View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Theseus

Phu Pham

Hybrid mobile application with Ionic and MEAN stack

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

April 25, 2016

Helsinki Metropolia University of Applied Sciences

Author	Phu Pham
Title	Hybrid mobile application with Ionic and MEAN stack
Number of Pages	41 pages
Date	25 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software engineering
Instructor	Kari Salo, Principal Lecturer

Due to the rapid emergence of web technologies, many different web application frameworks are built and developed every day. Choosing the right frameworks to work with is one of the most important decisions to make, especially for startup, small and medium-sized companies. This thesis was carried out to provide references about a web development stack called MEAN and a hybrid mobile application framework called Ionic.

The main goals of this thesis work were to build web and mobile applications with the Ionic and MEAN stack, examine the core features and usage of the technologies and propose the best practices and recommended architecture of a MEAN application.

The goals were achieved by observing and monitoring the development process of web and mobile applications. The project investigated and introduced the strengths and weaknesses of the technologies used and compared different alternative technologies in the market. The project also followed the principles and best practices during the development and decisions were made certain matters.

The project successfully built fully functional web and mobile versions of the application, covered the core features and their usages in real-world scenarios, introduced useful development tools and went through the setting and building process. Moreover, the project also suggested the best practices in organizing the application and designing the architecture.

Overall, the thesis work introduces and provides references about MEAN stack and Ionic. In future study, more real-world cases can be implemented to illustrate the performances and abilities of the technologies.

Keywords	Node.js, MongoDB, Express, AngularJS, Ionic, JavaScript, Hybrid application

Contents

Abbreviations and Terms

API	Application Programming Interface
CLI	Command-line interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Internet Information Services
JSON	JavaScript Object Notation
LTS	Long Term Support
MEAN	MongoDB Express AngularJS Node.js
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
NoSQL	Not only SQL
RDBMS	Relational Database Management System
REST	Representational State Transfer
SEO	Search engine optimization
SQL	Structured Query Language
URI	Uniform Resource Identifier

1 Introduction

Today, along with the development of technologies, web and mobile applications have gained incredible success since they allow us to share information, communicate with each other and connect people all over the world. Moreover, by providing highly functional and interactive features, more and more businesses are now operating on the internet. These applications are used for shopping, social networking, banking and so on. Hence, there is a great demand for fast, responsive and remarkably interactive web and mobile applications.

According to research about mobile marketing conducted by Dave Chaffey, the percentages of web views from mobile devices have increased dramatically in recent years [1]. This is also the reason why more and more mobile applications are built everyday. However, as there are multiple mobile platforms, one has to write one application for each platform. There are certain applications that do not need all the features of a native application. That is where the hybrid solution comes in. Hybrid applications are easier and cheaper to develop, especially for simple and lightweight applications. [2.]

The main objective of this thesis is to build a real-time web and hybrid mobile application with the MEAN stack and Ionic framework. The application was built to support early education and was used in this project as an example for Ionic and MEAN development. Hence, this paper does not focus on the features and functionalities of the application itself but on the technologies used and how they work together. This thesis also includes the basic steps to build the application from scratch, the factors that need to be taken into account during the development process, and good practices in web development with the Ionic and the MEAN stack.

2 Theoretical background

2.1 Client side and server side

Web development is about the communication between two parties, the server and the clients, over the HTTP (Hypertext Transfer Protocol). The server is responsible for serving pages or services requested by the clients, and displaying them to the users. In most cases, the client is a web browser that is used to surf the Internet, download media files or watch videos online. [3.]

2.2 Full-stack web development

Full-stack developers are the ones who are able to work with both back-end and front-end technologies. Specifically, it means that a full-stack developer is expected to be proficient in the server and hosting environment, different types of databases, user interface and user experience, quality assurance, understanding customer and business needs, etc. However, being a full-stack developer does not mean that he/she has to master every possible technology out there. It is more about having an understanding of each side of web development (client-side and server-side), communicating efficiently with team members, and being a good asset to the company when it comes to facing and dealing with new challenges. [4.]

2.3 Back-end technologies

In the client-server model, the front-end is considered a client, and the back-end is considered a server. By being closer and having the capability to communicate with the resource, which in this case is the database, a back-end application provides services and handles requests of the front-end. In order to ensure that the data or services requested by the front-end system are properly delivered by the back-end, there are a few essential and critical back-end technologies that are required from a full-stack developer. These are web frameworks, database applications and web server software. [5.]

A web application framework, or simply “web framework”, is a code library that is designed to help web developers to build reliable, scalable and maintainable dynamic web applications and web services. By providing the necessary functionalities and features, a web framework helps developers to stay focused on the logic of the application and to write clean, maintainable code without having to “reinvent the wheel”. Even though the majority of web frameworks are exclusively server-side technologies, some web frameworks have started focusing on client-side programming as well as providing full support to web developers (especially full-stack developers). According to Github stars, some of the most popular server-side web frameworks nowadays are Meteor, Ruby on Rails, Express, Laravel and Mean [22].

Database applications are software programs that are designed to collect, manage and retrieve information efficiently. In the world of web development, all data that is shared between components of a web application or needs to be persistent between requests are usually maintained in a database. There are two groups of databases that are commonly used by web developers, and they are relational databases and non-relational databases. Relational databases represent data in tables and rows, and the tables in a database are related to each other in some way. Relational databases are mostly used in large enterprise scenarios because they offer secure, robust and easy interactions with the data. The most popular are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2 [23]. Relational databases can sometimes also be called relational database management systems (RDBMS) or SQL databases. Non-relational databases, also known as NoSQL databases, are usually grouped into five categories: Key-value (Memcached, db4o, RIAK), Graph (Neo4j, HyperGraphDB, Jena), Column (Cassandra, Hypertable), Document (CouchDB, Solr, MongoDB) and Multi-model (ArangoDB, CortexDB, FoundationDB). NoSQL databases represent data in different ways from the tabular relations used in the relational databases. NoSQL databases have been used for many big data and real-time web applications because of their simplicity of design, flexibility and scalability, but considerations in choosing a NoSQL database for a web application really depends on the problem it has to solve. [24.]

In order to fully get a website up and running, all the computers that host it must install a web server software. The web server software uses HTTP to serve the files that form Web pages to users, in response to the requests that are forwarded by the clients. Leading web servers include Apache HTTP Server, Microsoft's Internet Information Server (IIS) and nginx from NGNIX [25]. Selecting which web server software to use

depends on different aspects such as its compatibility with the operating system and other servers, security characteristics and its performance capability.

2.4 Front-end technologies

In contrast to the back-end, a front-end application is the application that users interact with directly. To create the front-end for a website, a full-stack web developer must be at least adept at the three main languages: HTML, CSS and JavaScript. These are the three core components that control a web page's content, appearance and interactive functionality. HTML is a HyperText Markup Language file format that is used for structuring the content of a web page, and Cascading Style Sheets (CSS) is a style sheet language used for applying all the visual styles to the web page. Along with HTML and CSS, JavaScript is a high-level and interpreted programming language that defines almost all the interactive functionalities on the web page. In addition to being fluent in those languages, a full-stack web developer also needs to be familiar with other front-end frameworks that offer responsive web design, which is an optimal solution to displaying the web pages properly across multiple devices with different screen sizes and resolutions (desktops, laptops, tables, phones). Strong experience with the Ajax technique, a JavaScript code library that allows pages to be dynamically loaded in the background without having to refresh the browser, is also required for front-end development. [5.]

2.5 Hybrid application

The number of mobile users has surpassed desktop users since 2014 and continued growing even faster in 2015. For that reason, the time consumed on digital media on mobile devices is now greater than on desktop and other devices. According to KPCB research, in the US the digital media time spent on mobile accounts for 51% which is much higher than that of desktop (42%) and other connected devices (7%). [1.]

In order to adapt to this trend, building a mobile version of a web application is necessary. Hence, this project targeted both desktop and mobile users. There are three leading approaches to build a mobile application:

- Native: The approach uses specific development tools and programming language for each kind of device for development. For instance, Xcode and Objec-

tive-C / Swift are used to build applications for iOS devices, Android Studio and Java for Android devices, Visual Studio and C# for windows phone devices. Native applications allow full access to native platform features and give the best performance compared to the others. [6;7.]

- Web: The approach uses web technologies such as HTML5, CSS and JavaScript. A web application runs on a mobile browser like a normal website. A web application is designed to look and perform better for every platform but it will not feel like a native application. Developers can easily create a sophisticated web application that can run on multiple devices. However, a web application cannot access native device features and normally requires an internet connection to work. Moreover, web applications will not be distributed on application stores like Google Play for Android applications and App Store for iOS applications. [6;7.]
- Hybrid: This approach enables developers to use web technologies to build a native-like application. A hybrid application can be considered as a web application that is wrapped within a native container that grants access to some native features on the device. In other words, a hybrid application is a web application that is translated to a native application by a third party. The biggest advantage of hybrid applications is that they are faster to build and easier to develop and maintain than native applications. For example, to add a new functionality, developers only need to write it once but can distribute it on multiple platforms. In contrast, with native applications, developers have to reproduce the new functionality on every platform they want to support. Even though hybrid applications are generally considered to be the best of both native application and web application, it does have its own limitations. The performance of a hybrid application is not as fast as that of a native application since it still relies upon the browser speed. [6;7.]

