Aihaiti Abudureheman

CREATING A COST SAVING WEB APPLICATION

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
May 2017

ABSTRACT

Centria University of Applied Sciences	**Date** May 2017	**Author** Aihaiti Abudureheman
Degree programme Information Technology		
Name of thesis CREATING A COST SAVING WEB APPLICATION		
Instructor Kauko Kolehmainen		**Pages** 45+0
Supervisor Kauko Kolehmainen		

The purpose of this thesis was to provide some theoretical and practical background information for the fact that single-page application is cost effective.

This thesis mainly focused on creating a single-page web application. In this thesis, one of the most popular JavaScript frameworks called Angular was used to create a single-page application.

There were two case studies. The first case study was about creating a web application with the single-page-web application frameworks, which was a cost saving web application. The second case study was about creating a web application without single-page web application framework, which was not cost saving web application.

This thesis used JavaScript and Typescript as the main programming language and AngularJS as the main single-page application framework.

This thesis also covered some fields related to networking. The main purpose was to prove the fact that the less page reloads for a web page means the less request needs to be sent to the server. The less request to the server means the less number of bandwidth is needed for a website. The less bandwidth means less hosting cost.

Keywords
Web development, web hosting, HTML, CSS, JavaScript, AngularJS, Typescript, single page application

CONCEPT DEFINITIONS

HTML stands for Hyper Text Markup Languages. So HTML provides a way to markup text with tags and that tags tell the browser how to display text on the browser. So it is the language of web browser that can be used to communicate with a web browser. It is the main building blocks of any website. It is the skeleton of the website. (Freeman 2012, 6).

CSS is an abbreviation for cascading style sheet. It is used to add style to HTML element. It can be added to the HTML element by using CSS rules inside style element. It uses HTML element's id and class selectors to find the specific element and apply a style to it. It is also used to add animations. It is also of the major elements to create a website. (Freeman 2012, 30).

JavaScript is useful technology web developers. It is the real programming language of the web. With the help of JavaScript, more behaviors can be added web pages. It transforms the static, boring web pages to a dynamic website. The most important features of JavaScript are that it help with reaching and interacting with users, grab some events and data, and draw graphics. In short, it gives live to the static website ((Freeman 2014, 1).

Typescript is a language for application-scale JavaScript. Typescript adds optional types, classes, and modules to JavaScript. Typescript supports tools for large-scale JavaScript applications for any browser, for any host, on any host. Typescript compiles to readable, standards-based JavaScript. The AngularJS itself started to use Typescript offers optional static typing (Microsoft, 2017).

Web hosting is one of the networking terminology. It is the space on the internet where all the website's files are placed. Typically, web hosting runs on a server owned by a web hosting company. The web hosting is just a small section of that server. Just similar to rent a place and build a house. It is the tool to connect the website to the outside world (Loiselle 2013).

Bandwidth in computer networking, the term 'bandwidth' refers to the data that passes between a web-site and the rest of the internet. For example, if a website contains a 5MB picture, and 10,000 visitors view this picture, this website uses 50,000MB(50GB) of bandwidth only for enabling the pictures to be seen to 10,000 times once per user (Mitchell 2016).

Http is also known as the HyperText Transfer Protocol. In other words, it is an agreed-upon method for transferring hypertext documents around the web. Hypertext documents are usually referring to the HTML documents which are created for web pages. HTTP is a simple request and response protocol. It is one of the major concepts to understand in web development (Freeman 2012, 168).

The Server is a computer designed to process requests and deliver data to other computers over a local network or the internet. There are many types of servers, such as web servers, Email server, FTP server and more. In order to connect the website to the server, the website needs to have active web hosting service (Mitchell 2017).

DOM stands for Document Object Model. It is an application programming interface for valid HTML and well-formatted XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. Everything that visible on the web can be called DOM or document object model (Hégaret, Wood & Robie 2000).

