is built exclusively in JavaScript, so it is one language to manage client-side, server-side, and database. All the components used in the MEAN stack are completely open-source and are currently supported by corporate developers such as MongoDB and Google. MEAN is very easy to get started as the both front-end and back-end are built using one language. All the components in MEAN are relatively light-weight. With MEAN, front-end

developers are able to understand the back-end code and back-end developers are able to understand front-end code. Even database queries in MEAN are more like JavaScript code. Since, MEAN exclusively uses JavaScript from client-side to server-side as well as for databases storing data in JSON format; it might reduce application development times and be faster.

The MEAN stack is flexible and scalable. The MongoDB does not force developers to finalize the data structure schema, which can be agile. Data structure schema can be changed as per business requirements without much impact at any stage of a software development lifecycle. Angular is good for single page application development and is responsive. Node is scalable and is good in handling asynchronous calls and managing lots of concurrent tasks in a single thread. Node creates an environment for applications requiring high-level IO tasks such as for web servers to serve multiple connections and requests with limited memory.

We compared the MEAN stack with another popular stack, the LAMP stack, which has been applied successfully for decades. In this paper, we focused on the strength of the MEAN stack with two applications (Chat System and Address Book System), built to demonstrate the benefits of the MEAN stack over the LAMP stack.

In Chat System, both the LAMP and MEAN stacks displayed real-time messages without any button clicks or page refreshes. In the LAMP stack, continuous requests were sent to the server for getting the updated message where as in the MEAN stack live messages were displayed without any requests from client to server.

Address Book System demonstrated performance in context including data transformation, response time, and database flexibility of the sample application. The performance of the application was measured with the help of benchmarking tools. The data transformation rate was

beneficially lower in the MEAN stack compared to the LAMP stack. In the MEAN stack, the same JSON formats are used to transfer data from AngularJS to Node and from Node to MongoDB, so there is no need for extra conversion. In the LAMP stack, the frontend JSON format needs to be changed to a PHP format and a PHP format to a MySQL query in order to transfer data back and forth from client to server and vice versa.

## References

- [1] Holmes, S. (2013). Getting MEAN with Mongo, Express, Angular and Node. Manning Publication.
- [2] Davis, S. (2014). Mastering MEAN introducing the MEAN Stack IBM.
- [3] AngularJS Official Document, Superheroic JavaScript MVW Framework (2010). Retrieved from <http://www.webcitation.org/6enl2KElY>
- [4] Sevilleja, C., & Lloyd, H. (2015). MEAN Machine, A beginner's practice guide to the Javascript stack. Leanpub.
- [5] Chodorow, K. & Dirolf, M. (2010). MongoDB, The Definitive Guide. O'reilly Publishing.
- [6] Segun, K. (2010). The Little MongoDB Book.
- [7] MongoDB Documentation Release 3.0.2, (April 2015), Retrieved from <http://www.webcitation.org/6enlUBi3K>
- [8] Performance Best Practices for MondoDB: A MongoDB White Paper. Dec, 2015
- [9] Imperva Web Application Attack Report, (July, 2013), retrieved from <http://www.webcitation.org/6enldqBHW>
- [10] Mardan, A. (2014). Express.JS Guide. Lean Publishing.
- [11] Kozlowski, P., & Bacon, P.D. (2013). Mastering Web Application Development with AngularJS. PACKT publication.
- [12] Rai, R. (2014). Socket.IO Real-time Web Application Development. PACKT Publishing
- [13] Rosebroc, E., & Filson, E. (2004). Setting up LAMP: Getting Linux, Apache, MySQL and PHP Working Together. Sybex Inc.
- [14] Doyle, M. (2010). Beginning PHP 5.3. Wiley Publishing.
- [15] Mejia, A. (2014). MEAN Stack Tutorial MongoDB, ExpressJS AngularJS NodeJS
- [16] Node.js Official Website. (2009). Node.js v5.1.0 Documentation, [Online], Available: <https://node.js.org>

- [17] MongoDB official link. (2016). Retrieved from <http://www.webcitation.org/6fIDfFpZY>
- [18] Smith, B. (2015). Beginning JSON. Apress Publication
- [19] Kurniawan, A. (2014). Node.js Succinctly. Syncfusion Inc.
- [20] What is Linux: An overview of the Linux Operating System. (2009). Retrieved from <http://www.webcitation.org/6enlps2je>
- [21] The Apache Software Foundation. Apache, HTTP Server. (1997 - 2015) Retrieved from <http://www.webcitation.org/6enlzEFLb>
- [22] MySQL 5.7 Reference Manual. (2016). Retrieved from <http://www.webcitation.org/6enm8ySfc>
- [23] PHP Manual. (2001 - 2016). Retrieved from <http://www.webcitation.org/6enmH7Onx>
- [24] Ortiz, A. (2013, March). Server-side Web Development with JavaScript and Node.js. Retrieved from <http://www.webcitation.org/6enmqb5TO>
- [25] Sample CSV Data. Retrieved from <http://www.webcitation.org/6enmtdF54>
- [26] Gube, J. (2008). 15 Helpful In-Browser Web Development Tools. Retrieved from <http://www.webcitation.org/6evWEIbaH>
- [27] Joe. D. (2013). Siege-windows. Available: <https://code.google.com/archive/p/siege-windows/>
- [28] Clayton, R. (2014). Mean's great, but then you grow up. Retrieved from <http://www.webcitation.org/6gOmx39S8>
- [29] W3Schools. (n.d.). SQL Injection. [Online], Available: [http://www.w3schools.com/sql/sql\\_injection.asp](http://www.w3schools.com/sql/sql_injection.asp)
- [30] Inmediate. (n.d). LAMP (Software Architecture) Retrieved from <http://www.webcitation.org/6gTOX2LIV>