Obviously, there is no one perfect solution for all situations. Each approach above has its own advantages and disadvantages. In most cases, people have to face the dilemma of native or hybrid application. With a native application, it requires more time and resources to build and maintain the application. As a result, the application will provide best performance and user experience. With a hybrid application, it saves a lot of development time and resources, but it means making sacrifices in terms of performance. For a lightweight application that has simple functionalities and features, the difference in performance between native and hybrid is negligible. However, if the application

grows big and more complicated features are added, users might have an unpleasant experience. [6;7.]

As stated in the introduction, the objective of this project is to build a mobile application powered by the MEAN technology in the backend. As the only developer in this project, I decided to choose a hybrid approach for the mobile application. This is a reasonable option with regard to the limited time and resources available.

In this project, a web application was built with the MEAN stack and a mobile application was built with Ionic – a hybrid mobile framework. These were feasible options because of the following reasons. First, the MEAN stack and Ionic use only one programming language in both the front-end and back-end side – JavaScript. Nowadays, JavaScript is a must on almost interactive web applications. Thus, almost all web developers are familiar with JavaScript. Working with MEAN stack and Ionic reduces the time of learning new programming languages and keeps the application persistent on both front-end and back-end side. Second, JavaScript is fast when it comes to event handling. Asynchronous feature of JavaScript prevents an operation being blocked by other operations. By passing a “callback” function that is invoked when the operation is done, JavaScript process allows parallel execution and hence, it performs better than blocking programming languages. Third, each framework in the project brings its own benefits. These frameworks will be discussed in detail in chapter 4 below.

3 Application

The product of this project is a communication channel between parents and teachers in a daycare center. This project was carried out by TinyApp, a start-up company focusing on early education. The implementation was done by the author of this paper. The idea was inspired by the CEO of the company and the design was made by a designer of the team. The following is the use case diagram for the application.

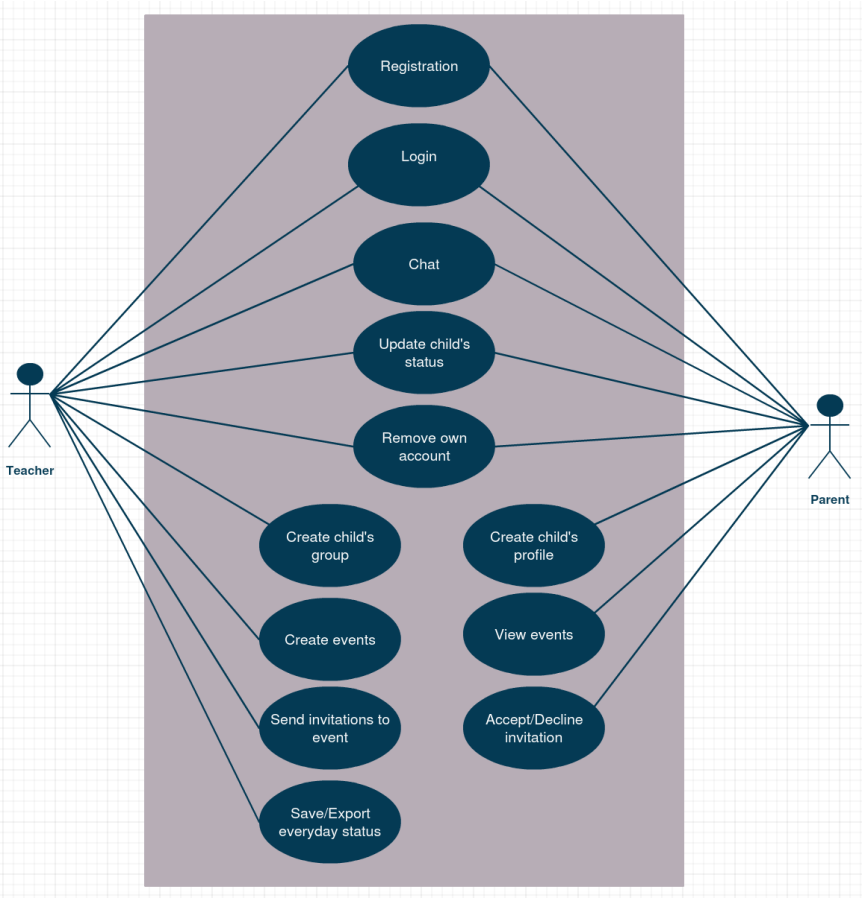


Figure 1. Use case diagram.

Figure 1 displays the main features of the application. Teachers and parents from a daycare center need to register an account to the system to be able to use it. When a user logs in, he/she should be able to chat, update a child's status or remove his/her own account. The teacher can add children to a group so that the parent can pick up their child and start adding the child's profile information. In addition, a parent can update his/her child's status of a day. The statuses are "incare", "outcare", "sick", "on holiday", etc. Similarly, a teacher can also view and update a child's status. Moreover, a teacher can create public or private events, and send event invitations to the parents within a group. From the parent's side, he/she can accept invitations and view public events. Only the teacher can create reports about the status of each child in the center and export the reports to a spreadsheet. Finally, teachers and parents can contact each other through an internal chat service. They can share media inside the application.

4 Technologies

4.1 MEAN stack

JavaScript was heavily criticized in its early days, but it has grown into one of the most popular and powerful languages in the world of web development. MEAN is one of the best examples of leveraging JavaScript to help web developers become full-stack developers by mastering only one programming language. MEAN is a full-stack web framework that consists of four very popular projects written in JavaScript: MongoDB, ExpressJS, AngularJS and NodeJS.

4.1.1 MongoDB

MongoDB is a cross-platform document-oriented database that allows users to utilize data and technology, reduce deployment risk and minimize time to value (TTV) with a significantly lower cost. MongoDB was developed by MongoDB Inc. in 2007 [8].

Flexible data model

MongoDB makes use of the key-value pair concept and stores data as documents in BSON (Binary JSON) format. A “collection” is a group of documents that share a similar data structure. “Similar” means that two documents in a collection do not necessarily have exactly the same data structure. One document can have some extra or different properties from the other. Each field in a document can contain other documents, arrays or arrays of documents. [9;10.]

Below are examples of documents in MongoDB:

```
User document:
{
  _id: <ObjectID1>,
  username: "johndoe123",
  courses: ["Math", "Data structures", "Nano phys-ics"],
  contact: {
    phone: "+358123456789",
    email: "john.doe@gmail.com",
  }
  group: <GroupID1>
}
Group document:
{
  _id: <GroupID1>,
  name: "Blue Sky",
  subject: "Bio physic"
}
```

Listing 1. MongoDB documents

In this example, the “courses” field is an array of strings, the “contact” field is an embedded document and the “group” field is a reference to “group” document that has the field “_id” with value of “GroupID1”.

Storing data as documents enables developers to localize the data. With the support of embedded documents, MongoDB reduces the work needed to join different documents. As a result, performance and scalability are substantially boosted by minimizing read requests to the server. Moreover, MongoDB document is similar to object, which is the native data type of many programming languages. This makes it easier to map the data model from front-end to back-end. [9;10.]

Dynamic schema

MongoDB documents have dynamic schemas. Unlike relational databases such as MySQL or MariaDB, MongoDB allows developers to create records without defining the data structure in advance. In other words, MongoDB documents are self-describing to the system. New fields can be added to a document whenever needed without any negative effects. There will be no data migrations, no system updates and no system downtime when changing the schema of a document. This feature, in fact, is really useful for expanding the application. When new features are implemented and data struc-

tures need to be changed, MongoDB dynamic schema saves a great amount of work from expensive database migration operations or even from re-designing the whole database system. [9.]

Auto-sharding

MongoDB provides a sharding method to address issues when the data grows. As the size of the database system increases, it requires a stronger server that has a larger storage capacity and can handle high read and write throughput. MongoDB sharding solves these hardware limitations by distributing data across multiple machines called shards. Each shard works as an independent database server. Hence, the shards balance the data and reduce the workload on a single server. Auto-sharding allows MongoDB to scale horizontally. [9;11.]

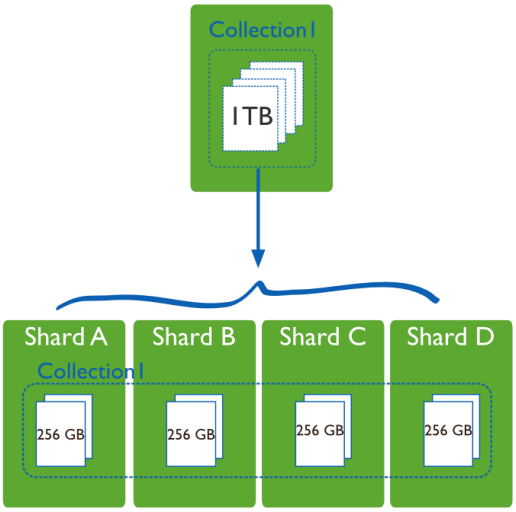


Figure 2. Sharded collection [11]

Figure 2 gives an example of the MongoDB sharding technique. For instance, we have a 1TB data set. Instead of having one expensive server to handle all the data and requests, we can have 4 different shards and distribute the data set for all of them. Each shard handles 256 GB of data and the throughput is also reduced on a single shard. The more shards we have, the less workload each shard has to handle. [11.]

