

Anuj Adhikari

# Full Stack JavaScript: Web Application Development with MEAN

---

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

July 10, 2016



Author(s)	Anuj Adhikari
Title	Full Stack JavaScript: Web Application Development with MEAN
Number of Pages	40 pages
Date	10 July 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Principal Lecturer
<p>The thesis was carried out as a research project on the topic Full Stack JavaScript. The main objectives of the thesis were to study the different components of the MEAN stack, the most popular Full Stack JavaScript framework, and develop a prototype web application based on it.</p> <p>A considerable amount of time was invested in examining the different web technologies. Finally, the study concentrated on the Full Stack JavaScript, and the MEAN stack in particular. All the components of the MEAN stack were studied in great detail. Moreover, the strengths and weaknesses of all the components were analysed in comparison with other web stacks, and a prototype web application was developed using the MEAN stack.</p> <p>As a result, a Single Page Application based on the MVC architecture was developed, and a REST web service was built to provide data to the application. The application demonstrates the practical implementation of the MEAN stack in a real-life project. The application development process provides a step by step guide to the installation of all the components and the related tools. In addition, it suggests the best practices for building a MEAN application.</p> <p>The thesis concludes that the MEAN stack is one of the best solutions to develop modern web applications, and can be used in a wide range of web applications. However, the instability and immaturity of the stack make it inappropriate in certain contexts. Therefore, further studies and implementations can be done to explore the usefulness of the stack in different contexts.</p>	
Keywords	MEAN stack, Node.js, Express, MongoDB, AngularJS, REST API, SPA, HTTP, MVC

## Contents

1	Introduction	1
2	Web Application Development	2
2.1	Front-end and Back-end	2
2.2	RESTful API	4
2.3	Single Page Application	6
3	Full Stack JavaScript	8
3.1	JavaScript	8
3.2	Rise of Full Stack JavaScript	9
4	MEAN Stack	11
4.1	Node.js	11
4.2	Express	15
4.3	MongoDB	16
4.4	AngularJS	20
5	Implementation of MEAN Stack	23
5.1	Project Description	23
5.2	Development Environment Setup	25
5.3	Implementation	27
6	Results and Discussion	34
7	Conclusion	40
	References	

## Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CLI	Command Line Interpreter
CSS	Cascading Style Sheets
DBMS	Database Management System
DOM	Document Object Model
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
JSON	JavaScript Object Notation
MEAN	MongoDB Express AngularJS Node.js
MVC	Model View Controller
ORM	Object Relational Mapping
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SPA	Single Page Application
SQL	Structured Query Language
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language

## 1 Introduction

With the recent evolution in web technologies, there has never been a more exciting time for developers and technologists around the globe to build modern web applications. Web development is not anymore confined in the realm of pure HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript on the front-end, and PHP/Perl on the back-end. There is a plethora of new languages, web frameworks and tools to choose for any web application development. Although the rise of web technologies has helped to ease the application development process, it has created confusion among developers to select a perfect technology stack to start with.

For any web application development, it is important to choose a correct technology stack which allows rapid prototyping, constant iteration, code reuse, maximum efficiency, and robustness. It is also important that the technology stack is easy to learn and understand by the developers working on the front-end and the back-end. Thus, the concept of Full Stack JavaScript was developed. Originally web development was based on the LAMP (Linux, Apache, MySQL, PHP/Perl) stack and Java (Java EE, Spring), which consists of different programming languages. JavaScript solved this multi-language paradigm by introducing MEAN (MongoDB, Express, AngularJS, Node.js) stack which is based on a single language ‘JavaScript’. [1.]

This thesis was carried out as a research project on the topic ‘Full Stack JavaScript’. The main objective of the thesis was to study the different components of the most popular Full Stack JavaScript framework, MEAN stack, and build a prototype application based on it. The prototype application was developed only to illustrate the implementation of the MEAN stack, so the report focuses more on the practical implementation of the MEAN stack’s components in the application than the application itself. Furthermore, the report analyses the strengths and weaknesses of the MEAN stack, and suggests the context where to use it and where to avoid it. The report also suggests the best design principles and architecture to follow when developing a MEAN stack web application.

## 2 Web Application Development

### 2.1 Front-end and Back-end

Web application development is the combination of the front-end and the back-end development. Front-end web development, also known as client-side development, involves the practice of creating Graphical User Interface (GUI) for clients (users) so that the users can interact with the application. It involves the use of primary web technologies and tools such as HTML, CSS, and JavaScript. [2.]

HTML is a mark-up language which provides the structure to a web page. It defines how a web page would look like so it can be considered the skeleton of any web application. CSS, on the other hand, is a style sheet language which provides style and visual enhancements to the documents written in HTML. JavaScript is the most advanced language among these technologies. It performs HTML DOM (Document Object Model) manipulation to provide a dynamic interface to users. Moreover, it provides an interactive interface to the users by creating pop-up messages, validating form inputs, and changing the layout based on events like user-input or mouse clicks. All these technologies are controlled by the browser to provide a front-end web interface. [3.]

Back-end web development, also known as server-side development, involves the development of computer programs and databases to serve the client. A web application in its primary days did not need to have a front-end but a functioning server-side application was enough for it to be considered a web application. Several changes have been made in this field since then. Today's sophisticated web applications cannot run without both the front-end and back-end services. Back-end technologies usually consist of the programming languages such as PHP, Ruby, Python, Java, Node.js, and different frameworks.

Web application development encompasses all the web technologies related to the front-end and the back-end. In addition, a web application must ensure communication between the client and the server with the use of different communication protocols. Protocols are a set of rules for exchanging messages in a communication network. Protocols vary with different tasks and layers. HTTP (HyperText Transfer Protocol), TCP/IP (Transmission Control Protocol/ Internet Protocol), FTP (File Transfer Protocol), SMTP (Simple

Mail Transfer Protocol), SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are some common examples of the protocols used in web applications. [4.] A web application, in its most elementary form, sends an HTTP request to a server to establish connection, and the server sends an HTTP response to the client. A typical example of communication in a web application is illustrated in figure 1.

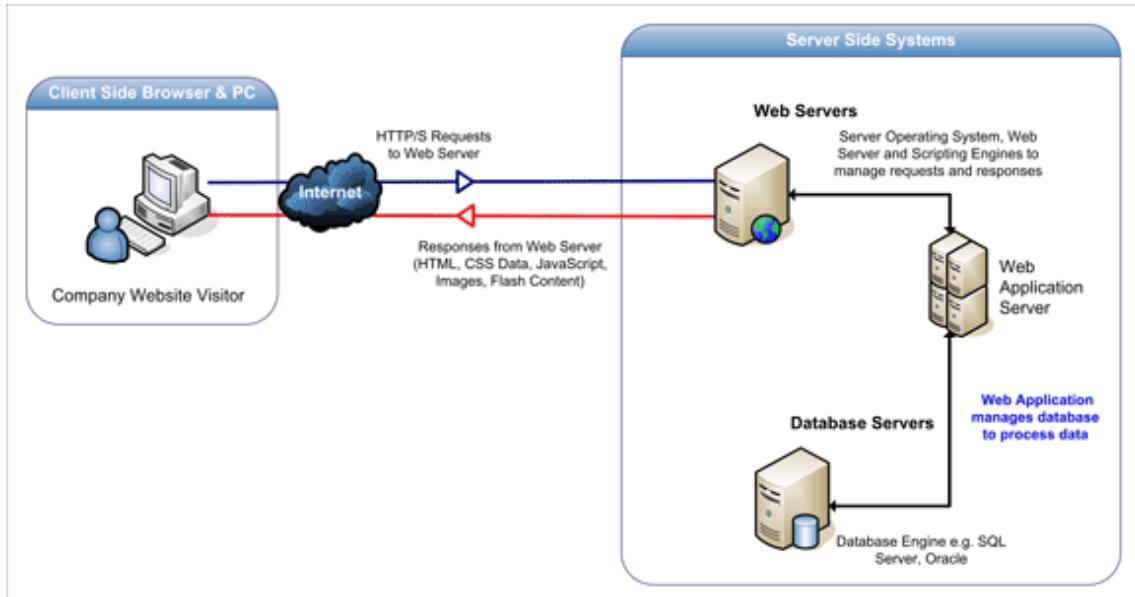


Figure 1. Web application model [4]

Figure 1 illustrates the communication among the three layers of a web application model. The first layer is a client-side web browser, the second layer is a server-side dynamic content generator, and the third layer is a database server. A user sends an initial request using the HTTP protocol through the browser over the internet to the server. The web server then processes the request by accessing the database server and retrieving the requested data. The web server then sends the response to the user over the internet through the browser. The response usually contains the data requested by the user. [4.]

Building a good Database Management System (DBMS) to store information is a crucial part of web application development. DBMS allows the users to create, save, update, and query (search) data in a web application. There are two types of databases: relational and non-relational. Relational databases, also known as SQL databases, are traditional databases which store data into tables in the form of rows and columns. The

tables maintain some kind of relation with each other, thus giving it the name ‘relational database’. Oracle database, MySQL, and SQL Server are the popular relational databases. [5.]

Non-relational databases, also known as NoSQL databases, store data in forms other than tables. NoSQL databases store data in a key-value pair, graph, document, column, or multi-model depending on the database selected for application development. MongoDB, HBase, Cassandra, Redis, and Riak are some of the examples of popular NoSQL databases. Relational databases are generally used in enterprise applications which can handle large related data, whereas, non-relational databases, which are flexible and scalable in nature, are used in data-driven real-time web applications with rapidly increasing data. [5.]

In addition to a good database, a good framework for web application development has also become a necessity nowadays. Use of web frameworks in the development of sophisticated web applications is more in practice than the use of native programming languages. Web frameworks, also known as web application frameworks, are designed to ease the task of web application development by providing database-access libraries, templates, session management, and code reuse facility. JavaScript frameworks such as AngularJS, ReactJS, Backbone.js, Ember.js, and Knockout.js are the most popular frameworks in the front-end development. Whereas, PHP frameworks (Laravel, Symfony, CakePHP), JavaScript frameworks (Node.js, Meteor, Express), Rails, Pyramid, ASP.NET, Java EE, Spring, and Django are the most popular frameworks in the back-end development. [5.]