## Appendix

### Source Code:

### Chat System with MEAN:

#### index.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>MEAN Chat Application:
5.   </title>
6. </head>
7. <head>
8.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
9.   <body ng-app="myApp">
10.    <divclass="container" ng-controller='ChatCtrl'>
11.      <div>
12.        <h4class="hidden-xs">MEAN Chat Application:
13.      </h4>
14.      <hr/>
15.      <divclass="box box-warning direct-chat direct-chat-warning">
16.        <divclass="box-body">
17.          <divclass="direct-chat-messages">
18.            <divclass="direct-chat-msg" ng-
19.            repeat='text in messages | filter :filterText'>
20.              <divclass="direct-chat-info clearfix">
21.                <spanclass="direct-chat-name">{{ text.username }}
22.              </span>
23.            </div>
24.              <imgclass="direct-chat-
25.              img"src="http://upload.wikimedia.org/wikipedia/en/e/ee/Unknown-
26.              person.gif"alt="">
27.            <divclass="direct-chat-text right">
28.              <span>{{ text.message }}
29.            </span>
30.          </div>
31.        </div>
32.      </div>
33.      <divclass="box-footer">
34.        <divclass="clearfix">
35.          <form>
36.            <divclass="input-group">
37.              <labelclass="radio">Enter your username
38.            </label>
39.            <inputclass="form-control" ng-
40.            model="userName" autofocus="autofocus">
41.          </div>
42.        </form>
43.      </div>
44.      <divclass="input-group">
45.        <form ng-submit="sendMessage()">
46.      </form>
47.    </div>
48.  </body>
49. </html>
```

```

43.         <input type="text" placeholder="Type message..." autofocus=
"autofocus" class="form-control" ng-model="message" ng-enter="sendMessage()">
44.         <span class="input-group-btn">
45.             <button type="submit" class="btn btn-warning btn-
flat">Send
46.                 </button>
47.             </span>
48.         </div>
49.     </form>
50. </div>
51. </div>
52. </div>
53. </div>
54. <script type="text/javascript" src="/socket.io/socket.io.js">
55. </script>
56. <script type="text/javascript" src="client.js">
57. </script>
58. </body>
59. </html>

```

## client.js

```

1. var app = angular.module('myApp', []);
2. app.directive('ngEnter', function() {
3.     return function(scope, element, attrs) {
4.         element.bind("keypress", function(event) {
5.             if (event.which === 13) {
6.                 scope.$apply(function() {
7.                     scope.$eval(attrs.ngEnter);
8.                 });
9.                 event.preventDefault();
10.            }
11.        });
12.    };
13. });
14. app.factory('MessageCreator', ['$http', function($http) {
15.     return {
16.         postMessage: function(message, callback) {
17.             $http.post('/message', message)
18.                 .success(function(data, status) {
19.                     callback(data, false);
20.                 })
21.                 .error(function(data, status) {
22.                     callback(data, true);
23.                 });
24.         }
25.     };
26. });
27. app.controller('ChatCtrl', ['$scope', 'MessageCreator', function($scope, Message
Creator) {
28.     var date = new Date();
29.     $scope.userName = '';
30.     $scope.message = '';
31.     $scope.dateTime = date;
32.     $scope.filterText = '';
33.     $scope.messages = [];
34.     var socket = io.connect();
35.     //receive new messages from chat
36.     socket.on('receiveMessage', function(data) {
37.         $scope.messages.unshift(data);
38.         $scope.$apply();

```

```

39.     });
40.     //load previous messages from chat
41.     socket.on('pastMessages', function(data) {
42.         $scope.messages = data.reverse();
43.         $scope.$apply();
44.     });
45.     //send a message to the server
46.     $scope.sendMessage = function() {
47.         if ($scope.userName == '') {
48.             window.alert('Choose a username');
49.             return;
50.         }
51.         if (!$scope.message == '') {
52.             varchatMessage = {
53.                 'username': $scope.userName,
54.                 'message': $scope.message,
55.                 'dateTime': $scope.dateTime
56.             };
57.             MessageCreator.postMessage(chatMessage, function(result, error) {
58.                 if (error) {
59.                     window.alert('Error saving to DB');
60.                     return;
61.                 }
62.                 $scope.message = '';
63.             });
64.         }
65.     };
66. });

```

## app.js

```

1. var express = require('express');
2. var http = require('http');
3. var path = require('path');
4. var mongoose = require('mongoose');
5. var socketio = require('socket.io');
6. var app = express();
7.
8. //Connect to local MongoDB
9. var db = mongoose.connection;
10. db.on('error', console.error);
11. mongoose.connect('MongoDB://localhost/GDG');
12.
13. //MongoDB Schemas
14. varchatMessage = newmongoose.Schema({
15.     username: String,
16.     message: String,
17.     dateTime: Date
18. });
19. var Message = mongoose.model('Message', chatMessage);
20. app.set('port', process.env.PORT || 3000); // all environments
21. app.use(express.static(path.join(__dirname, 'public')));
22. app.use(express.favicon());
23. app.use(express.logger('dev'));
24. app.use(express.json());
25. app.use(express.urlencoded());
26. app.use(express.methodOverride());
27. app.use(app.router);
28. if ('development' == app.get('env')) { // development only
29.     app.use(express.errorHandler());

```