Replication

Replication is the process of maintaining multiple copies of data on different shards or database servers. Replication offers redundancy and high data availability with the help of self-healing shards. [12.]

In MongoDB, replication is applied by using replica sets. A replica set is a group of MongoDB instances that manage a data set. In one replica set, there are several data bearing nodes and one optional arbiter node. Among the bearing nodes, there is one and only one node selected as a primary node while the remaining nodes are considered secondary nodes. The primary node receives all write operations from the client application. All the secondary nodes replicate and apply operations from the primary node to their data sets. [12.]

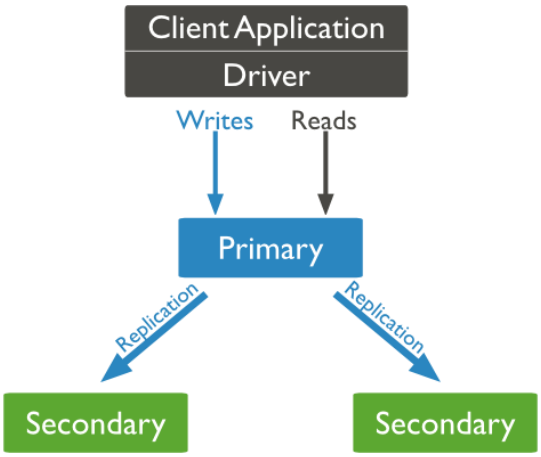


Figure 3. Replication process [12]

Figure 3 displays the replication in MongoDB. The client application communicates with the primary node for read and write operations. The primary node records the changes and then replicates the data to secondary ones.

MongoDB guarantees that the application will always be available even if there is a node failure. During a maintenance break or automatic failover, if a primary node fails to communicate with other members in the set for more than 10 seconds, an election will be established between the secondary members to choose new primary node. The previously failed node will join the set and work as a secondary node after its recovery. [12.]

Drawbacks of MongoDB

MongoDB does not support multi-document transactions. It means that MongoDB does not take different separated operations in a single transaction. Instead, MongoDB takes these operations individually. If one operation fails during the transaction, the other operations will continue normally. On the contrary, a transactional database can take multiple operations in one transaction. The transaction is successful if and only if all the operations are completed without errors.

This feature becomes crucial when we need to update multiple documents at once. Even though MongoDB supports atomic operations that allow us to modify multiple documents, MySQL seems to be a better option when it comes to complicated database transactions and performance. This is a common case for building banking software or web stores. [14.]

4.1.2 Node.js

Node.js (also called Node) is a server side software platform that allows one to create his own web server, build fast and scalable applications with JavaScript. Node.js runs on V8, an open-source JavaScript engine developed by Google. Node.js was created by Ryan Dahl in 2009 and has been continuously developed and improved until now. Despite the “Node.js” name, it is not solely written in JavaScript. Instead, Node’s source code includes 40% JavaScript and 60% C++. [13.]

Node.js uses event-driven architecture and non-blocking I/O model that boosts performance and optimizes throughput as well as scalability. In contrast to a thread-based concurrency model, Node.js can handle many network connections concurrently, making it an excellent option for data-intensive and real-time applications. [14.]

Single-threaded and asynchronous I/O

Node uses single threaded and asynchronous event-driven architecture to run the server. This is one of the reasons that makes Node very fast in performance and efficient in resource usage. Most of the popular web servers, including Apache and IIS, use blocking I/O to handle requests and use multi threads to avoid concurrency. This

means that when a new connection is established, it will be served with a separate thread associated with an amount of RAM. Blocking I/O increases complexity and overhead for the server as it causes threads to wait on I/O while the server processes and retrieves the data. [13.]

Leveraging the power of JavaScript, Node uses a single execution thread and asynchronous I/O to handle all the requests. Because of asynchronous or non-blocking I/O architecture, there is no delay on I/O or context switching. Instead of giving each connection its own thread, Node ties all the connections to the same thread and maintains an event loop. The loop looks for new events and then delegates them to handler functions. In the meantime, Node continues running other processes and invokes their handler functions when the operation of the events is completed. With this design, Node server serves the requests in parallel and hence, responds to the requests much quicker in comparison to blocking I/O servers. Parallel execution utilizes the resources on Node server. With the same performance requirements, servers running with Node need fewer resources than others. This is extremely beneficial for the server owner. [13;14.]

Performance and Utilization

Thanks to rapid technological development, people are able to access the internet almost anywhere and at any time. It means that there can be a huge number of requests sent to the server at the same time. For that reason, many online applications are facing problems of scalability, performance and utilization. Being able to handle concurrency and give the best user experience are the biggest concerns of application owners.

As explained earlier in the “Single-threaded and asynchronous I/O” feature, Node gives better performance with fewer resources due to its non-blocking I/O architecture. This is one of the key points that makes Node preferable over other frameworks. There is much research conducted to compare the performance between Node and other popular platforms. In 2014, Kai Lei, Yining Ma and Zhi Tan from Peking University published their paper comparing and evaluating performance of different web technologies including PHP, Python and Node.js. They conducted three fundamental test modules: printing out “Hello world”, computing the value of Fibonacci numbers and performing database operations. [15.]

The first module is to output the string “Hello World” on each request.

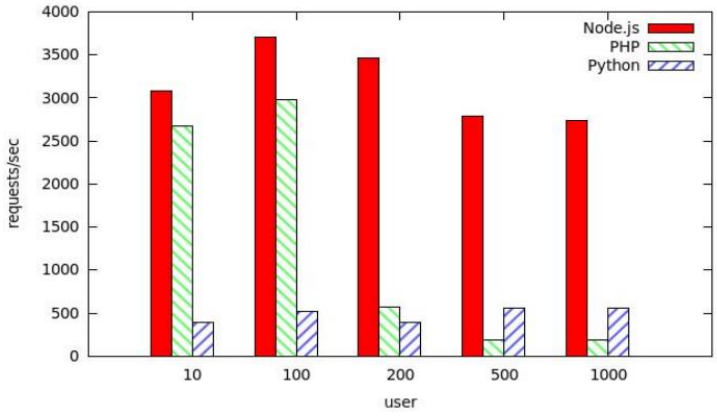


Figure 4. Results for “Hello world” mean requests per second. [15]

As displayed in figure 4, Node gives the best performance among the three candidates. For 10 users, Node can serve up to 3000 requests per second (req/s), followed by PHP with about 2700 req/s. The difference is even more obvious when the number of users increases up to 1000. Node still takes the lead with 2800 req/s, but PHP drops down to less than 250 req/s. Python keeps the performance stable with about 400-500 req/s in all test cases. [15.]

The next module is the Fibonacci number calculation. In this module, the web technologies are required to compute the 10th, 20th and 30th numbers of Fibonacci sequence.

Table 1. Results for “Calculate” Fibonacci numbers (10/20/30). [15]

Web development technology	Mean time per request [ms]
Node.js	0.401 / 0.654 / 16.993
Python-Web	1.578 / 4.764 / 345.307
PHP	0.488 / 5.942 / 560.553

According to table 1, Node performs best with average time to compute the 10th, 20th and 30th Fibonacci numbers is 0.4/0.7/17 milliseconds per request (ms/req) respectively. PHP’s equivalent results are 0.5, 5.9 and 561 ms/req. Python proves its efficiency for complex operation over PHP with the results are 1.6, 4.8 and 345 ms/req. However, there is still a big gap in performance between Node and Python. [15.]

The last test module is the database operations. This module aims to test the ability to tackle the IO-intensive challenge.

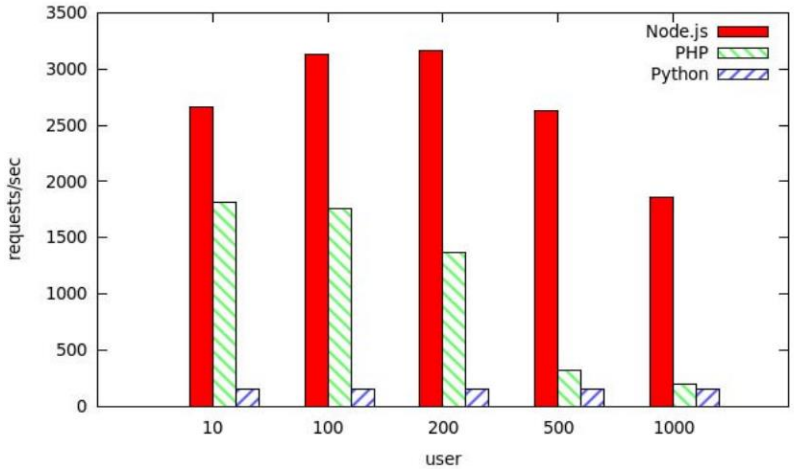


Figure 5. Results for “database operation” mean requests per second. [15]

Figure 5 once again shows that Node dominates the other two technologies in performing database operations. Node serves from 1900 (with 1000 users) to 3200 (with 200 users) req/s in all test cases. PHP takes second place but with a significantly lower rate. Python maintains 550 req/s for all the tests. Even though the results for this module are quite similar to the two previous test modules, Node reinforces its lead over PHP and Python at a large scale (1000 users). [15.]