## 2.2 RESTful API

REST is an architectural style used in web development in order to create web services. REST only defines the principles on which a web service is developed for the client-server communication. It is not a set of rules (protocols) for creating web services. Any web services or APIs (Application Programming Interfaces) that are designed with the REST architecture are called RESTful APIs, or just REST APIs. REST provides good performance, scalability, and reliability in a distributed computing system. There are several constraints for an application to become a REST application. However, a concrete implementation of REST APIs must follow at least four basic design principles: [6.]

- Use of HTTP methods: REST APIs must follow the HTTP methods explicitly. They must use GET to retrieve a resource from the server, POST to create a resource, PUT to modify or update a resource, and DELETE to delete a resource.
- Stateless communication: Communication between the client and the server must be stateless, meaning that every request from the client must contain all the information required for the server to process them. The server should not require any stored data to process the request.
- Use of directory-structure like URLs: REST APIs must use the URLs that are straightforward, properly structured, and easily understood, such as `http://www.myservice.org/discussion/topics/{topic}`
- Data transfer in XML or JSON: The data transferred between the client and the service-exchange must be in XML or JSON format so the data are properly structured and readable. [6.]

In addition, REST web services (APIs) must have a clear separation of client-side logic and server-side logic. A uniform interface separates clients and servers, which allows developers to work on the individual part of web application and improve one without affecting another. Clients and intermediaries should be able to cache server responses to avoid reuse of stale data in response to future requests. Clients also cannot assume a direct connection to the server. In most cases, intermediaries between the client and the server serve the request-response cycle. Furthermore, in REST web services, server may temporarily extend the client by transferring logic to the client. [6.]

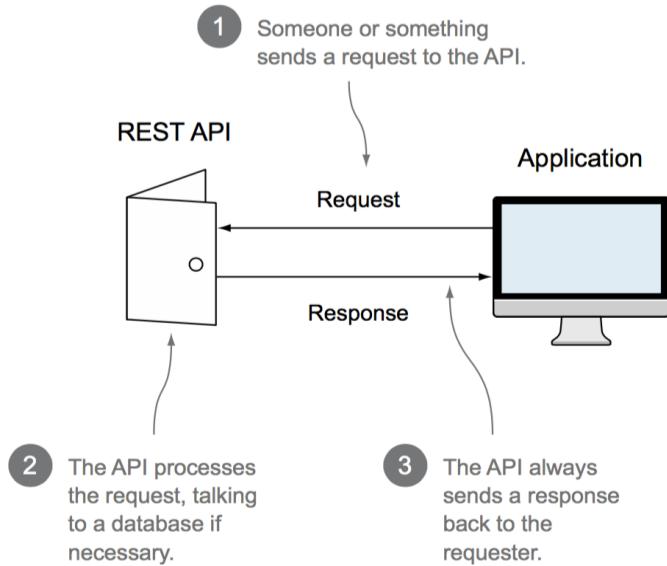


Figure 2. REST API [9]

Figure 2 illustrates the communication between a client-side application and a REST API. The client does not know the implementation of the server and does not directly communicate with the server. The REST API takes all the incoming HTTP requests from the client, processes them, and sends HTTP responses [9,162].

### 2.3 Single Page Application

Single Page Application (SPA) is a web application which fits into a single web page. In contrast to the traditional full page loads, an SPA loads all the resources required to navigate throughout the web application on the first page load. It then dynamically changes the contents as the user interacts with the application, so no full page request will ever be made again. However, URLs are updated in the address bar of the browser with a hash tag following the name of the resources accessed. [22.]

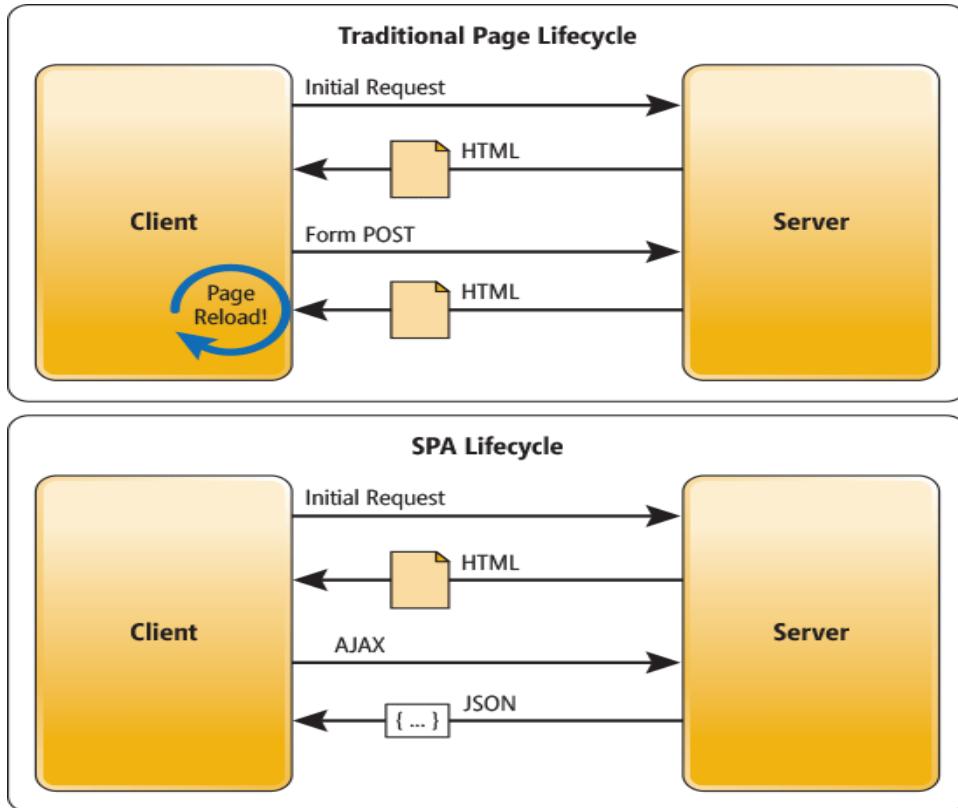


Figure 3. Comparison of traditional page lifecycle and SPA lifecycle [24]

Figure 3 illustrates the distinction between the lifecycle of a traditional web page and an SPA web page. In traditional web applications, every time when a client sends a request to a server, the server will respond by rendering a new HTML page. All the subsequent requests to the server are also processed in a similar fashion and every time a new page is loaded on the client's browser. In an SPA application, after the first page loads, all interactions between the client and the server happen with the AJAX (Asynchronous JavaScript and XML) calls, which in return send data in JSON (or XML) form. The browser then uses the JSON data to update the page dynamically, without any page reloads. [24.]

SPAs use AJAX (to interact with the server), HTML templates, MVC frameworks, and JavaScript to perform most of the navigation works on the front-end [23,10]. Modern front-end JavaScript frameworks such as AngularJS, Ember.js, React, and Meteor.js have simplified the tasks of creating SPAs by providing rich DOM manipulation and two-way data binding features. SPAs provide a rich interface and fluid user experience. Moreover, SPAs make users feel like they were interacting with a desktop application.

### 3 Full Stack JavaScript

#### 3.1 JavaScript

JavaScript is a cross-platform, light-weight, object-oriented scripting language designed primarily for adding interactivity to modern web pages and web applications. It is one of the three main components of the front-end technologies besides HTML (providing structure to the web page) and CSS (providing style to the web page). It was developed by an American programmer Brendan Eich at Netscape in 1995. Originally called by the names 'Mocha' and 'LiveScript', Javascript got its current name in December 1995 with the release of the third beta version. [25.]

JavaScript was submitted to ECMA (European Computer Manufacturer Association) for standardization so that other browsers could implement it on the basis of the standardization. ECMA international then published the standard JavaScript language specifications under the name ECMAScript. The current JavaScript specification is based on the ECMAScript 2016, which was released in June 2016. [25.]

JavaScript provides rich features to create dynamic and interactive interfaces in client-side applications. It can detect the user's browser and OS (Operating System), and performs platform-specific operations. Originally termed as an interpreted language, JavaScript can be executed without preliminary compilation by the browser so it can perform simple calculations on the client-side. It is often used to validate the user's input in forms and send the data asynchronously with the AJAX. It also performs HTML DOM manipulation to provide dynamic interface. It is an object-oriented programming language and supports several built-in objects with inheritance. Thus, JavaScript enables developers to add dynamic features to static HTML pages, control multimedia and add animations. Moreover, JavaScript provides several third-party APIs and libraries to facilitate the task of building dynamic web pages. One of the most popular JavaScript library is JQuery. [25.]

JQuery is a library based on JavaScript, developed by an American programmer John Resig. All the functions that can be programmed in traditional JavaScript can be performed in JQuery, but with much simpler methods. Therefore, a function that needs several lines of codes to be written in JavaScript can be easily written in few lines using JQuery. It is almost supported by all modern browsers and does not require extra plugins or extensions to run. One of the main features of JQuery is DOM manipulation. DOM is

a tree-like structural representation of the elements in an HTML page. JQuery syntax makes it easy to navigate throughout these elements, find and manipulate them. For example, it can be used to find all elements with a certain property in a web page, change the elements' properties or make them respond to certain events like mouse clicks. This task is difficult to perform in pure JavaScript but JQuery's syntax makes it easier. [26.]

For example, in order to change the background colour of the body in an HTML page, one writes the following function in JavaScript:

```
function changeBackground(color) {
    document.body.style.background = color;
}
onload="changeBackground( 'red' );"
```

whereas, the same task can be performed in one line of code using the following JQuery function:

```
$( 'body' ).css( 'background' , '#ccc' );
```

Moreover, JQuery provides simpler functions to create animations, handle different events, and develop AJAX applications.

AJAX (Asynchronous JavaScript and XML) is a web development technique to create asynchronous web applications. With the use of AJAX, asynchronous communication can be made between the client and the server. AJAX uses XMLHttpRequest object to communicate with the server, and sends and receives information in XML, JSON, HTML, or even in text file format. XMLHttpRequest is an API that allows asynchronous communication between the client and the server without the need of a full page reload. Hence, AJAX allows to update a certain portion of a web page without page refresh (reload). [27.]