```

30. }
31. app.post('/message', function(req, res) {
32.     var date = new Date();
33.     var message = new Message({
34.         username: req.body.username,
35.         message: req.body.message,
36.         dateTime: date
37.     });
38.     message.save(function(err, saved) {
39.         if (err) {
40.             res.send(400);
41.             return console.log('error saving to db');
42.         }
43.         res.send(saved);
44.         io.sockets.emit('receiveMessage', saved);
45.         console.log('message: ' + date);
46.     })
47. });
48.
49. app.get('/message', function(req, res) {
50.     Message.find(function(err, allMessages) {
51.         if (err) {
52.             return res.send(err);
53.         };
54.         res.send(allMessages);
55.     })
56. });
57. app.get('*', function(req, res) {
58.     res.sendFile('./public/index.html');
59. });
60. var server = http.createServer(app).listen(app.get('port'), function() {
61.     console.log('Express server listening on port ' + app.get('port'));
62. });
63. //Start Socket.io
64. var io = socketio.listen(server);
65. io.set('log level', 1);
66. //Socket on connect
67. io.sockets.on('connection', function(socket) {
68.     console.log('client connected');
69.     Message.find(function(err, allMessages) {
70.         if (err) {
71.             return console.error(err);
72.         };
73.         socket.emit('pastMessages', allMessages);
74.     })
75. });

```

## Chat System with LAMP:

### index.php

```
1. <?php
2. require_once 'Controller.php';
3. $chatApp=newController();?>
4. <!doctype html>
5. <html ng-app="ChatApp">
6.   <head>
7.     <title>LAMP Chat Application:
8.   </title>
9. </head>
10. <scriptsrc="Controller.js">
11.   </script>
12. <body ng-controller="ChatAppCtrl">
13.   <divclass="container">
14.     <h4class="hidden-xs">LAMP Chat Application:
15.   </h4>
16.   <hr/>
17.   <divclass="box box-warning warning">
18.     <divclass="box-body">
19.       <divclass="messages">
20.         <divclass="msg" ng-repeat="message in messages" ng-
21.           if="historyFromId< message.id" ng-class="{ 'right':!message.me}">
22.           <divclass="chat-info clearfix">
23.             <spanclass="chat-name" ng-class="{ 'pull-left':message.me, 'pull-
24.               right':!message.me}">{{ message.username }}
25.             </span>
26.             <spanclass="chat-timestamp " ng-class="{ 'pull-
27.               left':!message.me, 'pull-right':message.me}">{{ message.date }}
28.             </span>
29.             <imgclass="chat-img"src="Unknown-person.gif"alt="">
30.             <divclass="chat-text right">
31.               <span>{{ message.message }}
32.             </span>
33.             </div>
34.           </div>
35.           <divclass="box-footer">
36.             <divclass="clearfix">
37.               <form>
38.                 <divclass="input-group">
39.                   <labelclass="radio">Enter your username
40.                   </label>
41.                   <inputclass="form-control" ng-
42.                     model="me.username" autofocus="autofocus" ng-blur="WriteCookie()">
43.                   </div>
44.                 </form>
45.               </div>
46.               <divclass="input-group">
47.                 <inputtype="text" placeholder="Type message..." autofocus="a
48.                   utofocus"class="form-control" ng-model="me.message" ng-enter="saveMessage()">
49.                 <spanclass="input-group-btn">
50.                   <buttontype="submit"class="btn btn-warning btn-
51.                     flat">Send
52.                 </button>
```

```

50.             </span>
51.         </div>
52.     </form>
53. </div>
54. </div>
55. </div>
56. </div>
57. </body>
58. </html>

```

## controller.php

```

1. <?php
2. require_once 'SPA_Common.php';
3. require_once 'Model.php';
4.
5. class Controller extends SPA_Common\Controller
6. {
7.     protected $_model;
8.
9.     public function __construct()
10.    {
11.        $this->setModel('SPA_Chat\Model');
12.        parent::__construct();
13.    }
14.
15.    public function listAction()
16.    {
17.        $this->setHeader(array(
18.            'Content-Type' => 'application/json'
19.        ));
20.        $messages = $this->getModel()->getMessages();
21.        foreach ($messages as &$message) {
22.            $message->me = $this->getServer('REMOTE_ADDR') === $message->ip;
23.        }
24.        return json_encode($messages);
25.    }
26.
27.    public function saveAction()
28.    {
29.        $username = $this->getPost('username');
30.        $message = $this->getPost('message');
31.        $ip = $this->getServer('REMOTE_ADDR');
32.        $this->setCookie('username', $username, 9999 * 9999);
33.        $result = array(
34.            'success' => false
35.        );
36.        if ($username && $message) {
37.            $cleanUsername = preg_replace('/^' . ADMIN_USERNAME_PREFIX . '/', '',
, $username);
38.            $result = array(
39.                'success' => $this->getModel()-
>addMessage($cleanUsername, $message, $ip)
40.            );
41.        }
42.        if ($this->isAdmin($username)) {
43.            $this->_parseAdminCommand($message);
44.        }
45.        $this->setHeader(array(
46.            'Content-Type' => 'application/json'
47.        ));

```