Another example proving the power of Node is the benchmark comparison between Node and Java EE. Marc Fasel conducted a test by sending multiple requests to 2 separate servers written in Node and Java EE. The result is displayed in table 2 below.

Table 2: Node and Java EE benchmark results. [26]

Concurrent requests	**Average response time (ms)**		**Requests / second**	
	Node	Java	Node	Java
10	19	23	509	422
50	109	119	453	416
100	196	243	507	408
150	294	363	506	411

Java's server is multi-threaded while Node's server is single-threaded. Taking the benefits of non-blocking I/O, Node obviously performed better. The average response time of Node is about 17-20% faster in most cases while number of requests served each second is about 20-23% higher than that of Java. [26.]

The test results above indicate that Node is a good option for a scalable, fast and IO-intensive application.

NPM – Node Package Manager

Npm is a node package manager that gives developers the ability to fully manage third-party Node modules. We can browse, query, install and publish any Node modules from NPM's central repository. Each module has a unique name that is used to identify itself. One node module can have multiple versions and we can specify the exact version we want to install with npm. All the modules that are installed via NPM will be stored in a folder named "node_modules" and their names, associated with version, will be stored in a manifest file called "package.json". NPM makes Node applications easier to distribute, update and upgrade. [13.] For example, if we want to upgrade a module called "mongoose" from version 3.x to version 4.x, all we need to do is to change the version in "package.json" file and then run "npm install". The new version will be automatically updated. Another benefit is the distribution. Instead of uploading all the modules in "node_modules" folder to the server, we only need to upload the "package.json" file. From the server, we can run the command "npm install" to install all the modules. Thus, Node applications are normally lightweight and convenient to share among servers and local machines. [14.]

Node's bad use cases

Node.js is no doubt a great server-side platform to work with. However, it only shows its true advantages in appropriate applications. It means Node.js is not the solution of every application. First of all, Node does not work well with relational databases. Node is reasonably new in comparison with PHP or Java. Hence, the relational database tools for Node are not widely supported and lack consistency. Secondly, Node is not an ideal option for heavy server side processing. As Node is single-threaded, it uses only one CPU core to handle the process. If one request runs too slowly, the whole process

slows down as well. This situation occurs with compute-intensive applications. Thus, depending on the application, Node can be a good or a bad option. [27.]

4.1.3 Express

Express is the E in MEAN. Express is a minimal and flexible web framework for Node. It is built on top of Connect - a modular-oriented framework to build web applications in a well-tested and reusable manner. Express comes with a robust set of tools and structure that allows us to write applications faster and easier. [17.]

Application object, request object and response object

Express has three core components known as application object, request object and response object.

Application object is an instance of Express, created by calling top-level express() function and normally referenced by a variable named “app” [16]. The “app” object contains all the functionalities built on Express.

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

Listing 2. Application object. [16]

The simple snippet above shows us how to create an express application object. This “app” object allows us to route HTTP requests, configure middleware and application behaviors, render HTML views and set the template engine.

The request object is created when there is a client request to the server. It contains all the information about the HTTP request such as the query string, request body, request header, parameters, HTTP headers, request cookies and so on.

The response object is also created with the request object in the same client request. The response object contains properties and methods to send a response from the server. Express allows us to send response in different formats such as attachment, cookie, json, file or status code.

Below is an example of request and response usage.

```
app.get('/user/:id', function(request, response) {
      response.json({'userID': request.params.id});
    });
```

Figure 3. Request and response objects. [16]

In the example above, when the client sends a request to `/user/:id`, a handler function is passed with the request and response object. “id” passed in the request url can be found in request.params. “response.json” method is used to send a response in json format.

Middleware

A middleware is a function that handles HTTP requests. It has access to the request object, response object and the “next” middleware function in the request-response cycle. A middleware can modify request and response object, perform an action, end a request by sending out a response, or call the next middleware function in the stack.[16] The picture below gives an overview of a middleware.

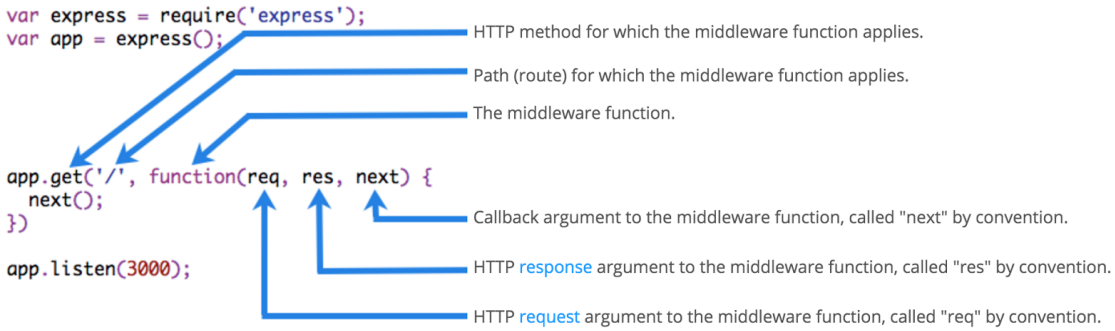


Figure 6. Elements of an Express middleware. [16]

Figure 6 indicates that if a middleware does not terminate a request by sending out a response, it must pass the control to the next middleware by calling “next()”.

Middlewares can be loaded in “use()” method of the application object, meaning “app.use()” method.

Routing

The route refers to the interface of request endpoints, also known as the URI or path of a HTTP request. Routing is the work of connecting route handler functions (or responses) to a request on the same route. A route can have multiple handler functions. [16.]

Express provides a very simple interface for the routing process. It follows the structure below:

app.METHOD(PATH, HANDLER)

Where:

- app is an application object
- METHOD is an HTTP verb or request method such as GET, POST, PUT, DELETE and so on.
- PATH is the request endpoint or URI
- HANDLER is the handler function to be called when route is matched. [16.]

Templating

Express makes our application more dynamic by offering template engine integration. The template engine allows us to separate program-logic and view into different parts. Instead of generating HTML pages using string concatenation, we can use predefined HTML templates and pass variables in them. The template engine compiles these templates to actual HTML pages at runtime and serves them to browsers. Express works well with Pug, Mustache, Ejs and some other template engines. Templating makes it easier and quicker to design a HTML page with Express. [16.]

4.1.4 AngularJS

AngularJS (Angular for short) is a structural JavaScript framework that enables us to build modern and dynamic client-side web applications. With Angular conventions, we can use and extend HTML as a template language. Angular in this paper refers to Angular version 1.x.

A popular way of building a web application is to get data, process data on the server, then generate HTML components together with that data and pass it to the client side. With Angular, the flow is done in a different way. The ideas of data binding and dependency injection separate the logic between the server side and client side; hence, it reduces the workload from the server side and makes Angular suitable for any server technology. [18.]

Two-way data binding

In Angular, data is synchronized between model and view components. It means that the changes in the view will make the changes in the model and vice versa.

To have a better understanding of two-way data binding, we should examine how one-way data binding works.

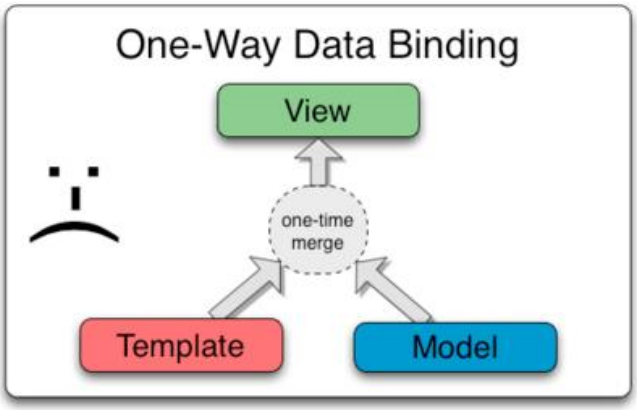


Figure 7: One-way data binding. [18]

Figure 7 shows how a classical template system works. The system performs a one-time merge between template and model to generate a view. After that, the changes that take place in the view or model will not affect the other. This means extra work is needed for the developer to control the changes. [18.]

Angular does not work like that.

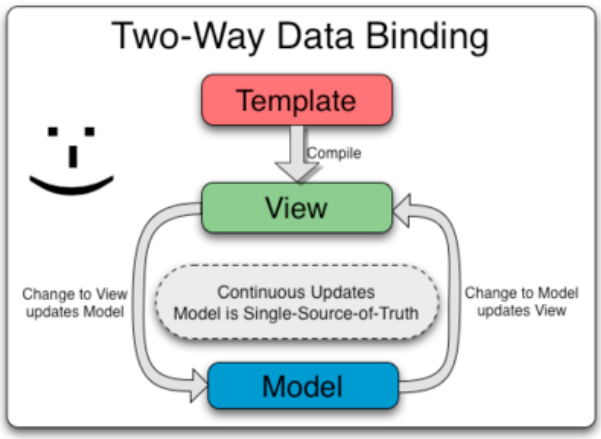


Figure 4. Two-way data binding. [18]

Figure 8 illustrates how Angular binds the model and view. The template and data are passed separately to the browser and then the browser compiles the template into the view while the data is assigned to the model. The difference here is that the view is bound to the model, meaning that the changes in the view will reflect the changes in the model and vice versa. This feature enables us to do the same amount of work with less time and effort. [18.]