### 3.2 Rise of Full Stack JavaScript

JavaScript in its primary days earned a bad reputation because of the lack of performance and its incompatibilities with the prominent browsers of that time. However, things started to change after large browser vendors invested time and money in improving the language. JavaScript, finally, became the de facto standard of the client-side scripting. While JavaScript remained prominent in the client-side, several new technologies were introduced in the server-side such as PHP, JAVA, .NET, Ruby, and Python. Developers

started to realize that use of two separate languages in the development of the client and the server was complicating the tasks of web programming. Several attempts were made to unify the two sides by creating client components on the server and compiling them into JavaScript (for example: ASP.NET web forms and GWT), but they failed. The only solution to this problem was the implementation of JavaScript on the server-side, and Node.js was introduced. [28.]

Node.js is actually the backbone of Full Stack JavaScript web development. It finally put the power of JavaScript on the server with the idea of non-blocking programming paradigm. Node.js became popular in a short time due to its easy-to-use components. It allowed the developers to quickly set up a server and start building applications on top of it. Several frameworks started to emerge to facilitate Node.js implementation such as Express and Connect.js. Express became the most prominent one. Node.js ecosystem continued to expand and a package manager like 'npm' was introduced. [28.]

While Node.js was invading the server, NoSQL database started to gain popularity in the field of database. MongoDB, a NoSQL database, was introduced with the concept of storing data in Binary JSON. Due to the use of JSON, which is a JavaScript way of storing data, MongoDB became the preferred database for Node.js applications. JavaScript also started to evolve on the client-side and new frameworks like AngularJS, Backbone.js, and ReactJS started to take over the use of traditional JavaScript, JQuery, and AJAX for building Single Page Applications. When all the prominent tools to build a Full Stack JavaScript application were ready, additional tools based on JavaScript were introduced to improve the development process such as Mocha.js and Chai.js for application testing, Gulp.js and Grunt.js for automation of build tasks. [15; 28.]

Node.js eco-system, with addition of AngularJS on the front-end, Express as the back-end framework, and MongoDB as database raised a new term in web application development called MEAN (MongoDB, Express.js, AngularJS, Node.js) stack. The word MEAN stack is used to refer Full Stack JavaScript but it is still a subset of the term Full Stack JavaScript. Node.js, Express, and MongoDB are the prominent members of the Full Stack JavaScript. AngularJS, however, can be replaced with Backbone.js, ReactJS, or Ember.js to meet the developers' requirements. Nevertheless, the term Full Stack JavaScript is generally used to refer to the MEAN stack. [15; 28.]

## 4 MEAN Stack

### 4.1 Node.js

Node.js is a software platform which helps to build asynchronous and event-driven network applications. It contains built-in HTTP server libraries which allow developers to create their own web server and build highly scalable web applications on top of it. The V8 JavaScript runtime engine used by Node.js is the same engine used in Google's Chrome browser. [7.] The V8 engine complies the codes directly to the native machine code leaving out the interpreter and byte code execution process, which gives Node.js a huge boost in performance [8]. In addition to the V8 engine, Node.js is composed of several components as shown in figure 4.

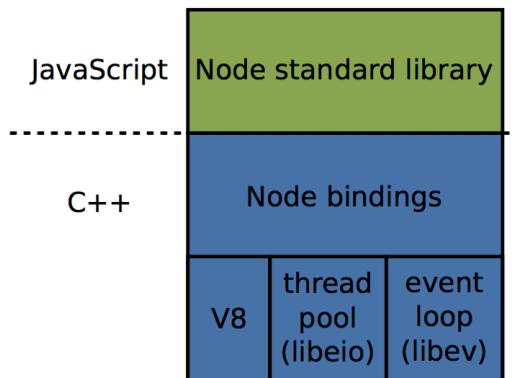


Figure 4. Node.js architecture [8]

Figure 4 illustrates the architecture of Node.js. Node.js is composed of Node standard library at the top, thin C++ node bindings in the middle, and V8 engine, libeio and libev at the bottom. Node standard JavaScript library exposes operating system features to the application, while the C++ bindings expose the core APIs of the underlying elements to JavaScript. The V8 engine provides the run-time environment for the application, and libeio handles the thread pool to make asynchronous (non-blocking) I/O calls to libev, the event loop. [8]

The asynchronous, non-blocking I/O feature of Node.js plays an important role in resource management and performance enhancement of Node.js applications. Unlike other mainstream servers like Apache and IIS, Node.js uses single-threaded, non-blocking I/O operation. What this means is that instead of running each session (request) in a

separate thread and providing an associated amount of RAM for each session, Node.js uses a single thread to execute all requests and implements an event loop to avoid blocking of I/O. In a multi-threaded server, as the number of threads increase, the server overhead (scheduling and memory footprints) increases resulting into a slow performance of the overall system. Node.js uses an event-driven and non-blocking I/O approach to handle all the requests to the server. [10;11.]

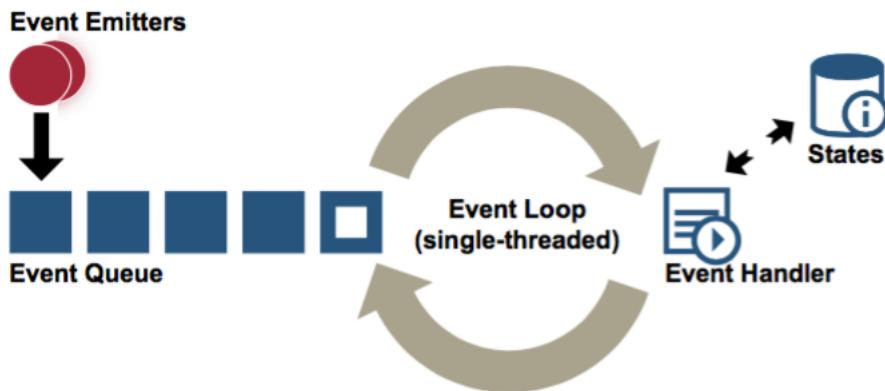


Figure 5. Node.js processing model [10]

In figure 5, Node.js creates an event-loop with event handlers for all requests. When an I/O operation occurs, the associate handler is queued up for execution and a callback function emits an event after the I/O operation is completed. In the mean-time, other I/O operations keep running outside the event loop of the server. Thus, Node.js performs the I/O operations asynchronously and does not block any script execution, allowing the event loop to respond to other requests. [10.] One common use of a callback function displaying the non-blocking I/O operation of Node.js is shown in listing 1.

```

var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
  if (err) return console.error(err);
  console.log(data.toString());
});
console.log("Reading file");
  
```

Listing 1. Node.js non-blocking code example

Listing 1 illustrates the non-blocking code example of Node.js. The program is trying to perform three operations: read a file 'input.txt', print the file's data in console, and print

message “Reading file” in console. The function `readFile()` executes first in the order of appearance but the program does not wait for file-reading function to complete. Once the function starts to read the file, it passes control to execute the next instruction immediately so the program prints “Reading file” before printing the file’s data. Once the file I/O is complete without any error, it will call the callback function `function (err, data)` and returns the file data as parameters, then prints the data below the ‘Reading file’ message in the console. Hence, there is no blocking of I/O operations and all the operations execute asynchronously.

Node.js has a rich development eco-system with several compatible libraries and package managers. One of the main package managers that comes pre-bundled with Node.js during installation is npm. Npm is run from the Command Line Interpreter (CLI) and it manages all the dependencies for the Node.js applications. Instead of manually downloading and configuring the JavaScript modules for use in the application, npm provides a simpler alternative. The names of the modules and dependencies can be included with their version numbers in ‘package.json’ file inside the folder structure, and the modules are downloaded issuing a simple ‘npm install’ command from the CLI. [12,37-42.]

The npm automatically handles all the downloading process and save all the named modules in the ‘node-modules’ folder. The modules can be updated by changing their version number in the ‘package.json’, and can be easily distributed by uploading a single ‘package.json’ file to the server instead of uploading the large ‘node-modules’ folder. After the update or upload, issuing the ‘npm install’ command installs the specific modules and dependencies. Therefore, Node.js applications are light-weight, flexible and easily sharable. [12,43-47.]

Node.js is actually the foundation of the MEAN stack. All the other technologies work on its foundation. Node.js introduces the power of JavaScript on the server side allowing the developers to create both server side and client side logic in one single language ‘JavaScript’. One of the major advantages of Node.js over other server-side platforms is the built-in event loop. The event loop is used by all the available modules and libraries in the Node.js which makes the I/O operations efficient. [10.] This Node.js feature maintains the speed and efficiency of the overall system.

Table 1. Performance benchmark of Node.js, Python-Web, and PHP [13]

Web development technology	Calculate value of Fibonacci	Mean requests per second [#/sec]	Mean time per request [ms]
Node.js	Fib (10/ 20/ 30)	2491.77/ 1529.4/ 58.85	0.401/ 0.654/ 16.993
Python-Web	Fib (10/ 20/ 30)	633.68/ 209.89/ 2.9	1.578/ 4.764/ 345.307
PHP	Fib (10/ 20/ 30)	2051.22/ 168.8/ 1.78	0.488/ 5.942/ 560.553

Table 1 illustrates the performance results of the Node.js in comparison with Python-Web and PHP when calculating Fibonacci (10/20/30). The study was conducted by the researchers in Peking University, China and results were published on the technical report by IEEE. It is clearly seen that the Node.js is better in handling requests and performing calculations than the other two technologies. When calculating the small Fibonacci of 10, Node.js and PHP, both perform well, whereas Python-Web lags behind. Node.js handles about 2500 requests per second taking 0.401ms time to handle each request. PHP falls short by only few numbers but the gap in their performances increases drastically when the calculation tends to become complex. When the task reaches to the calculation of Fibonacci of 30, both PHP and Python-Web perform poorly, almost processing 2 or 3 requests per second, and taking 345ms and 560ms to process each request respectively. In contrast to them, Node.js still maintains the quality by processing requests 20 times higher than PHP and taking a relatively short time, about 17ms. [13.]

Although Node.js is an ideal choice for a scalable, data-driven, I/O intensive applications, it is not a perfect solution for all applications. It uses JavaScript so it is more efficient when used with other JavaScript-based technologies. The use of single-thread to handle all requests is ideal for some cases but it is not a good feature for intensive data computing applications. Moreover, Node.js uses tight coupling between the web server and the web application so all the classes are dependent on each other. Such tightly coupled system is hard to maintain since a problem in one area causes the whole system to fail. Node.js also does not work well with the relational databases. All these factors must be considered before choosing Node.js as a development platform for any application. Studies have shown that Node.js is ideal for real-time data-driven web applications in collaboration with push technology and web sockets. [8.]