```

48.         returnjson_encode($result);
49.     }
50.     private function _isAdmin($username)
51.     {
52.         returnpreg_match('/^' . ADMIN_USERNAME_PREFIX . '/', $username);
53.     }
54.     private function _parseAdminCommand($message)
55.     {
56.         if ($message == '/clear') {
57.             $this->getModel()->removeMessages();
58.             returntrue;
59.         }
60.         if ($message == '/online') {
61.             $online = $this->getModel()->getOnline(false);
62.             $ipArr = array();
63.             foreach ($online as $item) {
64.                 $ipArr[] = $item->ip;
65.             }
66.             $message = 'Online: ' . implode(", ", $ipArr);
67.             $this->getModel()->addMessage('Admin', $message, '0.0.0.0');
68.             returntrue;
69.         }
70.     }
71.
72.     private function _getMyUniqueHash()
73.     {
74.         $unique = $this->getServer('REMOTE_ADDR');
75.         $unique .= $this->getServer('HTTP_USER_AGENT');
76.         $unique .= $this->getServer('HTTP_ACCEPT_LANGUAGE');
77.         return md5($unique);
78.     }
79.
80.     public function pingAction()
81.     {
82.         $ip = $this->getServer('REMOTE_ADDR');
83.         $hash = $this->_getMyUniqueHash();
84.         $this->getModel()->updateOnline($hash, $ip);
85.         $this->getModel()->clearOffline();
86.         $this->getModel()->removeOldMessages();
87.         $onlines = $this->getModel()->getOnline();
88.         $this->setHeader(array(
89.             'Content-Type' => 'application/json'
90.         ));
91.         returnjson_encode($onlines);
92.     }
93. }
94. ?>
95.

```

## controller.js

```

1. varChatApp = angular.module('ChatApp', ['ngCookies']);
2. ChatApp.directive('ngEnter', function() {
3.     returnfunction(scope, element, attrs) {
4.         element.bind("keydownkeypress", function(event) {
5.             if (event.which === 13) {
6.                 scope.$apply(function() {
7.                     scope.$eval(attrs.ngEnter);
8.                 });

```

```

9.         event.preventDefault();
10.     }
11.     });
12.     };
13. });
14. ChatApp.controller('ChatAppCtrl', ['$scope', '$http', '$cookies', '$cookieStore'
    ,
15.     function($scope, $http, $cookies, $cookieStore) {
16.         $scope.urlListMessages = '?action=list';
17.         $scope.urlSaveMessage = '?action=save';
18.         $scope.urlListOnlines = '?action=ping';
19.         $scope.pidMessages = null;
20.         $scope.pidPingServer = null;
21.         $scope.me = [];
22.         $scope.messages = [];
23.         $scope.online = null;
24.         $scope.lastMessageId = null;
25.         $scope.historyFromId = null;
26.         $scope.WriteCookie = function() {
27.             $cookies.username = $scope.me.username;
28.         };
29.         $scope.me = {
30.             username: $cookies.username,
31.             message: null
32.         };
33.         $scope.saveMessage = function(form, callback) {
34.             var data = $.param($scope.me);
35.             if (!($scope.me.username && $scope.me.username.trim())) {
36.                 return $scope.openModal();
37.             }
38.             if (!($scope.me.message && $scope.me.message.trim() &&
39.                 $scope.me.username && $scope.me.username.trim())) {
40.                 return;
41.             }
42.             $scope.me.message = '';
43.             return $http({
44.                 method: 'POST',
45.                 url: $scope.urlSaveMessage,
46.                 data: data,
47.                 headers: {
48.                     'Content-Type': 'application/x-www-form-urlencoded'
49.                 }
50.             }).success(function(data) {
51.                 $scope.listMessages(true);
52.             });
53.         };
54.         $scope.replaceShortcodes = function(message) {
55.             varmsg = '';
56.             msg = message.toString().replace(/(\[img\])(.*)\[\/img\])/ , "<imgsrc='
'$2' />");
57.             msg = msg.toString().replace(/(\[url\])(.*)\[\/url\])/ , "<a href='
'$2' />");
58.             returnmsg;
59.         };
60.         $scope.getLastMessage = function() {
61.             return $scope.messages[$scope.messages.length - 1];
62.         };
63.         $scope.listMessages = function(wasListingForMySubmission) {
64.             return $http.post($scope.urlListMessages, {}).success(function(data)
65.             {
                $scope.messages = [];

```

```

66.         angular.forEach(data, function(message) {
67.             message.message = $scope.replaceShortcodes(message.message);
68.             $scope.messages.push(message);
69.         });
70.         varlastMessage = $scope.getLastMessage();
71.         varlastMessageId = lastMessage && lastMessage.id;
72.         if ($scope.lastMessageId !== lastMessageId) {
73.             $scope.onNewMessage(wasListingForMySubmission);
74.         }
75.         $scope.lastMessageId = lastMessageId;
76.     });
77. };
78. $scope.onNewMessage = function(wasListingForMySubmission) {
79.     $scope.scrollDown();
80. };
81. $scope.pingServer = function(msgItem) {
82.     return $http.post($scope.urlListOnlines, {}).success(function(data)
83.     {
84.         $scope.online = data;
85.     });
86. };
87. $scope.init = function() {
88.     $scope.listMessages();
89.     $scope.pidMessages = window.setInterval($scope.listMessages, 3000);
90.     $scope.pidPingServer = window.setInterval($scope.pingServer, 8000);
91. };
92. $scope.scrollDown = function() {
93.     varpidScroll;
94.     pidScroll = window.setInterval(function() {
95.         $('direct-chat-
96.         messages').scrollTop(window.Number.MAX_SAFE_INTEGER * 0.001);
97.         window.clearInterval(pidScroll);
98.     }, 100);
99. };
100.});

```

## Model.php