MV* pattern

In the 1970s, the Model-View-Controller application structure was introduced in Small-Talk. Since then, MVC became popular and has been used in almost every development environment. Nowadays, MVC is considered a basic architecture pattern in most web frameworks such as Zend, Yii, Django, Ruby on Rails, etc. [19.]

The main idea of MVC pattern is that data management (Model), data presentation (View) and business logic (Controller) are separated at code level. The View displays the data from model to the user. The user interacts with the View and user input is handled by the Controller. The Controller modifies the Model based on user events. If the Model is changed, it notifies the changes to the View and the View updates its presentation.

Angular adopts this MVC architecture as one of its core concepts. Moreover, Angular also allows us to extend MVC pattern to MVVM pattern (that stands for Model-View-ViewModel). In MVVM pattern, the View handles user behavior and events. It is totally

separated with the Model. While the View is totally aware of the ViewModel, the View-Model is not aware of the View. However, the ViewModel can communicate with the view through data binding. All the business logic happens in the ViewModel. The view model invokes methods, commands and properties to interact with the model. [19.]

Angular is considered a MV* framework because of its flexibility in pattern design. MV* pattern gives us a clear overview of the application structure and makes it easier to organize our code. Moreover, it brings great benefits in extending, maintaining and testing the application.

Testability

Because Angular is a JavaScript framework, it does not receive any support from the compiler. For this reason, Angular was designed with a strong focus on testability. With built-in dependency injection, Angular makes testing much easier. We can test all application building blocks with unit testing or integration between components with end-to-end (e2e) testing. [18.]

To make testing easier, Angular provides dependency injection such as ngMock that allows us to produce a condition for a test. `$httpBackend` is a typical example. It mocks XHR requests and returns sample data. Hence, we can emulate backend behavior without sending real requests to the backend. Angular also abstracts the DOM. This is very convenient for test cases that require DOM manipulation. [18.]

Angular's common pitfalls

As with many other frameworks, Angular has its own issues. Angular renders its views from HTML templates and data. If the browser does not support JavaScript or the user disables it, the application will fail to display. Moreover, if there is a bug in the code, the whole application will crash. This is the problem of JavaScript itself. [14.]

Next, Angular does not work well with other JavaScript libraries such as jQuery. Most of the JavaScript libraries do not work directly with Angular. Instead, we have to use the Angular directive or rewrite the library with in the Angular way.

Another pitfall is search engine optimization (SEO). When a search engine crawls a site, it only crawls the HTML content of the page and does not execute any JavaScript files. It means that the view will not be rendered and the search engine only indexes the HTML template. Even though there are several ways to tackle this issue such as using Prerender, it is still annoying and requires extra work. [14.]

Apart from these issues, Angular is still an excellent front-end framework with all the benefits it provides.

4.1.5 MEAN architecture

There are several options to structure a mean application. However, most of them share a single feature: a RESTful API. The idea of RESTful API is clearly explained in figure 9 below.

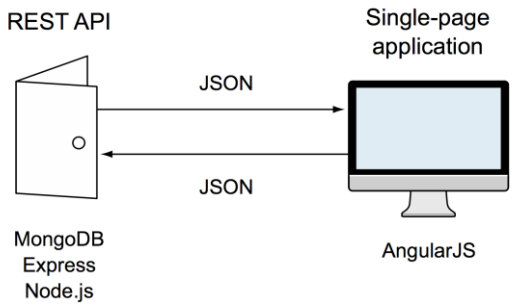


Figure 9. Common MEAN architecture. [14]

In figure 9, Angular is the front-end framework while MongoDB, Express and Node are running on the server side. Angular communicates with the server through an application program interface (or API) called representational state transfer (REST). RESTful systems normally use HTTP requests (GET, POST, PUT and DELETE) to manipulate data. This architecture allows us to build a fast, responsive and maintainable application. [14.]

This project was designed using the same architecture.

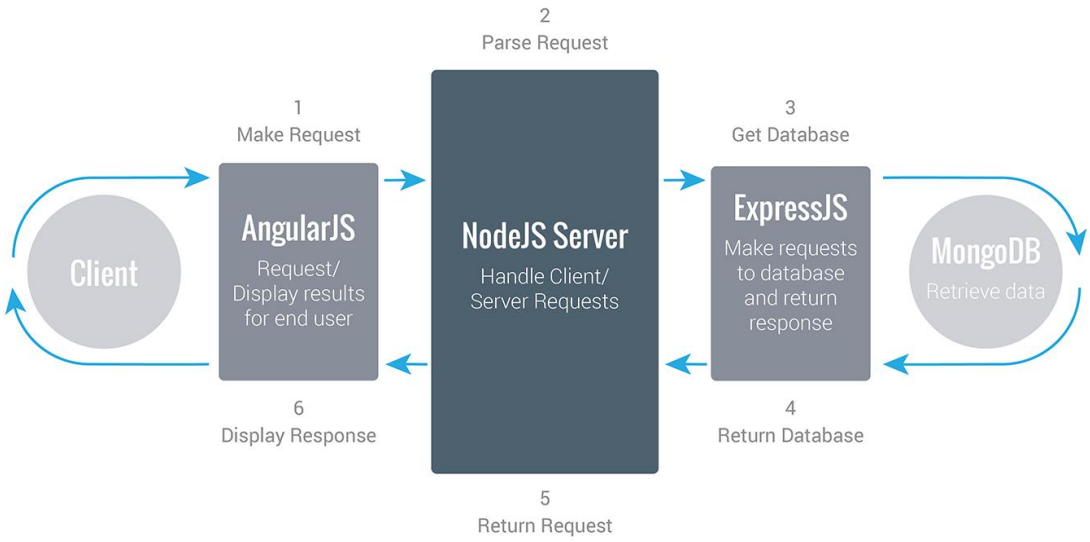


Figure 10. Architecture of the application in this project. [20]

Figure 10 describes the architecture of the application in this project as well as many other MEAN projects. Angular in the front-end handles the view and all the front-end logic. Angular's resource service is used to communicate with the Express server through RESTful API. Angular resource supports the basic actions associated with the HTTP methods such as get and query (GET method), save (POST method), remove and delete (DELETE method). We can also define our custom actions with appropriate HTTP method to make the communication easier with the server. When Angular makes a request to an API endpoint on Node server, Express links the request to the handler function. Then the handler function invokes method in the controller to retrieve the data from MongoDB. When the data arrives, Express will send the data, normally in json format, to the client. In the client side, Angular will receive the response, handle it accordingly and display it to the view. [14;20.]

4.2 Ionic

At an early stage, hybrid mobile applications were developed with Cordova. Cordova is a mobile application development framework that enables us to use HTML5, CSS3 and JavaScript to build a native-like mobile application. However, Cordova requires a lot of work to make an application look neat, user-friendly and native-like. In order to fill this gap, many other mobile frameworks were built. Some popular ones are Ionic, Onsen UI, Famo.us and AppGyver.

The ideal framework for this project was the one that could utilize the power of both Angular and Cordova so that front-end code could be reused in mobile development and adopt Cordova features. Among these frameworks, Ionic was the best fit for the web application for several reasons.

Firstly, none of these frameworks natively support both AngularJS and Cordova except Ionic. Onsen UI natively supports Cordova but needs more work to play with Angular. Meanwhile, AppGyver was built with Angular but does not work well with Cordova. Famo.us supports both but also requires extra work to setup and develop. [28.]

Secondly, Ionic offers great support with its own services and development tools. Ionic also has the biggest community in comparison to other frameworks [28]. Ionic benefits are discussed more in section 4.2.2.

4.2.1 Introduction

Ionic is a HTML5 hybrid mobile framework built on top of AngularJS and Cordova. Ionic makes building hybrid mobile applications easy, quick and elegant. Unlike a responsive mobile framework, Ionic comes with a set of native-style mobile components and popular mobile layouts. The combination of a powerful front-end framework AngularJS and modern native wrapper Cordova contributes greatly to building a powerful feature-rich mobile platform like Ionic. [21.]

4.2.2 Advantages of Ionic

Ionic provides the opportunity to build a hybrid mobile application that can replicate the features of a native application such as user experience, user interface and performance. There are several reasons showing why Ionic is a good option when it comes to developing mobile applications.

First of all, Ionic allows us to build mobile applications with the web platforms. One can build a mobile application with web technologies: HTML5, CSS3 and JavaScript. It is even better when Ionic is shipped with AngularJS and Cordova. Ionic adopts all the great features of these technologies and third-party modules. [21.]

Secondly, Ionic comes with powerful CLI tools. CLI tools make developing a hybrid application a lot easier and faster. We can quickly generate a seed project, preview the application in multiple browsers, emulate the application on iOS and Android emulator, or deploy the application on real devices with a few commands. [21.]

Thirdly, Ionic provides it own services that fully support our development process. Ionic Creator allows drag and drop components to design and export our application. Ionic Lab helps us to test iOS and Android version side-by-side immediately. Last but not least, Ionic offers native-like experience, good performance and beautiful predesigned components.