## 4.2 Express

Express is a node module which provides a minimal and flexible framework for Node.js web applications. It works on top of the core node modules without hiding any of the features of Node.js. In addition, it provides robust and clean functions to add to the node modules so the development of Node.js application using Express is far easier than using the native node modules. [14.] Due to the simplicity of Express, it has been adopted by large companies such as MySpace, Paypal, Apiary.io, Persona, and Ghost. The use of Express framework on top of Node.js helps to maintain clarity of the code. It also makes module integration easy to handle, and provides a solution structure for applications. [15, 142.] Express is installed using the npm package manager issuing “`npm install express`” command [14].

An Express server is made up of three building blocks: router, routes, and middleware. A web server’s core functionality depends on its excellent routing methods. In a client-server communication, a client requests some resources from the server, the server locates the resources and responds by sending the resources to the client. This is the core functionality of a web server and it requires excellent routing methods to serve the request. Express makes this tedious job really easy by allowing developers to create routes in simple structure. A route in Express is a combination of a HTTP verb and a path. The HTTP verb is generally one of the four HTTP methods: GET, POST, PUT and DELETE, and the path is the location of the resource (URI). [15,144-146.] A basic route in Express is created as below:

`app.METHOD (PATH, HANDLER)` where:

- app is an instance of express
- METHOD is an HTTP request method
- PATH is a route path (URI)
- HANDLER is the function which executes when the route is matched.

Middleware in Express are the functions that have the pattern `function (req, res, next)`. The `req` is the incoming request from the client, `res` is the response from the server, and `next` is the callback function. Therefore, middleware functions execute any code inside it, handle request and response objects, end request-response cycle, and call the next middleware function. A current middleware function must always call the next middleware function, even in the case of incomplete request-response cycle to

avoid request hanging. [14;15.] A simple Express web server containing all three building blocks is shown in figure 6.

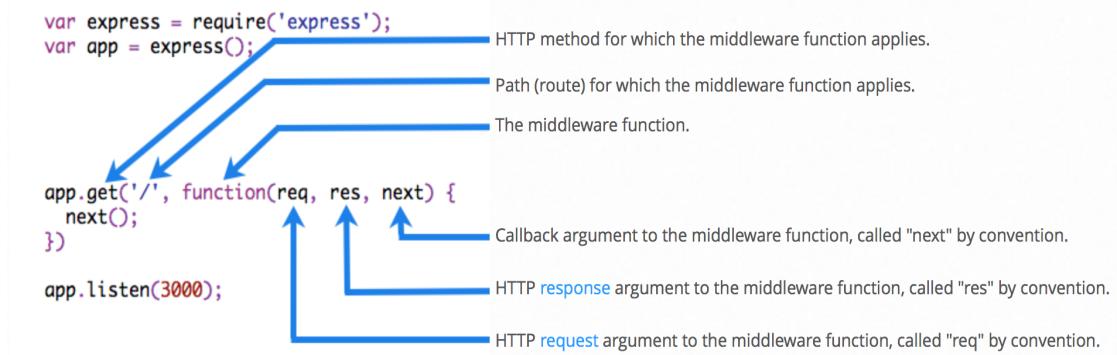


Figure 6. Express web server [14]

Figure 6 illustrates the three building blocks of an Express application: router, route, and middleware. The first two lines uses Node.js `require()` method to load express module in the application by creating the `app` object. The third line is a simple router with a route to '/' location and a middleware function. The application listens to port 3000 for any request.

Express also provides a simple application generator tool for providing structure to our Node.js application. It can be installed from the CLI by issuing '`npm install express-generator`' command. It also provides the options for creating template engine to write HTML codes. The application generator, by default, creates jade templates for the views, but it also supports Handlebars, EJS, Pug, and Mustache. The application structure created by the Express generator has separate directory for routes, views, and public-rendering files. However, the application structure is just one of many ways to structure an Express application. It can be easily modified during the application development process to meet the requirements of the application. [14.]

### 4.3 MongoDB

MongoDB is an open-source, non-relational, document database. It deviates from the need of creating Object Relational Mapping (ORM), for rapid application development. [16.] Unlike the relational databases, it does not contain columns and rows. However, the concept of rows still exists in MongoDB but it is called a document. The document

defines both itself and the data it contains. A document is a set of fields and it can contain complex data such as lists, arrays, or an entire document. Each document contains an ID field which can be used as a primary key for a query operation. A set of documents is called a collection and MongoDB holds a set of collections. The format in which the MongoDB stores the data is called BSON, which stands for binary JSON. Since JSON is the JavaScript way of storing data, MongoDB works perfectly with the applications built with JavaScript stack. [11,13-15.] A basic example of a MongoDB document is illustrated in listing 2.

```
{
  "firstName": "Homer",
  "lastName": "Simpsons",
  _id: ObjectId("52279effc62ca8b0c1000007")
}
```

Listing 2. Example of a MongoDB document

Listing 2 illustrates a code snippet from a MongoDB collection. The document stores the first and last names of a customer. Unlike traditional relational databases, MongoDB does not hold a data set corresponding to a set of columns, instead it uses the concept of name-value pair to store data [11,14]. The `_id` is the unique identifier (primary key) for that set of data (document). There are some variations in the naming terms of relational database and MongoDB, illustrated in table 2.

Table 2. Comparison of MySQL and MongoDB terms [17]

<b>MySQL</b>	<b>MongoDB</b>
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded documents and linking
Primary key	Primary key
Group by	Aggregation

As illustrated in table 2, in MongoDB, some MySQL terms like table is called collection, and row is called BSON document. Instead of Join operation, MongoDB embeds sub documents inside the main document and provides links to the sub documents. MongoDB, like MySQL uses unique identifier (primary key) for each document so that it is easy to query and find the data. MongoDB supports insert, query, update, and delete operations like any other databases. [17;11.]

One of the features that makes MongoDB stand out against the traditional databases is the inclusion of dynamic schema. Collections in MongoDB have different schema and the documents within the same collection can have as many different schema and shapes as required. This feature enables developers to start storing data in the database with any consideration of the database structural design. The documents' keys and values can be changed and updated when required since there is no pre-defined rule governing the data type validation. [18,7-12.]

Other important features of MongoDB are auto-sharding and replication. Since the data are stored in a document, they can be stored across multiple locations. As the size of databases increase, a single machine may not be able to store data and handle read-write operations. MongoDB solves this problem by allowing horizontal scaling, meaning that the data are distributed to multiple servers and all servers can work together to support data growth and provide efficient read-write operations. Likewise, the data can also be replicated to another data server so that the data are always available in case one server fails. [16.] This allows to build highly scalable and efficient data servers in comparison to relational databases such as MySQL and Oracle databases.

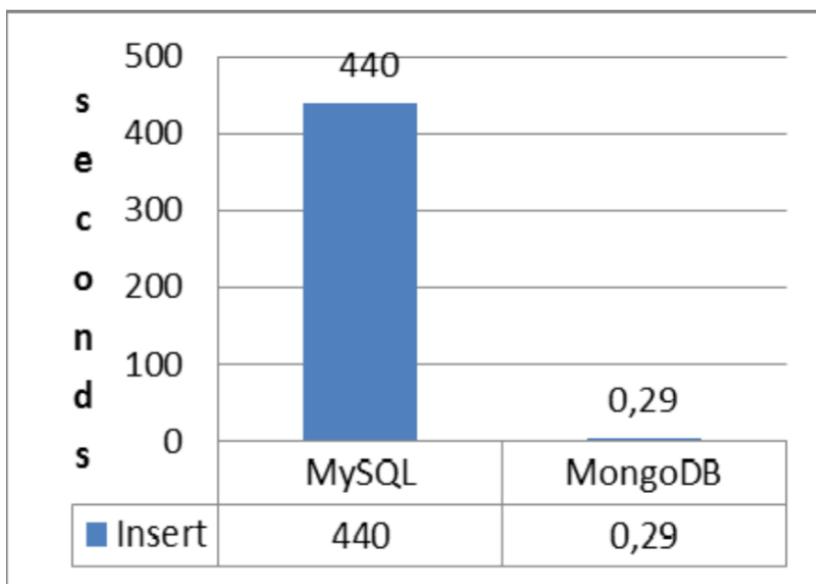


Figure 7. MySQL vs MongoDB insert [17]

Figure 7 illustrates the time taken by MySQL and MongoDB when running a script to insert 10,000 users' data. MySQL which uses traditional database approach took significantly longer time than MongoDB. It shows that MySQL took 440 seconds to insert 10,000 users, while MongoDB only took 0.29 seconds to perform the same task. It proves that the MongoDB is excellent in performing large read-write operations. [17]

Although MongoDB is good at performing majority of tasks, it has its drawbacks. MongoDB can not take multiple operations as one transaction. If any operation in one transaction fails, it will cause the entire operation to fail. It also can not perform the join operations like MySQL database so it is not a good choice in an application where there are multiple relationships among the data. The data has to be searched and updated in multiple documents at once and all operations need to be tied in a single transaction to ensure that the data is updated in all collections. Transactional databases work better in such cases than the non-transactional database like MongoDB. Although the flexibility of MongoDB to allow any sort of data is good in some cases, most applications still need certain kind of structure to their data to work properly. To solve this problem, Mongoose was created by the company behind MongoDB. [11,15.]

Mongoose is a MongoDB data modelling for Node.js. It was created to solve the problem of writing complex data validation, casting, and business logic in MongoDB. It provides

simple, elegant, data-modeling feature to the application. Using mongoose, one can define what kind of data can be in a document and what data must be in a document. In technical term, it provides data validation rules, query building functions, and business logic to the application data. Moreover, it provides an entire set of new features to use on top of MongoDB. It can manage the connections to MongoDB database, as well as read, write, and save data. It also allows only valid data to be saved in MongoDB by providing validation rules at the schema level. [11,15.]