```

1. <?php
2. namespace SPA_Chats;
3. require_once 'SPA_Common.php';
4. use SPA_Common;
5. class Model extends SPA_Common\Model
6. {
7.     public function getMessages($limit = CHAT_HISTORY, $reverse = true)
8.     {
9.         $response = $this->db->query("(SELECT * FROM messages
10. ORDER BY `date` DESC LIMIT {$limit}) ORDER BY `date` ASC");
11.         return $response->getResults();
12.     }
13.     public function addMessage($username, $message, $ip)
14.     {
15.         $username = addslashes($username);

```

```

16.         $message = addslashes($message);
17.         return (bool) $this->db->query("INSERT INTO messages
18. VALUES (NULL, '{$username}', '{$message}', '{$ip}', NOW())");
19.     }
20.     public function removeMessages()
21.     {
22.         return (bool) $this->db->query("TRUNCATE TABLE messages");
23.     }
24.     public function removeOldMessages($limit = CHAT_HISTORY)
25.     {
26.         return false;
27.         return (bool) $this->db->query("DELETE FROM messages
28. WHERE id NOT IN (SELECT id FROM messages
29. ORDER BY date DESC LIMIT {$limit})");
30.     }
31.     public function getOnline($count = true, $timeRange = CHAT_ONLINE_RANGE)
32.     {
33.         if ($count) {
34.             $response = $this->db->
>query("SELECT count(*) as total FROM online");
35.             return $response->getOne();
36.         }
37.         return $this->db->query("SELECT ip FROM online")->getResults();
38.     }
39.
40.     public function updateOnline($hash, $ip)
41.     {
42.         return (bool) $this->db->query("REPLACE INTO online
43. VALUES ('{$hash}', '{$ip}', NOW())")->order(mysql_error());
44.     }
45.     public function clearOffline($timeRange = CHAT_ONLINE_RANGE)
46.     {
47.         return (bool) $this->db->query("DELETE FROM online
48. WHERE last_update<= (NOW() - INTERVAL {$timeRange} MINUTE)");
49.     }
50.     public function __destruct()
51.     {
52.         if ($this->db) {
53.             $this->db->disconnect();
54.         }
55.     }
56. }
57. ??>

```

## SPA\_Common.php

```

1. <?php
2. namespace SPA_Common;
3. define('DB_USERNAME', 'root');
4. define('DB_PASSWORD', '');
5. define('DB_HOST', 'localhost');
6. define('DB_NAME', 'chatsystem');
7. define('CHAT_HISTORY', '150');
8. define('CHAT_ONLINE_RANGE', '1');
9. define('ADMIN_USERNAME_PREFIX', 'adm123_');
10. class SPA_MySQL_Database
11. {

```

```

12.     private $_dbLink, $_queryResponse;
13.     public $lastResult;
14.     public function __construct()
15.     {
16.         $this->_connect();
17.     }
18.     private function _connect()
19.     {
20.         $this->_dbLink = mysql_connect(DB_HOST, DB_USERNAME, DB_PASSWORD);
21.         mysql_select_db(DB_NAME, $this->_dbLink);
22.     }
23.     public function query($query)
24.     {
25.         $this->_queryResponse = mysql_query($query, $this->_dbLink);
26.         if ($this->_queryResponse) {
27.             return $this;
28.         }
29.     }
30.     public function getResults()
31.     {
32.         $this->lastResult = array();
33.         if ($this->_queryResponse && !is_bool($this->_queryResponse)) {
34.             while ($response = mysql_fetch_object($this->_queryResponse)) {
35.                 $this->lastResult[] = $response;
36.             }
37.             mysql_free_result($this->_queryResponse);
38.         }
39.         return $this->lastResult;
40.     }
41.     public function getOne()
42.     {
43.         $this->lastResult = null;
44.         if ($this->_queryResponse && !is_bool($this->_queryResponse)) {
45.             $this->lastResult = mysql_fetch_object($this->_queryResponse);
46.             mysql_free_result($this->_queryResponse);
47.         }
48.         return $this->lastResult;
49.     }
50.     public function disconnect()
51.     {
52.         return mysql_close($this->_dbLink);
53.     }
54. }
55. abstract class Model
56. {
57.     public $db;
58.     public function __construct()
59.     {
60.         $this->db = new SPA_MySQL_Database;
61.     }
62. }
63. abstract class Controller
64. {
65.     private $_request, $_response, $_query, $_post, $_server, $_cookies;
66.     protected $_currentAction, $_defaultModel;
67.     const ACTION_POSTFIX = 'Action';
68.     const ACTION_DEFAULT = 'indexAction';
69.     public function __construct()
70.     {
71.         $this->_request =& $_REQUEST;
72.         $this->_query =& $_GET;

```