5 Implementation

5.1 Environment setups

Before developing a MEAN application, we need to set up a development environment. Node.js can be downloaded and installed from Node's official website (https://nodejs.org/en/download/). There are different install packages for all different operating systems. The current Long Term Support (LTS) version of Node is 4.4.3. Similarly, MongoDB can be installed from their official website (https://docs.mongodb.org/manual/installation/) with full support for Linux, OS X and Windows. Current version of MongoDB is 3.2.

Besides the development environment, the following tools were also used in this project:

- Hardware
  - Macbook Pro 2015
- Software
  - Operating system: OS X El Capitan
  - IDE: Webstorm bundled with Git version control, embedded command line tool, multiple language support.
  - Project hosting service: Bitbucket that supports code collaboration, flexible deployment models and so on.

- Server hosting service: Heroku that supports multiple programming languages, easy admin control as well as different buildpacks and add-ons.
- Database hosting service: mLab comes with cloud automation, real-time monitoring and alert, free backup and recovery.
- Database management tool: Robomongo which supports native MongoDB queries, nice graphic user interface and works well on different operating systems.

5.2 Implementation

5.2.1 Web application

Application flow

The application flow of this project is similar to that of many MVC applications: taking incoming requests from the client, handling the requests and sending a response.

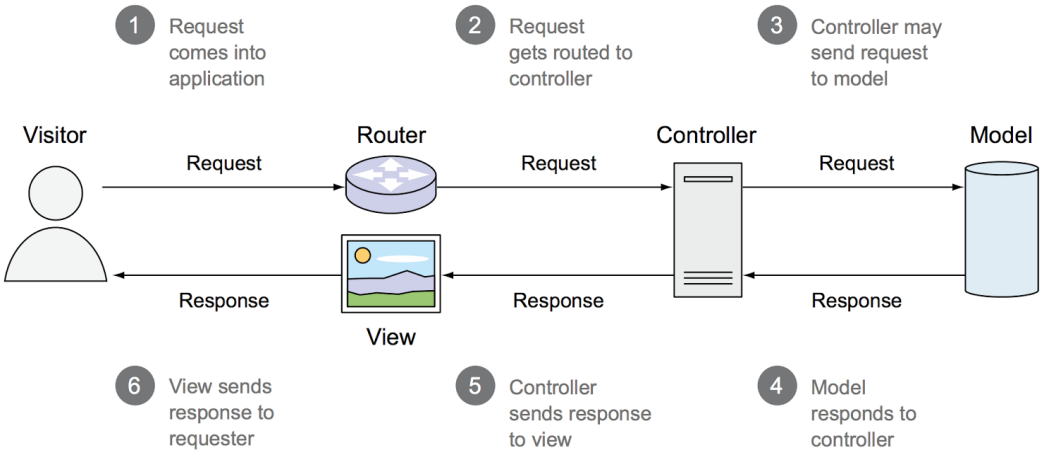


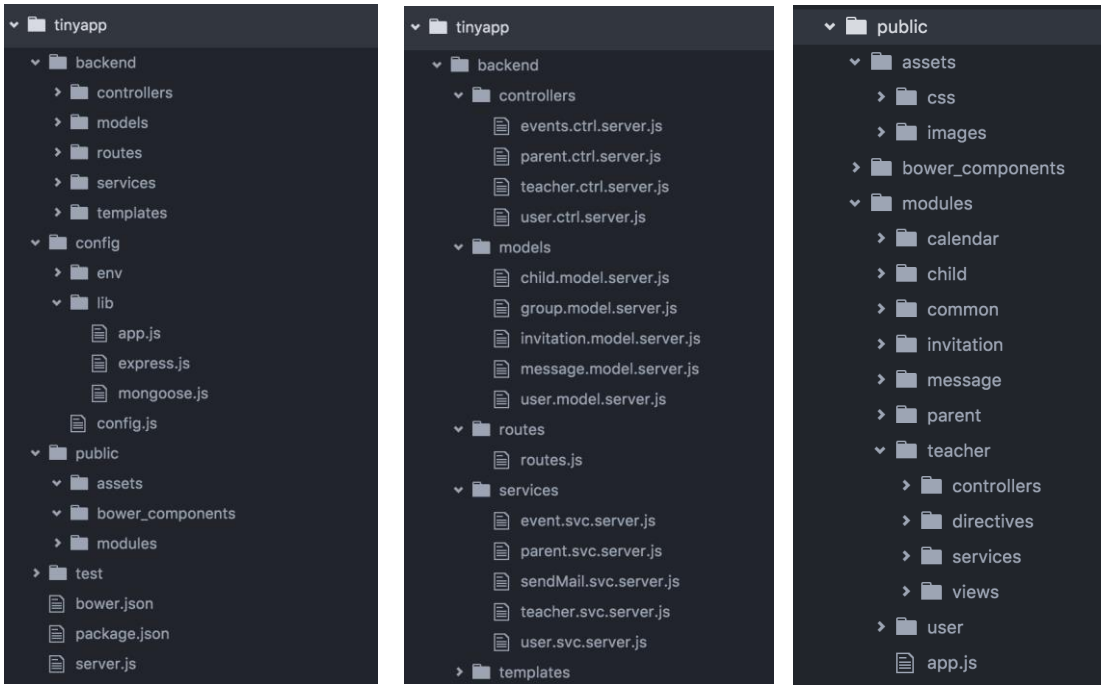
Figure 11. Application flow. [14]

Figure 11 features the basic flow of an MVC application and the importance of routing mechanism among model, view and controller. Depending on how the application is designed, the process of sending responses might be slightly different. The server can either compile the view before sending to the client or just send the raw JSON data for

the view to handle. In this project, raw data is sent to the client and Angular renders the views correspondingly.

Application structure

In this application, MVC architecture is applied in both front-end and back-end. Hence, it is a good practice to organize the application structure accordingly.



a. Application b. Backend c. Frontend

Figure 12. Folder structure.

Figure 12 above displays the folder structure of this application. To start the server, run “server.js” with Node from the root folder of the application. All the configurations for the application are inside the config folder including express, MongoDB and development settings. The logic is divided clearly between front-end and back-end. All the logic on the front-end side is inside the “public” folder while back-end logic is inside the “backend” folder. Both of them follow the MVC structure with model, views and controllers are separated completely from each other.

Server configuration

The configurations for express server are specified in “config/lib/express.js” file. Middlewares, view engine, and session are called within `app.use()` and `app.set()` methods of express.

```
var app = express();
app.use(bodyParser.json());
app.set('view engine', 'pug');
app.use(cookieParser());
app.use(flash());
app.use(session(sessionConfigObject));
```

Listing 4. Express configuration.

Express enables us to start a server within a single line of code.

```
var app = express();
    var port = config.port || 3000;
    app.listen(port, function () {
      console.log('Server is running on port', port);
    });
```

Listing 5. Start server with Express.

By using `app.listen()` method with given port, we have a server up and running.

Database setup

MongoDB itself is a schema-less NoSQL database. In MEAN stack development, MongoDB can be either used directly with Express or with an object modeling package called Mongoose. Mongoose allows us to access MongoDB database simply and easily. Mongoose can be installed via npm by running: `npm install mongoose`

We can connect to our MongoDB database by calling `connect()` method of mongoose:

```
var mongoose = require('mongoose');
var db = mongoose.connect(mongoUri, options, callback);
```

Listing 6. Connect to MongoDB with Mongoose.

The parameter *"mongoUri"* tells mongoose where to connect to the database. Depending on development environment, we can pass the mongoUri accordingly. If we are in "development" environment, mongoUri could be: "mongodb://localhost/tiny-app". However, if we are in "production" environment and using hosting service, for example "mLab", the uri is similar to this:

"`mongodb://<dbuser>:<dbpassword>@ds0543865.mlab.com:32987/tiny-app`"

In both cases, "*tiny-app*" is the name of the database to be used. The returned object from `mongoose.connect` is an instance of the database object.

Server routing

The Express router plays a key role in creating RESTful API in a MEAN application. When Express and MongoDB are successfully configured, the server can receive requests and send responses. Express makes the routing process easier with router supports. All the application endpoints are defined in "backend/routes/routes.js".

```
app.get('/api/users/:id', userCtrl.getUser);
app.post('/api/users/:id', userCtrl.updateUser);
app.delete('/api/users/:id', userCtrl.removeUser);
app.put('/api/users/registerDevice', userCtrl.registerDevice);
```

Listing 7. RESTful API example.

The example provided in listing 7 gives an overview of RESTful service. Different HTTP methods are defined to handle corresponding requests. When the routes are matched, the handling function is invoked. For example, if the client sends a GET request to "`/api/users`", the router will call the `getUser()` method of the "`userCtrl`" controller. Inside this method, the controller may interact with the database (or model) and send a request directly to the client.

Client side with AngularJS

Angular is responsible for everything that happens on the client side of a MEAN application. There are a few alternative methods to interact with the server. We can use $http or $resource service. Depending on the context of the usage, one would be more

efficient than the other. In this application, $resource service is used extensively to interact with RESTful server-side data sources.

```
angular.module('users').factory('User', function ($resource) {
  var userResource = $resource('/api/users/:id', {id:'@_id'}, {
        update: {method: 'PUT', isArray: false}
      });
  return userResource;
});
```

Listing 8. User service with Angular's resource.