#### 4.4 AngularJS

AngularJS is a JavaScript framework for building front-end of web applications. It is designed to build a Single Page Application with the introduction of MVC (Model View Controller) architecture to the front-end. [19.] AngularJS framework extends HTML functionalities by providing different elements and attributes, which help to build large web application with ease. It extends HTML with `ng-directives`, binds the HTML input controls to the application data with `ng-app` directive, and binds the application data to the view with `ng-bind` directive. [21.] A simple HTML template with AngularJS is shown in listing 3.

```
<html lang="en-US">
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>
</body>
</html>
```

Listing 3. AngularJS example [21]

In listing 3, AngularJS is added to the page with a `<script>` tag so AngularJS starts immediately when the page is loaded. The actual implementation starts inside the `<div>` container where AngularJS is initialised by `ng-app` directive. The `ng-model` directive then binds the value of input to the variable ‘name’, and finally `ng-bind` directive binds `<p>` element’s innerHTML to the variable ‘name’. Hence, the name typed by the user in

the input field is displayed inside the `<p>` element. It is common practice to use AngularJS expression instead of `ng-bind` directive. For example in the above case, `<p ng-bind="name"></p>` can be replaced with `<p>{{name}}</p>`. [21.]

One of the important features of AngularJS is its two-way data binding mechanism. The two-way data binding feature allows synchronization of data between the model and the view. Model refers to some JavaScript variables, and view is the HTML container where the model data is displayed. Whenever data in the model changes, the view will be updated immediately, and whenever the view component changes, the model data will be updated as well. This feature eliminates the need of tedious DOM manipulation in order to find and change the data in the DOM tree. [20,2.]

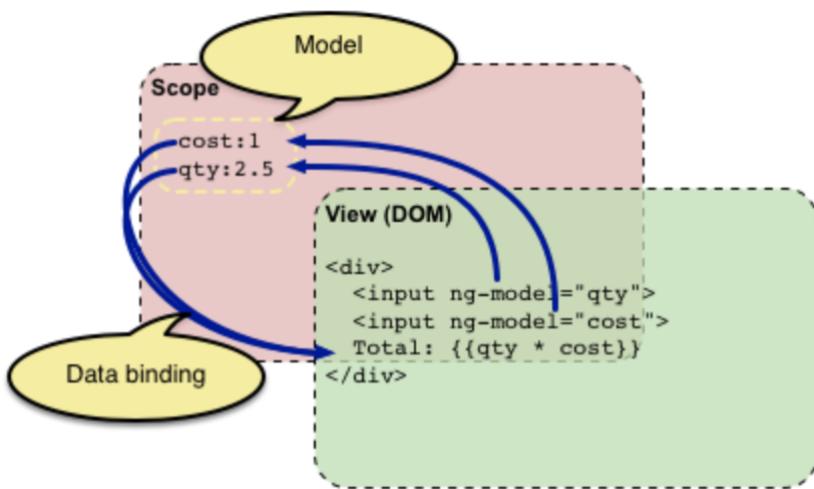


Figure 8. Two-way data binding in AngularJS [19.]

Figure 8 illustrates the two-way data binding feature of an AngularJS application. The view contains two text box inputs: cost and qty with `ng-model` attributes attached to them. The `ng-model` binds the input values to the scope that holds the model data. In one-way data binding, data is bound in only one direction. Once the view is rendered, any changes in the model data or view components are not automatically reflected in the view. If a user changes the values of cost and qty to get a new total, the total will not be automatically updated in the view. In AngularJS application, if the user changes the values of cost and qty, the new total will be displayed immediately in the view because of the live view provided by the two-way data binding. [19.]

Another important feature of the AngularJS is the introduction of MVC architecture on the front-end. In front-end web development, view and controller are generally placed in one place. Because of the automatic synchronization of model and view in AngularJS, controller can be kept outside of the view as shown in figure 9. [21.]

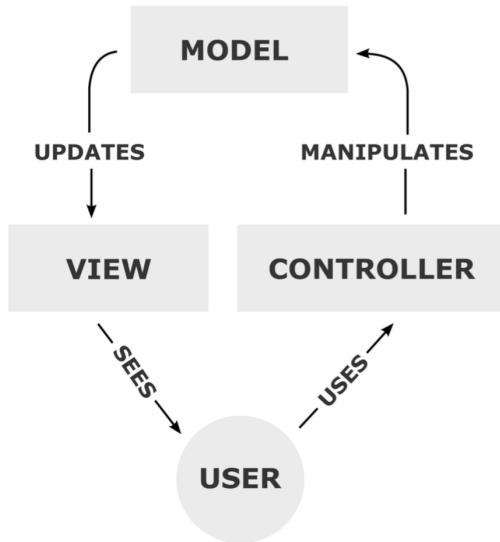


Figure 9. MVC architecture in AngularJS [15]

Figure 9 illustrates the MVC architecture in AngularJS. The model defines the application at a data layer, the view provides the visual presentation, and the controller manipulates the model data. Due to the two-way data binding between the model and the view, the controller focuses only on the model data, and the view reflects back any changes done in the controller from the model. [15,190-191.]

AngularJS was created with the concept of SPAs from its early days of development, so it provides excellent routing support to create SPAs. One can create multiple views for multiple URLs and AngularJS loads the appropriate view in the main view according to the URL requested. Since there are no redirects, AngularJS takes a minimal amount of time to load the pages. This feature provides good user experience. [20,3.]

AngularJS applications are also embeddable, testable, and injectable. They can easily be embedded with other applications and work perfectly with other technologies. Dependencies are automatically handled by the AngularJS injector subsystem by using different services and factory methods. This allows to create loosely coupled application, so individual components of the application can be tested in isolation. [20,4.]

## 5 Implementation of MEAN Stack

### 5.1 Project Description

Learn.io is the name of the prototype web application that was developed to illustrate the implementation of the MEAN stack in a real-life project. The web application provides minimalistic features of an online teaching platform. An admin can create courses, provide course descriptions, embed course videos, provide tutorials and other course materials. A user can simply browse the courses and request for full course videos. The application provides a basic foundation of an online teaching platform which can be extended later as a full e-commerce website to provide paid courses and tutorials.

After analysing the requirements, the following interfaces were implemented.

- An interface for admin to create courses, add description, embed video URLs, and other course materials.
- An interface for user to browse courses, view courses' teaser videos, and request the full video playlists providing the email address.
- An interface for admin to manage the courses, users and requests.

In addition to the above requirements, the application should also meet the following best practices set by the standard MEAN application:

- The application should be built with the MVC architecture.
- The application should be a Single Page Application (SPA).
- The application should consist of REST APIs for data communication.
- The application should follow the responsive mobile-first design principles.

Based on the requirements, the application was built in the MVC architecture. There is a complete separation between the server-side and the client-side logic. The model is implemented in the server, and the view and the controller are implemented in the client. Moreover, the application does not communicate directly to the server, but uses REST APIs to feed the data. The application uses the routes created in AngularJS to navigate throughout the web pages but all data are served by updating a single page. This SPA-REST API approach and the MVC architecture of the application are illustrated in figure 10 and figure 11.

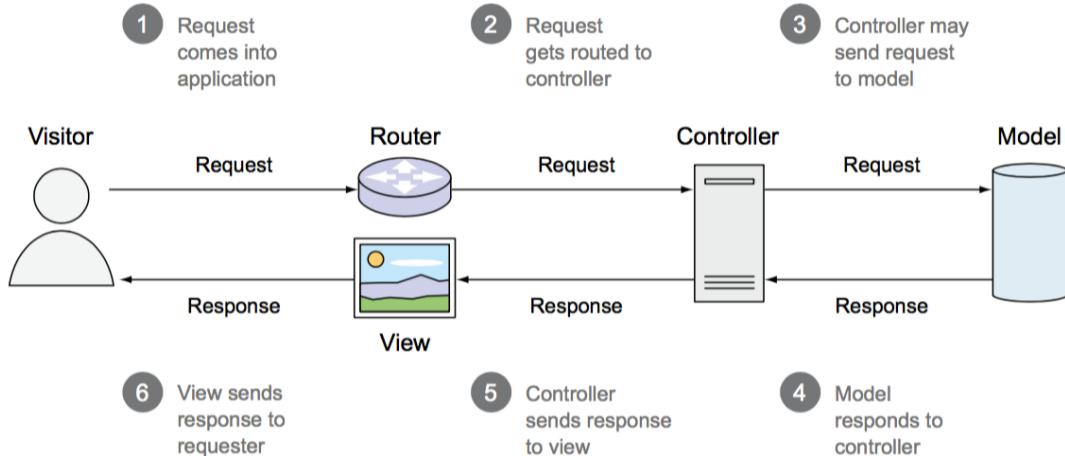


Figure 10. Request-response flow in MVC application [9]

Figure 10 illustrates the MVC architecture of the application. It also illustrates the routing mechanism the application uses to access specific web pages in the application. A request from a visitor is routed to the controller. The controller, if needed, makes a request to the model and the model responds to the controller. The controller then sends a response to the view and the view renders the page on the visitor's browser.

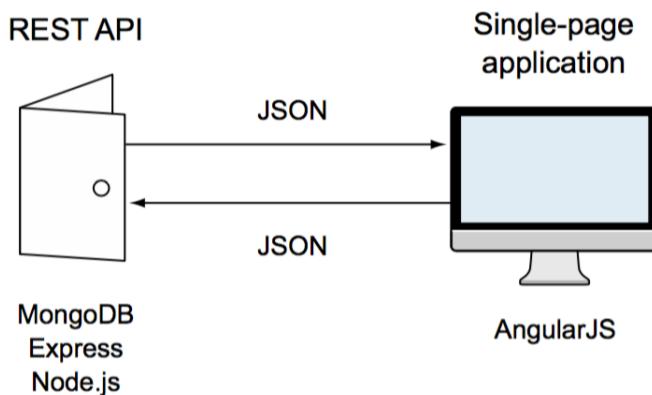


Figure 11. MEAN stack architecture of learn.io [9]

Figure 11 illustrates a common MEAN stack architecture that the application follows. A REST API is built using MongoDB, Express and Node.js, and a Single Page Application (SPA) is created using AngularJS. The REST API then feeds JSON data to the AngularJS SPA on the browser.

## 5.2 Development Environment Setup

The application was developed in Macbook pro 2013 running the operating system OS X El Capitan 10.11.6. Besides the hardware, several programs and tools were downloaded and configured to set up an environment for the MEAN application.