```

73.         $this->_post =& $_POST;
74.         $this->_server =& $_SERVER;
75.         $this->_cookies =& $_COOKIE;
76.         $this->init();
77.     }
78.     public function init()
79.     {
80.         $this->dispatchActions();
81.         $this->render();
82.     }
83.     public function dispatchActions()
84.     {
85.         $action = $this->getQuery('action');
86.         if ($action && $action .= self::ACTION_POSTFIX) {
87.             if (method_exists($this, $action)) {
88.                 $this->setResponse(call_user_func(array(
89.                     $this,
90.                     $action
91.                 ), array()));
92.             } else {
93.                 $this->setHeader("HTTP/1.0 404 Not Found");
94.             }
95.         } else {
96.             $this->setResponse(call_user_func(array(
97.                 $this,
98.                 self::ACTION_DEFAULT
99.             ), array()));
100.        }
101.        return $this->_response;
102.    }
103.    public function render()
104.    {
105.        if ($this->_response) {
106.            if (is_scalar($this->_response)) {
107.                echo $this->_response;
108.            } else {
109.                throw new \Exception('Response content must be type scalar');
110.            }
111.            exit;
112.        }
113.    }
114.    public function indexAction()
115.    {
116.        return null;
117.    }
118.    public function setResponse($content)
119.    {
120.        $this->_response = $content;
121.    }
122.    public function setHeader($params)
123.    {
124.        if (!headers_sent()) {
125.            if (is_scalar($params)) {
126.                header($params);
127.            } else {
128.                foreach ($params as $key => $value) {
129.                    header(sprintf('%s: %s', $key, $value));
130.                }
131.            }
132.        }

```

```

133.         return $this;
134.     }
135.     public function setModel($namespace)
136.     {
137.         $this->_defaultModel = $namespace;
138.         return $this;
139.     }
140.     public function setSession($key, $value)
141.     {
142.         $_SESSION[$key] = $value;
143.         return $this;
144.     }
145.     public function setCookie($key, $value, $seconds = 3600)
146.     {
147.         $this->_cookies[$key] = $value;
148.         if (!headers_sent()) {
149.             setcookie($key, $value, time() + $seconds);
150.             return $this;
151.         }
152.     }
153.     public function getRequest($param = null, $default = null)
154.     {
155.         if ($param) {
156.             return isset($this->_request[$param]) ? $this->_request[$param] : $default;
157.         }
158.         return $this->_request;
159.     }
160.     public function getQuery($param = null, $default = null)
161.     {
162.         if ($param) {
163.             return isset($this->_query[$param]) ? $this->_query[$param] : $default;
164.         }
165.         return $this->_query;
166.     }
167.     public function getPost($param = null, $default = null)
168.     {
169.         if ($param) {
170.             return isset($this->_post[$param]) ? $this->_post[$param] : $default;
171.         }
172.         return $this->_post;
173.     }
174.     public function getServer($param = null, $default = null)
175.     {
176.         if ($param) {
177.             return isset($this->_server[$param]) ? $this->_server[$param] : $default;
178.         }
179.         return $this->_server;
180.     }
181.     public function getSession($param = null, $default = null)
182.     {
183.         if ($param) {
184.             return isset($this->_session[$param]) ? $this->_session[$param] : $default;
185.         }
186.         return $this->_session;
187.     }
188.     public function getCookie($param = null, $default = null)

```

```

189.         {
190.             if ($param) {
191.                 return isset($this->_cookies[$param]) ? $this->_cookies[$param] : $default;
192.             }
193.             return $this->_cookies;
194.         }
195.         public function getModel()
196.         {
197.             if ($this->_defaultModel && class_exists($this->_defaultModel)) {
198.                 return new $this->_defaultModel;
199.             }
200.         }
201.         public function sanitize($string, $quotes = ENT_QUOTES, $charset = 'utf-8')
202.         {
203.             return htmlentities($string, $quotes, $charset);
204.         }
205.     }
206.     ?>
207.

```

## Address Book System with MEAN:

### index.html

---

```
1. <html ng-app="myApp">
2.   <head>
3.     <title>Contact List App
4.   </title>
5. </head>
6. <body>
7.   <divclass="container" ng-controller="AppCtrl">
8.     <h1>Contact List App
9.   </h1>
10.    <tableclass="table">
11.      <thead>
12.        <tr>
13.          <th>Name
14.        </th>
15.          <th>Email
16.        </th>
17.          <th>Number
18.        </th>
19.          <th>Action
20.        </th>
21.          <th>
22.        </th>
23.      </tr>
24.    </thead>
25.    <tbody>
26.      <tr>
27.        <td>
28.          <inputclass="form-control" ng-model="contact.name">
29.        </td>
30.        <td>
31.          <inputclass="form-control" ng-model="contact.email">
32.        </td>
33.        <td>
34.          <inputclass="form-control" ng-model="contact.number">
35.        </td>
36.        <td>
37.          <buttonclass="btnbtn-primary" ng-
38.            click="addContact()">Add Contact
39.          </button>
40.        <td>
41.          <buttonclass="btnbtn-info" ng-click="update()">Update
42.          </button>
43.          <buttonclass="btnbtn-info" ng-click="deselect()">Clear
44.          </button>
45.        </td>
46.      </tr>
47.      <tr ng-repeat="contact in contactlist">
48.        <td>{{contact.name}}
49.      </td>
50.        <td>{{contact.email}}
51.      </td>
52.        <td>{{contact.number}}
53.      </td>
54.        <td>
```