The example in listing 8 indicates how to write a service with Angular $resource. Beside default actions of `$resource` such as get, save, query, remove and delete, we can define our own actions. In this example, "update" action is a custom action that has method "PUT" and does not expect an array as a response. This "User" service can be injected to a controller and invoke to interact with the server. List 9 below gives an example of 'User' usage.

```
angular.module('users').controller('UserController',   ['$scope',
'User', function ($scope, User) {
    var user = User.get({id: 1}, function () {
        user.name = 'John Doe';
        user.$save();
    });
 }]);
```

Listing 9. Usage of 'User' service in a controller.

In this example, 'User' service makes a GET request to the API endpoint "`/api/users/:id`" where "id" is a parameter. The server finds a match and calls the handler function which is "`userCtrl.getUser`" in List 9 above. When the server sends response, the result can be modified and updated via `user.$save()` action. This action triggers a POST request, the server routes the request to "`userCtrl.updateUser`" handler function. Then, Angular can update the model or view when the data comes from the server.

A special feature that makes Angular more efficient than other traditional front-end frameworks is that no page reload is needed when navigating between the routes. This reduces the excessive requests to the server, especially for serving static files. This application uses Angular UI-Router, a Single Page Application routing framework, to

handle client routing. UI-Router comes with a state machine that allows us to define the nested states and nested views.

Further development

When the main application flow is ready, by using the idea for modularity, it is easier to develop the application further. Other features of the application such as authentication, messaging and calendar event, follow the same flow.

5.2.2 Mobile application

Architecture

The architecture of the mobile application in this project is very similar to that of the web application. Because Ionic is built on top of AngularJS, it can substitute the role of Angular in the MEAN stack. As figure 10 shows Ionic is the client side which displays the views on a mobile application. Ionic communicates with the server side through RESTful API in the same manner as Angular. For that reason, it saves a good amount of development time by using Ionic because most of the client code can be reused in Ionic platform.

Application structure

Ionic's default structure for an application contains the folder named "`www`" under the root folder. "`www`" includes all the client code for the project. Because Ionic uses AngularJS as its core, the structure inside the "`www`" folder is very similar to that of the folder "`public`" in the MEAN web application. In this project, the folder structure of Angular and Ionic are the same.

Server authentication

When a seed application is created, the first step is to configure the server to authenticate requests from Ionic application. At this stage, mobile version and desktop version are on separated domains. For this reason, MEAN application will not accept the re-

quests coming from Ionic application. So the next step is to configure cross-domain requests. This can be done with express middleware.

```
var app = express();
var allowCrossDomain = function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'POST, GET, PUT,
DELETE, OPTIONS');
  res.header('Access-Control-Allow-Credentials', false);
  res.header('Access-Control-Allow-Headers', 'Content-Type, Au-
thorization, my-header');

  next();
};
app.use(allowCrossDomain);
```

Listing 10. Cross-domain request configuration.

As shown in listing 10, cross-domain configuration is very simple with express. First, create a middleware and then enable it with "`app.use()`" method.

Allowing a cross-domain request is very dangerous. It means that the server accepts all the requests outside the application itself. This leads to critical security issues and the server is easily attacked by hackers. This is where JSON web tokens come in. JSON web tokens (jwt) allow parties to securely communicate with each other. From the server side, whenever a user logs in from a cross-domain, that user will receive an access token. When a user makes a request, the token will be set in request headers. The server authenticates the user based on the token in the header and only accepts the request if the token passes the security check. The token will be removed when the user logs out or after a certain period of time.

Testing

Ionic offers a few ways to test the application. The most convenient way is to run: "`ionic serve`" from the terminal. This command will start the application on localhost and run the application on a browser. This method is very useful when we want to test simple layout changes such as color and text.

Another testing option is to use Ionic Lab. By running: "`ionic serve --lab`", we can test both iOS and Android version of the application simultaneously on a browser. Ionic

also offers GUI version of Ionic Lab, which allows us to serve, emulate, run or upload application in a single click.

The last option is to use iOS and Android emulator. Ionic gives us the ability to run the application on an emulator with "`ionic emulate ios`" or "`ionic emulate android`" command. We can also specify which devices we want to test. This option is commonly used for testing real behavior when the application is deployed on a similar device.

6 Results

The products of this thesis work were a web application and a hybrid mobile application. Both versions have fully functional features set from the beginning. Teachers and parents can communicate with each other without any difficulties. On the one hand, teachers can update the children's status, create reports, events and send event invitations. On the other hand, parents can create their child's profile, update the child's status, accept or join the events invited by teachers. The screenshots below illustrate the web application and the mobile application.

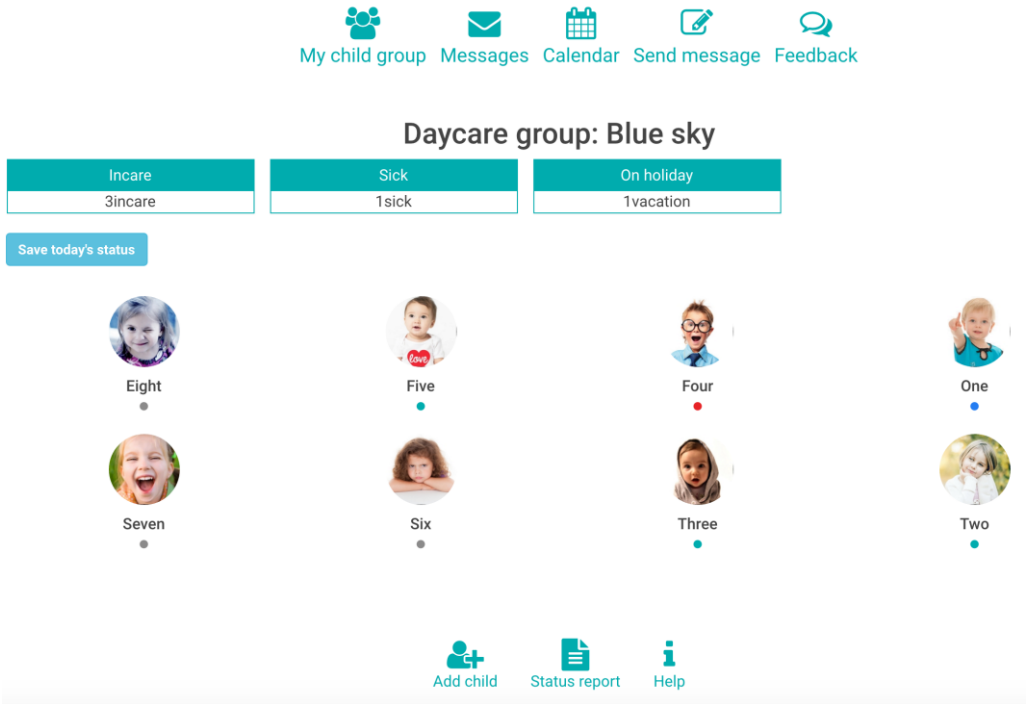


Figure 13: Web application landing page for teacher

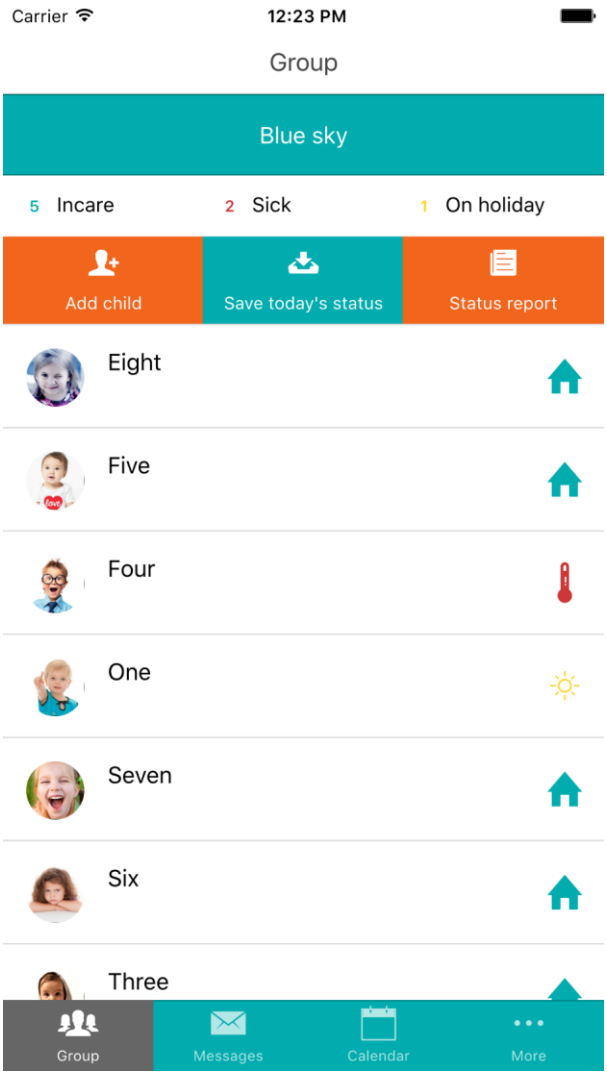


Figure 13. Mobile application landing page for teacher

Figure 12 and figure 13 indicate the landing page of the teacher in the web application and mobile application respectively. Children belonging to a group called “Blue sky” are shown here with their name, profile picture and status of the day.