### WebStorm IDE (Integrated Development Environment)

WebStorm IDE is a lightweight and powerful IDE which facilitates the development of a MEAN application. It has built-in git support and embedded CLI. It has also built-in support for the latest technologies such as Node.js and AngularJS. It can be downloaded from the official website: <https://www.jetbrains.com/webstorm>.

### Node.js

Node.js is the first program to be downloaded since it provides the platform on which a MEAN application is built. It can be downloaded from the website: <https://nodejs.org/en/download>.

After the download and installation, it can be checked by issuing node –v command from the terminal or CLI as illustrated in figure 12.

```
[helios:~ anuj$ node -v
v4.5.0
helios:~ anuj$ ]
```

Figure 12. Node.js version check

Figure 12 illustrates the command to view the Node.js version installed on the computer. The version used in the application is 4.5.0. It confirms that Node.js is properly installed in the development machine,

### Express

After the installation of Node.js, a root directory is made and inside the directory, express is installed issuing command from the terminal `npm install –g express`. An express

generator can also be installed to provide simple structure to the application. All the node modules and dependencies were installed using `npm install` command.

## MongoDB and Mongoose

MongoDB is installed from the official web site: <https://www.mongodb.com/download-center#community>. After locating the downloaded folder, MongoDB server can be started with simple command from the terminal: `mongod` as illustrated in figure 13.

```
[helios:~ anuj$ mongod
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] MongoDB starting : pid=1
085 port=27017 dbpath=/data/db 64-bit host=helios.local
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] db version v3.2.9
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] git version: 22ec9e93b40
c85fc7cae7d56e7d6a02fd811088c
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
1.0.2h 3 May 2016
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] allocator: system
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] modules: none
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten] build environment:
2016-09-22T23:10:35.340+0300 I CONTROL  [initandlisten]     distarch: x86_64
2016-09-22T23:10:35.341+0300 I CONTROL  [initandlisten]     target_arch: x86_64]
```

Figure 13. Starting the MongoDB server

As illustrated in figure 13, MongoDB can be started with the single command `mongod`. Other information about the MongoDB can also be seen in the figure. Mongoose was installed using the command: `npm install mongoose`.

## AngularJS, Angular Routes, JQuery, Bootstrap (front-end packages)

There are several ways to download the front-end packages. In this application, bower was used to download and configure the front-end package. Bower is the package manager for the front-end, whereas npm is the package manager for the back-end. All the modules and dependencies were downloaded using npm and bower.

## nodemon

nodemon is a utility that monitors the changes in the source code during development and automatically restarts the server. It is downloaded using the command: `npm install -g nodemon`, and implemented using the `nodemon` command.

### 5.3 Implementation

After the installation and configuration of all the required tools and programs, the development process was started.

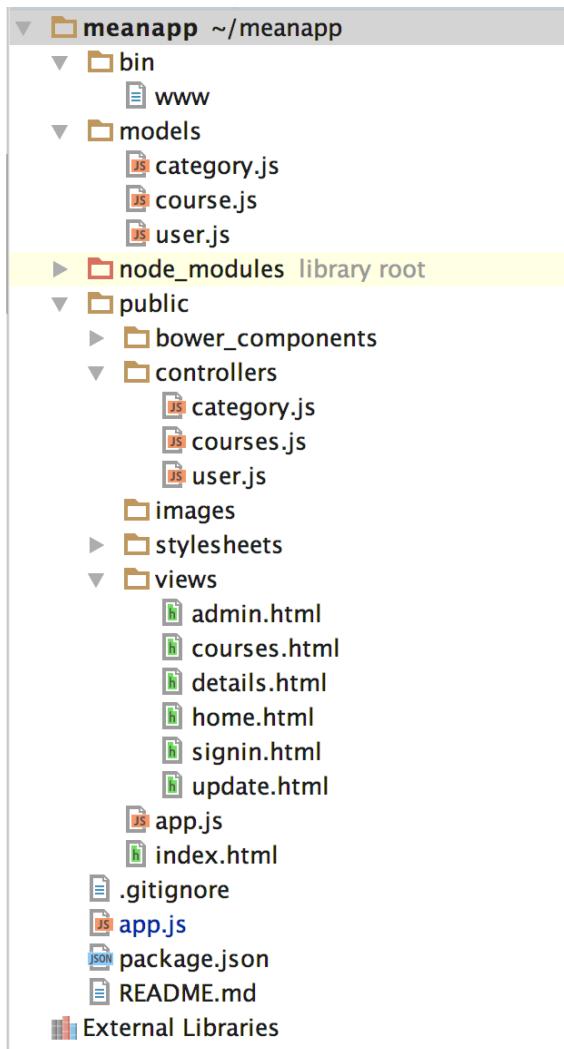


Figure 14. Application folder structure

Figure 14 illustrates an overview of the application's folder structure as seen in WebStorm IDE. The 'package.json' file contains all the dependencies required to run the application. The 'app.js' file (~/meanapp/app.js) contains all the Express middleware, and API routes of the application. The 'www' file contains the command to start server at port 3000. There is a clear division between the server-side logic and the client-side logic. All the client-side scripts are placed under the 'public' directory. The models, views and controllers are separated as per the MVC architecture. The data models are placed in

the server-side, whereas views and controllers are placed on the client-side. All the server-side dependencies are installed in ‘node\_modules’ folder and all the client-side dependencies including AngularJS are installed in ‘bower\_components’ folder.

### package.json

The ‘package.json’ file contains all the important information about the application and the dependencies required to build and run the application.

```

1   {
2     "name": "meanapp",
3     "version": "0.0.0",
4     "private": true,
5     "scripts": {
6       "start": "node ./bin/www"
7     },
8     "dependencies": {
9       "body-parser": "~1.15.1",
10      "cookie-parser": "~1.4.3",
11      "debug": "~2.2.0",
12      "express": "~4.13.4",
13      "jade": "~1.11.0",
14      "morgan": "~1.7.0",
15      "serve-favicon": "~2.3.0",
16      "mongoose": "*"
17    }

```

Figure 15. package.json file

Figure 15 illustrates the ‘package.json’ file’s contents which provides overall information about the application. It contains the application name, version, and all the dependencies. On line 6, the code shows that the application needs the scripts on /bin/www file to start the server and it can be started with the node ./bin/www command.

### app.js (Server-side)

The app.js file is the main configuration file of the Express application. It configures the server and imports all the modules to run the application.

```

1  var express = require('express');
2  var app= express();
3  var path = require('path');
4  var favicon = require('serve-favicon');
5  var logger = require('morgan');
6  var cookieParser = require('cookie-parser');
7  var bodyParser = require('body-parser');
8  var mongoose = require('mongoose');
9
10 Category= require('../models/category');
11 Course= require('../models/course');
12 User= require('../models/user');
13
14
15 // Connect to Mongoose
16 mongoose.connect('mongodb://localhost/meanapp');
17 var db = mongoose.connection;
18
19 //Use public folder for view
20 app.use(express.static(__dirname + "/public"));
21

```

Figure 16. app.js file

Figure 16 illustrate the contents of app.js file. On line 1, it imports the express framework to get started. It imports node modules for handling paths, serving favicon, logging requests and responses, and handling forms and cookies. Line 16 shows the connection to our MongoDB server, where ‘meanapp’ is the name of our database. Line 20 tells the application to use public folder for rendering views. The `app.use()` function tells the application to use the parameters given inside the brackets.

## Models

After creating the database, and configuring Mongoose and Express (See Figure 16, line 8, 16, and 17), model classes were created using Mongoose. A simple schema for the course is illustrated in figure 17 below.

```

var mongoose= require('mongoose');

//Courses Schema
var courseSchema = mongoose.Schema({
    title:{
        type: String,
        required: true
    },
    description:{
        type:String
    },
    price:{
        type:Number
    },
    pictureUrl:{
        type:String
    },
    demoUrl:{
        type:String
    },
    category:{type:String}
});

var Course = module.exports = mongoose.model('Course', courseSchema);

```

Figure 17. Model data of course

Figure 17 illustrates the model class for the course. The schema is designed using mongoose and it is exported for creating REST APIs.

## REST APIs

The routing mechanism in Express helps to create REST APIs for the application. Generally, in a MEAN application, REST APIs are built to feed the data to the client using Node.js, Express, and MongoDB.

```

app.get('/api/v1/courses', function(req, res){
    Course.getCourses(function(err, courses){
        if(err){
            throw err;
        }
        res.json(courses);
    });
});

app.get('/api/v1/courses/:_id', function(req, res){
    Course.getCourseById(req.params._id, function(err,
courses){
        if(err){

```

```

        throw err;
    }
    res.json(courses);
});
});

app.post('/api/v1/courses', function(req, res){
    var courses=req.body;
    Course.addCourses(courses,function(err, courses){
        if(err){
            throw err;
        }
        res.json(courses);
    });
});

app.put('/api/v1/courses/:_id', function(req, res){
    var id=req.params._id;
    var courses=req.body;
    Course.updateCourses(id,courses,{},function(err,
courses){
        if(err){
            throw err;
        }
        res.json(courses);
    });
});

app.delete('/api/v1/courses/:_id', function(req, res){
    var id=req.params._id;
    Course.removeCourses(id,function(err, courses){
        if(err){
            throw err;
        }
        res.json(courses);
    });
});
}
);

```

Listing 4. REST APIs for course

Listing 4 illustrates the code snippet for creating REST APIs for the courses. The HTTP methods GET, POST, PUT, DELETE were used to perform the various operations at the endpoints. The URI follows the path root-api/api/v1/ representing the version one of the REST APIs. When a user provides the route (URI) such as http://localhost:3000/api/v1/courses, the getCourses( ) function is invoked which will return all the courses in JSON if there is no error.