```

55.         <buttonclass="btnbtn-danger" ng-
      click="remove(contact._id)">Remove
56.         </button>
57.     </td>
58.     <td>
59.         <buttonclass="btnbtn-warning" ng-click="edit(contact._id)">Edit
60.         </button>
61.     </td>
62. </tr>
63. </tbody>
64. </table>
65. </div>
66. <scriptsrc="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.12/angular.m
    in.js">
67. </script>
68. <scriptsrc="controllers/controller.js">
69. </script>
70. </body>
71. </html>

```

## controller.js

```

1. var myApp = angular.module('myApp', []);
2. myApp.controller('AppCtrl', ['$scope', '$http', function($scope, $http) {
3.     console.log("Hello World from controller");
4.     var refresh = function() {
5.         $http.get('/contactlist').success(function(response) {
6.             console.log("I got the data I requested");
7.             $scope.contactlist = response;
8.             $scope.contact = "";
9.         });
10.    };
11.    refresh();
12.    $scope.addContact = function() {
13.        console.log($scope.contact);
14.        $http.post('/contactlist', $scope.contact).success(function(response) {
15.            console.log(response);
16.            refresh();
17.        });
18.    };
19.    $scope.remove = function(id) {
20.        console.log(id);
21.        $http.delete('/contactlist/' + id).success(function(response) {
22.            refresh();
23.        });
24.    };
25.    $scope.edit = function(id) {
26.        console.log(id);
27.        $http.get('/contactlist/' + id).success(function(response) {
28.            $scope.contact = response;
29.        });
30.    };
31.    $scope.update = function() {
32.        console.log($scope.contact._id);
33.        $http.put('/contactlist/' + $scope.contact._id, $scope.contact).success(
        function(response) {
34.            refresh();
35.        })

```

```

36.     };
37.     $scope.deselect = function() {
38.         $scope.contact = "";
39.     }
40. });

```

## server.js

---

```

1. var express = require('express');
2. var app = express();
3. var mongojs = require('mongojs');
4. var db = mongojs('contactlist', ['contactlist']);
5.
6. var bodyParser = require('body-parser');
7. app.use(express.static(__dirname + '/public'));
8. app.use(bodyParser.json());
9.
10. app.get('/contactlist', function(req, res) {
11.     console.log('I received a GET request');
12.     db.contactlist.find(function(err, docs) {
13.         console.log(docs);
14.         res.json(docs);
15.     });
16. });
17.
18. app.post('/contactlist', function(req, res) {
19.     console.log(req.body);
20.     db.contactlist.insert(req.body, function(err, doc) {
21.         res.json(doc);
22.     });
23. });
24.
25. app.delete('/contactlist/:id', function(req, res) {
26.     var id = req.params.id;
27.     console.log(id);
28.     db.contactlist.remove({
29.         _id: mongojs.ObjectId(id)
30.     }, function(err, doc) {
31.         res.json(doc);
32.     });
33. });
34.
35. app.get('/contactlist/:id', function(req, res) {
36.     var id = req.params.id;
37.     console.log(id);
38.     db.contactlist.findOne({
39.         _id: mongojs.ObjectId(id)
40.     }, function(err, doc) {
41.         res.json(doc);
42.     });
43. });
44.
45. app.put('/contactlist/:id', function(req, res) {
46.     var id = req.params.id;
47.     console.log(req.body.name);
48.     db.contactlist.findAndModify({
49.         query: {
50.             _id: mongojs.ObjectId(id)

```

```
51.         },
52.         update: {
53.             $set: {
54.                 name: req.body.name,
55.                 email: req.body.email,
56.                 number: req.body.number
57.             }
58.         },
59.         new: true
60.     }, function(err, doc) {
61.         res.json(doc);
62.     });
63. });
64.
65. app.listen(3000);
66. console.log("Server running on port 3000");
```

## Address Book System with LAMP:

### index.php

---

```
1. <?php
2. include_once 'dbconfig.php';
3. ?>
4. <!DOCTYPE html PUBLIC "-//
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
5. <htmlxmlns="http://www.w3.org/1999/xhtml">
6.   <head>
7.     <!-- Latest compiled and minified CSS -->
8.     <linkrel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/
       css/bootstrap.min.css">
9.     <!-- Optional theme -->
10.    <linkrel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.
      1/css/bootstrap-theme.min.css">
11.    <title>Address Book by LAMP Stack
12.    </title>
13.    <scripttype="text/javascript">
14.      functionedt_id(id)
15.      {
16.        if(confirm('Sure to edit ?'))
17.        {
18.          window.location.href='server.php?edit_id='+id;
19.        }
20.      }
21.      functiondelete_id(id)
22.      {
23.        if(confirm('Sure to Delete ?'))
24.        {
25.          window.location.href='server.php?delete_id='+id;
26.        }
27.      }
28.    </script>
29.  </head>
30.  <body>
31.    <divclass="container">
32.      <h1>Address Book List ByLAMP Stack
33.    </h1>
34.    <tableclass="table">
35.      <thead>
36.        <tr>
37.          <th>Address
38.        </th>
39.        <th>Latitude
40.        </th>
41.        <th>Longitude
42.        </th>
43.        <th>Action
44.        </th>
45.        <th>
46.        </th>
47.      </tr>
48.    </thead>
49.    <tbody>
50.      <formmethod="post"action="server.php">
51.        <tr>
52.          <td>
```