From the learning perspective, the project reached its goal in building a functional web application and a cross-platform application with Ionic and the MEAN stack. It covered all the technologies needed and how to make them work together. It also proposed good practices in architecture design, application flow and organizing code for future maintenance and development.

However, from the real-world perspective, this project did not cover everything about the technologies. Because of the project’s scope, this paper only investigated the benefits gained from the technologies used but not their drawbacks or limitations. Moreover,

logic and integration testing was not implemented in the project due to limited time and resources. So even though the application was tested manually over and over again, it is possible that unexpected errors might occur when using the application or implementing new features. It is good practice to write tests (unit test, e2e test, API test) for each feature of the application. Testing can prevent many unintentional errors, server breakdowns and security issues. Hence, it saves time finding bugs and fixing errors.

7 Discussion

7.1 Web application

Besides the functionalities, the web application was built with MEAN architecture. The design of this application followed the MVC pattern on both server side and client side. MVC pattern allows us to separate code into modules hence it makes the code clean and clear as well as easy to maintain and extend in the future.

With only one programming language in the whole application, the client can communicate with the server seamlessly. The data is passed within the application in one native JavaScript format - JSON. This is a great advantage as we do not need to worry about data synchronization within the application.

Moreover, adopting the flow of RESTful API encourages us to have a clear division between client code and server code. Nowadays, many companies divide their development team into two smaller teams: front-end and back-end. In this situation, having separated front-end and back-end logic brings huge benefits to the development team. The backend and front-end teams can work smoothly together without a clear understanding of the other side. They only need to define the RESTful API and request, response data format. Thus, this design helps the team focus on their roles, their strengths and their core competencies.

7.2 Mobile application

Similar to the web application, the mobile application is also functioning with all features available. With great support from Ionic, the process of building a cross-platform mobile application was shortened by a large amount of time, especially when building it

with a MEAN application. Ionic inherits many of Angular's features and, hence, large amount of front-end code was reused in the mobile application. So if one wants to build a hybrid application from a web application built with MEAN, Ionic is a considerable option.

7.3 Seed projects and alternatives

There are several seed projects about MEAN development available in the field. Mean.js and Mean.io are the two main players as they provide a great starting point for MEAN developers. All the basic features such as authentication, server and database configuration, default packages, are ready to use. In general, Mean.js and Mean.io are very similar. This is because the creator of these two projects is the same person: Amos Haviv. Amos first started with Mean.io but then he left the company and created Mean.js.

Beside Ionic, there are also other similar frameworks for mobile development such as Onsen UI, Famo.us, React Native, etc. Among those, React Native emerges to be one of most popular. The selling point of React Native is that one can build native cross-platform applications with JavaScript. Unlike many other hybrid frameworks, React Native uses a "learn once, write anywhere" approach. However, in the end, Ionic is still one of the most powerful frameworks with its great performance, documentation and community support, especially when it comes to developing with the MEAN stack.

8 Conclusion

Following the rapid changes in web development, more and more new web frameworks are built and released every day. However, there is no one-size-fits-all solution. The selection of the frameworks or technologies to use in a specific project depends on its features and requirements. The project picked up two modern technologies for mobile (Ionic) and web (MEAN stack) in order to investigate and introduce these technologies. The main goals of this thesis work were to build web and mobile applications with Ionic and MEAN stack, examine the core features and usage of the technologies, propose best practices and recommended architecture of a MEAN application. The thesis achieved the following results.

First of all, functioning prototypes of the applications were built. Both web and mobile applications have all the requested functionalities and features with user-friendly interfaces. Ionic and MEAN stack are applied in the project.

Secondly, the project covered many core features and their applications in real-world scenarios. Each framework in the MEAN stack was introduced and illustrated with examples. The project also used and applied these core features in the application. The mobile development with Ionic also demonstrated how easy and fast it is to build a hybrid application with high performance and native-like experience.

Finally, the project introduced useful tools for development, and went through the setting and building process. Moreover, the project also suggested best practices in organizing the application and designing the architecture.

This thesis work proposed a new approach in web and mobile development with modern technologies. Moving from MySQL to NoSQL represents a fundamental change in persistence strategy. AngularJS makes a major shift by moving the logic from server side to client side. This change also moves the MVC pattern from server to client. In addition, this thesis proved that building an Ionic application with MEAN is a good option. All the application designs at coding level can be transferred from MEAN to Ionic without any obstacles.

In conclusion, this thesis work successfully achieved its original goals. However, there are still some limitations that could be overcome in the future. First, the application was not tested thoroughly. The testing was done manually; hence it did not cover any cases, which can be done at a coding level. In the future, all the features should be implemented with unit, integration and API tests to minimize the chances of unexpected errors. Secondly, because of limited time and resources, this thesis did not apply the core features of the technologies. Some features require bigger applications to test, for instance, the auto sharding feature of MongoDB. In this case, setting up a large database server and heavy queries are needed.

References

1 Dave Chaffey. Mobile marketing statistics compilation [online]. Smart insights official website; 21 January 2016
URL: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics.
Accessed: 20 February 2015

2 Greg Crowe. Hybrid apps: The future of mobile development? [online]. GCN official website; 12 February 2013
URL: https://gcn.com/Articles/2013/02/12/Hybrid-apps-future-mobile-development.aspx
Accessed: 21 February 2015

3 Carey Wodehouse. Client-Side Web Development: How Scripting Languages Work [online]. Upwork official website; 5 May 2015
URL: https://www.upwork.com/hiring/development/how-scripting-languages-work.
Accessed: 25 February 2016

4 George Fekete. Being a full stack developer [online]. Sitepoint official website; 22 September, 2014
URL: http://www.sitepoint.com/full-stack-developer.
Accessed: 26 February 2016

5 Michael Wales. 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack [online]. Udacity official website; 08 December 2014.
URL: http://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html
Accessed: 5 March 2016

6 Mario Korf and Eugene Oksman. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options [online]. Salesforce developer website; April 2015.
URL:
https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options
Accessed: 7 March 2016

7 Pietro Saccomani. Native, Web or Hybrid Apps? What's The Difference? [online]. Mobiloud official website; June 2012.
URL: http://www.mobiloud.com/blog/2012/06/native-web-or-hybrid-apps.
Accessed: 8 March 2016

8 MongoDB 3.2 manual [online]. MongoDB official website.
URL: https://docs.mongodb.org/manual.
Accessed: 10 March 2016

9 MongoDB architecture [online]. Mongodb official website.
URL: https://www.mongodb.com/mongodb-architecture.
Accessed: 12 March 2016

10 Data models [online]. MongoDB official website.
URL: https://docs.mongodb.org/manual/data-modeling.
Accessed: 14 March 2016

11 Sharding [online]. MongoDB official website.
URL: https://docs.mongodb.org/manual/sharding.
Accessed: 14 March 2016

12 Replication [online]. MongoDB official
URL: https://docs.mongodb.org/manual/replication
Accessed: 15 March 2016

13 David Herron. Node Web Development (2nd Edition). Packt Publishing Ltd; 2013.

14 Simon Holmes. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications Co; 2015

15 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), 2014 IEEE 17th International Conference on; 2014: pp. 661 – 668

16 Express documentation [online]. Express official website.
URL: http://expressjs.com.
Accessed: 20 March 2016

17 Hage Yaapa. Express Web Application Development. Packt Publishing Ltd; 2013.

18 Developer guide [online]. AngularJS official website.
URL: https://docs.angularjs.org/guide.
Accessed: 20 March 2016

19 Philipp Tarasiewicz and Robin Böhm. AngularJS. Brainy Software; 2014.

20 MEAN.JS full stack development solution [online]. Bacancy technology official website.
URL: http://www.bacancytechnology.com/mean-js-full-stack-development-solution.
Accessed: 22 March 2016

21 Jeremy Wilken. Ionic in action. Manning Publications Co; 2015.

22 Web application frameworks [online]. Github official website.
URL: https://github.com/showcases/web-application-frameworks.
Accessed: 25 March 2016

23 DB-Engines Ranking [online]. DB-engines website.
URL: http://db-engines.com/en/ranking.
Accessed: 25 March 2016

24 V Murali Krishna Raju. Relational databases vs Non-relational databases [online]. Linkedin official website; 12 November 2015.
URL: https://www.linkedin.com/pulse/relational-databases-vs-non-relational-raju-togaf-9-certified-.
Accessed: 25 March 2016

25 Web server usage statistics [online]. Builtwith website; 25 April 2016.
URL: http://trends.builtwith.com/web-server.
Accessed: 26 April 2016

26 Marc Fasel. Performance Comparison Between Node.js and Java EE [online]. Dzone website; 23 October 2013.
URL: https://dzone.com/articles/performance-comparison-between.
Accessed: 27 April 2016

27 Paul Shan. Node.js – reasons to use, pros and cons, best practices! [online]. Voidcanvas website; 11 October 2014.
URL: http://voidcanvas.com/describing-node-js.
Accessed: 27 April 2016

28 Danny Markov. Comparing the top frameworks for building hybrid mobile apps [online]. Tutorialzine website; 13 October 2015.
URL: http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps.
Accessed: 27 April 2016