## Public directory (Client-side scripting)

The public directory contains all the files and scripts needed to provide the routes (URIs) and render HTML pages. The `ngRoute` module routes the Angular application to different pages without page reloads to make a Single Page Application. The routes are defined in the `public/app.js` file.

```
var app = angular.module('myApp', ['ngRoute']);

app.config(function($routeProvider,$locationProvider){
    $routeProvider.when('/', {
        controller:'CoursesController',
        templateUrl: '/views/home.html'
    })
    .when('/courses', {
        controller:'CoursesController',
        templateUrl: 'views/courses.html'
    })
    .when('/courses/details/:id',{
        controller:'CoursesController',
        templateUrl: 'views/details.html'
    })
    .when('/courses/add',{
        controller:'CoursesController',
        templateUrl: 'views/admin.html'
    })
    .when('/courses/edit/:id',{
        controller:'CoursesController',
        templateUrl: 'views/update.html'
    })
})
```

Figure 18. Routes in AngularJS

Figure 18 illustrates the routing mechanism in AngularJS. The routes refer to the pages to perform CRUD operations. The first line of code adds `ngRoute` as a dependency in the application module and then uses `$routeProvider` to configure the routes. The contents provided by the routes are put inside a `ng-view` directive in `index.html`. The `ng-view` directive then acts as a place holder for all the available views, thus creating a Single Page Application. When the user provides one of the routes, the related controller is invoked and the view is rendered inside the `ng-view` directive of the `index.html` without page reloads.

## Controllers in AngularJS

The AngularJS controller module is used to consume the REST APIs when the user provides the appropriate routes (URIs) and shows the results to the user.

```
myApp.controller('CoursesController', ['$scope', '$http', '$location', '$routeParams',
]  function($scope, $http, $location, $routeParams){
  console.log('CoursesController loaded...');

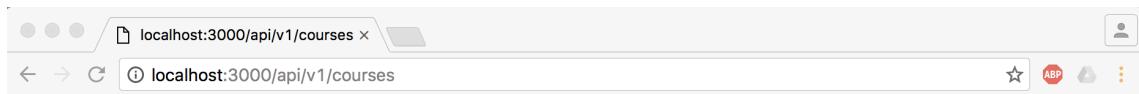
]  $scope.getCourses = function(){
]    $http.get('/api/v1/courses').success(function(response){
]      $scope.courses = response;
]    });
]  });
]
```

Figure 19. Course.js controller

Figure 19 illustrates a code snippet from the Course.js controller class. The code snippet illustrates the GET request method for all courses. The controller module uses the `$http.get` method to consume the REST service at the given URI (`/api/v1/courses`). It assigns the JSON returned back from the API to the `$scope.courses`, which sets a model object named `courses`. AngularJS then binds `courses` object to the application's DOM rendering it on the user's browser.

## 6 Results and Discussion

The implementation of the MEAN stack in real-life web application development was carried out successfully. The prototype application followed all the best practices of developing a MEAN application. A RESTful API was created on the server-side using Node.js, Express, and MongoDB, and a Single Page Application was developed on the client-side using AngularJS. Moreover, the application is responsive and fits well on all screen sizes.



```
[{"_id": "57d46b38e4e6b00cca483d6e", "title": "Node JS: What exactly is node.js?", "description": "Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. Although Node.js is not a JavaScript framework, many of its basic modules are written in JavaScript, and developers can write new modules in JavaScript.", "price": 2.99, "pictureUrl": "http://linkis.com/url-image/https://static-2.gumroad.com/res/gumroad/files/fd3dafaf14dcc48278207b168ca86693a/original/NodeProgram-text.jpg", "demoUrl": "pU9Q6oiQN0", "category": "Full Stack Javascript", "__v": 0}, {"_id": "57d5ae2f03423305c58d1d0b", "title": "Angular JS: Learn Complete in 7 days", "description": "Learn All", "category": "Full Stack Javascript", "price": 9.9, "pictureUrl": "https://i.ytimg.com/vi/qvHecQOiu8g/maxresdefault.jpg", "demoUrl": "pU9Q6oiQN0", "__v": 0}, {"_id": "57d5b4cc01ea4c05ead2cd8a", "title": "Beginning Machine Learning", "category": "Machine Learning", "price": 0, "pictureUrl": null, "demoUrl": "https://i.ytimg.com/vi/cKxRVEZd3Mw/hqdefault.jpg?custom=true&w=336&h=188&stc=true&jpgq=90&sp=68&sign=0690VU9Kedvy_XjHMujFz4ND2M", "description": "This course provides a broad introduction to machine learning and statistical pattern recognition. Topics include supervised learning, unsupervised learning, learning theory, reinforcement learning and adaptive control. Recent applications of machine learning, such as to robotic control, data mining, autonomous navigation, bioinformatics, speech recognition, and text and web data processing are also discussed."}]
```

Figure 20. RESTful APIs

Figure 20 illustrates the RESTful APIs built using the Node.js, Express, and MongoDB. Given the URI 'localhost:3000/api/v1/courses', the GET method renders all the courses' information on the browser.

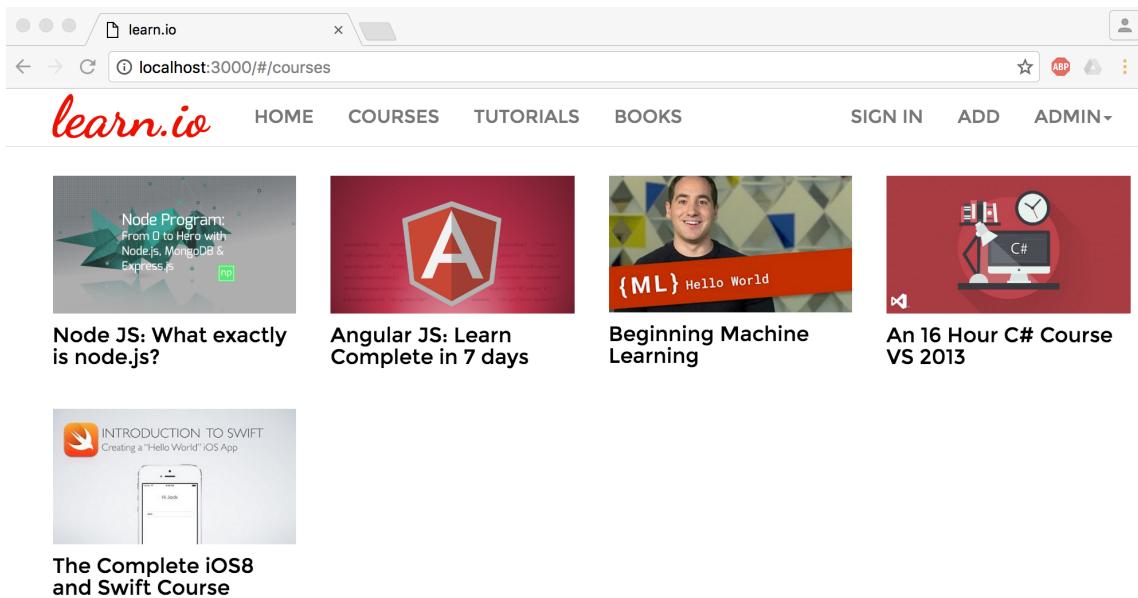
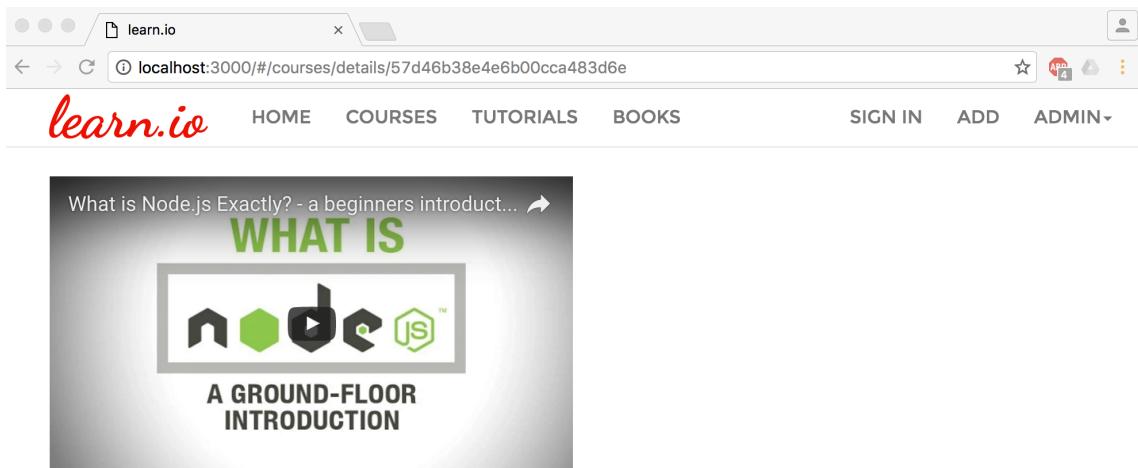


Figure 21. Course page

Figure 21 illustrates the course page of the application. The application has a menu for navigating to different pages but all the views are rendered inside the `ng-view` directive in the `index.html` file. The URL contains # (hash) sign, which indicates the SPA's routing mechanism. The application downloads all the resources necessary in the first request and keeps updating the views later.



## Node JS: What exactly is node.js?

Description: Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. Although Node.js is not a JavaScript framework, many of its basic modules are written in JavaScript, and developers can write new modules in JavaScript.

Price: 2.99€

Part of Full Stack Javascript Course

[Get full access](#)

Figure 22. Detail page of a course

Figure 22 illustrates the details of a course having the id shown in the URI. A user can browse the courses in the course page and click the link to go to the details. However, the page itself is not reloaded, the application will request the certain courses details from the REST API and JSON data sent back from the API is updated in the page. Using this interface, users can view the video and request for the full video playlist.

The screenshot shows a web browser window for 'learn.io' at 'localhost:3000/#/courses/add'. The page has a header with 'learn.io' logo and navigation links for HOME, COURSES, TUTORIALS, and BOOKS. On the left, there's an 'ADMIN PANEL' sidebar with a list of links: ADD COURSES, TUTORIALS, BOOKS, USERS, and REQUEST. The main content area is titled 'Add Courses' and contains fields for Title, Category, Description, Price, PictureUrl, and DemoUrl, each with a corresponding input field. A 'Submit' button is at the bottom.

Figure 23. Admin panel

Figure 23 illustrates a simple admin panel to perform the CRUD operations. The admin can manage the resources of the web application from this page.

The application was designed using Bootstrap's grid layout so the application follows the best responsive design principles.