```

53.         <inputclass="form-
control"name="address" placeholder="Address"value="<?php
54.             echoisset($fetched_row['address']) ? $fetched_row['address'] : '';
55.             ?>" required>
56.         </td>
57.         <td>
58.             <inputclass="form-
control"name="latitude" placeholder="Latitude"value="<?php
59.                 echoisset($fetched_row['latitude']) ? $fetched_row['latitude'] : ''
;
60.             ?>" required>
61.         </td>
62.         <td>
63.             <inputclass="form-
control"name="longitude" placeholder="Longitude"value="<?php
64.                 echoisset($fetched_row['longitude']) ? $fetched_row['longitude']
: '';
65.             ?>" required>
66.         </td>
67.         <td>
68.             <buttontype="submit"name="btn-save"class="btnbtn-
primary">Add Address
69.             </button>
70.         </td>
71.         <td>
72.             <buttontype="submit"name="btn-update"class="btnbtn-
info">Update
73.             </button>
74.             <buttontype="submit"name="btn-cancel"class="btnbtn-
info">Clear
75.             </button>
76.         </td>
77.     </tr>
78. </form>
79. <?php
80. $sql_query = "SELECT
81. addresslists.address,
82. latlong.latitude,
83. latlong.longitude
84. FROM addresslists
85. INNER JOIN latlong ON latlong.latitude = addresslists.latitude
86. AND latlong.latitude = addresslists.latitude
87. ";
88. $result_set = mysql_query($sql_query);
89. $row = mysql_fetch_row($result_set);
90. while ($row = mysql_fetch_row($result_set)) {
91. ?>
92.         <tr>
93.         <td>
94.             <?php
95. echo $row[0];
96. ?>
97.         </td>
98.         <td>
99.             <?php

```

```

100.     echo $row[1];
101.     ?>
102.                                     </td>
103.                                     <td>
104.                                     <?php
105.     echo $row[2];
106.     ?>
107.                                     </td>
108.                                     <td>
109.                                     <a href="javascript:delete_id('<?php
110.                                     echo $row[3];
111.                                     ?>')">
112.                                     <button class="btn btn-danger">Remove
113.                                     </button>
114.                                     </a>
115.                                     </td>
116.                                     <td>
117.                                     <a href="javascript:edt_id('<?php
118.                                     echo $row[3];
119.                                     ?>')">
120.                                     <button class="btn btn-warning">Edit
121.                                     </button>
122.                                     </a>
123.                                     </td>
124.     </tr>
125. <?php
126. }
127. ?>
128. </tbody>
129. </table>
130. </div>
131. </body>
132. </html>
133.

```

## server.php

```

1. <?php
2. include_once 'dbconfig.php';
3. // delete condition
4. if (isset($_GET['delete_id'])) {
5.     $sql_query = "DELETE FROM addresslists WHERE id=" . $_GET['delete_id'];
6.     mysql_query($sql_query);
7.     header("Location: $_SERVER[PHP_SELF]");
8. }
9. // delete condition
10.
11. if (isset($_POST['btn-save'])) {
12.     // variables for input data
13.     $address = $_POST['address'];
14.     $latitude = $_POST['latitude'];
15.     $longitude = $_POST['longitude'];
16.     // variables for input data
17.
18.     // sql query for inserting data into database
19.     $sql_query = "INSERT INTO addresslists(address,latitude,longitude) VALUES('$
    address','$latitude','$longitude')";
20.     // sql query for inserting data into database
21.
22.     // sql query execution function

```

```

23.     if (mysql_query($sql_query)) {
24. ?>
25. <scripttype="text/javascript">
26. alert('Data Are Inserted Successfully ');
27. window.location.href='index.php';
28. </script>
29. <?php
30.     } else {
31. ?>
32. <scripttype="text/javascript">
33. alert('error occured while inserting your data');
34. </script>
35. <?php
36.     }
37.     // sql query execution function
38. }
39. if (isset($_GET['edit_id'])) {
40.     $sql_query = "SELECT * FROM addresslists WHERE id=" . $_GET['edit_id'];
41.     $result_set = mysql_query($sql_query);
42.     $fetched_row = mysql_fetch_array($result_set);
43. }
44. if (isset($_POST['btn-update'])) {
45.     // variables for input data
46.     $address = $_POST['address'];
47.     $latitude = $_POST['latitude'];
48.     $longitude = $_POST['longitude'];
49.     // variables for input data
50.
51.     // sql query for update data into database
52.     $sql_query = "UPDATE addresslists SET address='$address',latitude='$latitude
    ',longitude='$longitude' WHERE id=" . $_GET['edit_id'];
53.     // sql query for update data into database
54.
55.     // sql query execution function
56.     if (mysql_query($sql_query)) {
57. ?>
58. <scripttype="text/javascript">
59. alert('Data Are Updated Successfully');
60. window.location.href='index.php';
61. </script>
62. <?php
63.     } else {
64. ?>
65. <scripttype="text/javascript">
66. alert('error occured while updating data');
67. </script>
68. <?php
69.     }
70.     // sql query execution function
71. }
72. if (isset($_POST['btn-cancel'])) {
73.     header("Location: index.php");
74. }
75. ?>
76.

```

## dbConfig.php

---

```
1. <?php
2. $host      = "localhost";
3. $user      = "root";
4. $password  = "";
5. $database  = "addressbook";
6. mysql_connect($host, $user, $password);
7. mysql_select_db($database);
8. ?>
```