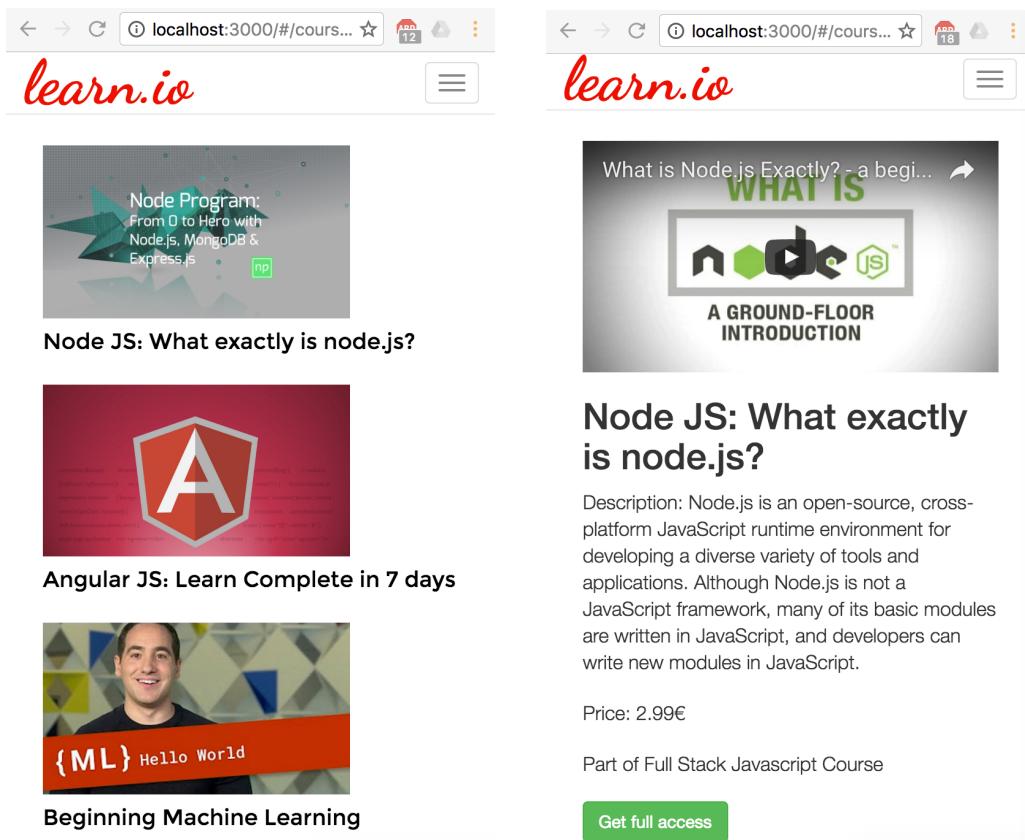


Figure 24. Responsive design

Figure 24 illustrates the responsive design of the application. Since the application was developed in a local environment, it was tested on a browser by minifying the width of the browser.

MEAN stack indeed proved itself as one of the best solutions to bring all the technologies together for modern web application development. The MEAN approach of creating RESTful APIs on the server-side and a Single Page Application on the client-side clearly separates the client and server implementation. Since the client does not communicate directly to the server, future changes on the client-side application will not affect the server. Similarly, hybrid applications and native applications can be developed and they can run perfectly having access to the RESTful APIs.

In spite of all the benefits, a few problems were faced during the application development. The first problem was storing the data in MongoDB. Since MongoDB uses a non-

relational database model, it was difficult to keep a relation between the data and perform related queries. In some cases, documents were embedded inside another document and in other cases, the references to other documents were included by using an array of referring object ids. These practices can lead to data repetition and unbounded arrays, which might cause a great problem in large enterprise applications with huge amounts of related data.

The second problem was with the initial loading time of the SPA. Since the SPA downloads all the resources at the first load, it took quite a long time for the application to load at first. The loading time might be longer if the web application uses several heavy graphics and multimedia. Although the development of REST APIs is good in long-term development, it is still an additional task while trying to develop a small web application within a short time. Instead, the client can directly access the data from the database. REST APIs should be implemented in large-scale web applications to minimize the coupling between the client and server, and to serve different types of clients (Web, Android, iOS, Hybrid). This loose coupling allows developers to update the server and client-side components independently in the future without affecting each other.

Since the application was developed to show the implementation of each component of the MEAN stack and was created from scratch, it took a good deal of time just to set up the working environment and directories. The application development process can be accelerated by using different seed projects such as mean.io and mean.js, which provide ready-to-use boilerplate codes and authentication. Similarly, scaffolding tools and code generators like yeoman can be used to simplify the development process.

## 7 Conclusion

The main goal of the thesis was to study the different components of the MEAN stack, the most popular Full Stack JavaScript framework, and to develop a prototype application based on it. A considerable amount of time was taken to do the research on the topic of Full Stack JavaScript web development, and MEAN stack in particular. All the building components of the MEAN stack: MongoDB, Express, AngularJS, and Node.js were studied in great detail. The strengths and weaknesses of each component were analysed, and were also compared with the other technologies.

After studying the theoretical aspects of the stack, the focus was shifted on the practical implementation of the stack. As a result, a prototype web application was developed with the MEAN stack. The propose of the prototype application development was to show the implementation of each component of the MEAN stack so the application would provide minimalistic functions of an online teaching platform. It can be further developed into a complete payment-based online learning application by creating a rich content management system, and integrating video streaming and payment methods.

Based on the research and practical implementation, it can be concluded that the MEAN stack provides one of the best solutions for modern web application development. The MEAN application follows the MVC architecture, encourages the development of REST APIs on the server-side and Single Page Application on the client-side. This approach makes a clear division between the server-side and the client-side logic, and facilitates the agile development. It allows to create scalable and maintainable applications using a single language.

However, MEAN stack is not the solution for all web applications. MEAN stack is not a good choice for developing enterprise applications which deal with a large amount of related data. Also, it is not suitable in the fields where security is the main concern. It is an evolving technology stack and lacks the stability and robustness of the old technologies like Java and PHP. Nevertheless, MEAN stack performs well in real-time web applications with rapidly increasing data. It is useful for rapid prototyping and quickly setting up a web server and web application for start-up companies. With all the JavaScript developers' community behind the stack, the stack is going to stay and evolve with time.

## References

- 1 The MEAN Stack on OpenShift [online]. OpenShift developers' official website.  
URL: <https://developers.openshift.com/languages/nodejs/example-meanstack.html>  
Accessed 2 June 2016
- 2 Dana Nourie. Java technologies for Web Applications [online]. Oracle official website  
URL: <http://www.oracle.com/technetwork/articles/java/webapps-1-138794.html>  
Accessed 2 June 2016
- 3 Hanin M. Abdullah, Ahmed M. Zeki. Frontend and Backend Web technologies in Social Networking Sites: Facebook as an Example. Advanced Computer Science Applications and Technologies (ACSAT), IEEE 3<sup>rd</sup> International Conference on; 2014. p.85-89.
- 4 Web Applications: What are They? What of Them [online]. Acunetix official website  
URL: <http://www.acunetix.com/websitesecurity/web-applications/>  
Accessed 3 June 2016
- 5 Mimi Gentz. NoSQL vs SQL [online]. Microsoft Azure official website; 24 June 2016  
URL: <https://azure.microsoft.com/en-us/documentation/articles/documentdbnosql-vs-sql/>  
Accessed 3 June 2016
- 6 Rodriguez A. RESTful Web services: The basics [online]. IBM official website; 9 February 2015  
URL: <https://www.ibm.com/developerworks/library/ws-restful/>  
Accessed 5 June 2016
- 7 Nodejs.org official website [online].  
URL: <https://nodejs.org/en/>  
Accessed 10 June 2016
- 8 Erik Elof, Daniel Torstensson. An Investigation into the Applicability of Node.js as a Platform for Web Service. Linköping: Institutionen för datavetenskap; 2012.
- 9 Holmes S. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications; 2015
- 10 Benjamin Erb. Concurrent Programming for Scalable Web Architectures. Universität Ulm; April 2012.

- 11 MongoDB official website [online].  
URL: <https://docs.mongodb.com/manual/>  
Accessed 15 June 2016
- 12 Herron D. Node Web Development. 2<sup>nd</sup> ed. Packt Publishing Ltd; 2013
- 13 Kai Lei, Yining Ma, Zhi Tan. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js. Computational Science and Engineering (CSE), IEEE 17th International Conference on; 2014: p.661-668.
- 14 Express official website [online].  
URL: <https://expressjs.com>  
Accessed 17 June 2016
- 15 Ihrig CJ, Bretz A. Full Stack Development with MEAN. Cambridge, AUS: Site-Point Pty. Ltd; 2015
- 16 MongoDB documentation official website [online].  
URL: <https://docs.mongodb.com/getting-started/shell/introduction/>  
Accessed 17 June 2016
- 17 Györödi C, Györödi R, Pecherle G, Olah A. A comparative study:MongoDB vs. MySQL. Engineering of Modern Electric Systems (EMES), IEEE 13th International Conference on; 2015. p.1-6.
- 18 Chodorow K. MongoDB: The Definitive Guide. 2<sup>nd</sup> ed. Gravenstein Highway North, CA: O'Reilly Media Inc; 2013
- 19 AngularJS documentation official website [online].  
URL: <https://docs.angularjs.org/guide>  
Accessed 25 June 2016
- 20 Panda S. AngularJS Novice to Ninja. SitePoint Pty Ltd; 2014
- 21 AngularJS Tutorial [online]. W3schools official website  
URL: <http://www.w3schools.com/angular/>  
Accessed 27 June 2016
- 22 Single Page Application Tracking [online]. Google developers' official website  
URL: <https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications>  
Accessed 27 June 2016
- 23 Fernando Monteiro. Learning Single-Page Web Application Development. Packt Publishing; 2014

- 24 ASP.NET- Single-page Applications: Build Modern, responsive Web Apps with ASP:NET [online]. Microsoft developers' official website  
URL: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>  
Accessed 5 July 2016
- 25 Mozilla developers' official website [online].  
URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>  
Accessed 7 July 2016
- 26 jQuery API [online]. JQuery official website  
URL: <http://api.jquery.com>  
Accessed 10 July 2016
- 27 What's AJAX? [online]. Mozilla developers' official website.  
URL: [https://developer.mozilla.org/en-US/docs/AJAX/Getting\\_Started](https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started)  
Accessed 10 July 2016
- 28 Hernandez A. Init.js: A Guide to the Why and How of Full-Stack JavaScript [online]. Toptal LLC developers' official website.  
URL: <https://www.toptal.com/javascript/guide-to-full-stack-javascript-initjs>  
Accessed 10 July 2016