Angular is a structural framework for dynamic web apps. It lets developers use HTML as a template language and let the developer extend HTML's syntax to express application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the extra code. It is the most popular framework for creating single page application (Wahlin 2013-2014, 16-20).

Single page application is the one in which has a shell page and it can load multiple views into that. In traditional web application every time some part of the page is reloaded, it is going to reload the whole page again. But in single page application, the contents are loaded on the initial load and then different views or the little kind of mini web pages can be loaded on the fly and embedded into the shell (Wahlin 2013-2014, 16).

Ajax stands for asynchronous JavaScript and XML. It is a set of web development techniques using many web development technologies on the client-side to create asynchronous web applications. It is not a programming language or a tool, but a concept. It is a client-side script that communicates to and from a server or database without the need for a reloading after the first initial loading. It is the method of exchanging data between the server and the client without reloading the whole page (Segue Technologies 2013).

Git is the most popular version control system. It helps store application files which have been handled by individual or team. And it also keeps every updates or every version of application files. Git mostly implemented using command prompt in Windows and terminal in Mac (Freeman 2014, 31-34).

The Firebase Real-time Database is a cloud-hosted database. Data is stored as a JSON and synchronized in real-time to every connected client. One good thing about this online cloud-hosted database is that all the data are available when the app goes offline. It also can be editable directly from the Firebase platform (Firebase 2017).

ABSTRACT

CONCEPT DEFINITIONS

TABLE OF CONTENTS

TABLES

1 INTRODUCTION

Creating a single page application normally requires the developer to understand the concept of Ajax Development. The basic idea behind the Ajax development is that it is the method of exchanging data with a server, and updating parts of the web page without reloading the entire content of the page as the traditional web application do. There are many JavaScript frameworks which are used for Ajax development in web development field and AngularJS is one of the most popular ones.

The goal is to prove the fact that AngularJS based web application can be cost-effective mainly because of the less page reloading. In order to achieve this goal, two fully working web applications are created, one uses AngularJS as the main development tool, and other one does not use any frameworks. Both applications run and get the data with the help of Node.js server on the backend. For the database, Firebase is used as a backend data storage. Firebase is an online database storage. Chrome DevTool is used to collected page loading event information. The main goal is to compare the page loading frequency and times when pages change. Both applications consist of a typical page layout with the main page containing a link to the different page. Mostly the header and footer do not change when the main content changes.

This thesis is written for people who has a basic programming background either in JavaScript or in other programming languages like Java, C++, Python and PHP. Since it is somehow related to architectural design and it is not the step-by-step guide to creating single page applications, so this thesis is not suitable for beginners. Both use cases in this thesis are the simple web applications with several subpages, the main purpose is to make clear and explain the concept of cost saving advantages of single page web applications. There is one thing need to be mentioned that some practices in this applications are not suitable for large-scale applications, such as the way of creating the project structure might be different when developing large-scale industry level applications.

Since the main point of this thesis is to highlight the cost saving advantages of angular based applications based on the page loading frequency and page loading time when page changes, so the non-angular based use case does not have detailed explanation, but both application’s source codes are uploaded to my public GitHub repository. The majority of the source material used in this thesis are online resources, such as articles and electronic books, for this reason, there are no page numbers listed when referring to references.

2 BACKGROUND

AngularJS application can be created either by JavaScript or Typescript. Angular Material is the main UI library for creating well tested and accessible UI component. So before jump into AngularJS web application, this section gives some general information about the goals, restrictions of this thesis and some basic knowledge about the technologies used as a case study in this thesis.

2.1 JavaScript overview

JavaScript is one of the most popular programming languages for creating web applications. It is used almost by any websites and supported by almost all the modern browsers. It does not only work on the browser, it also works well on a gaming console, mobile phones, and tablets and in a lot of environments outside browsers (Freeman 2014, 1).

JavaScript is more suited for functional programming and object-oriented programming language. It is really easy to learn and use compared to other object-oriented and functional programming languages, such as C++, Java, and Python. It is the most fluid and flexible programming languages. In JavaScript, from writing code to deploying it is just a simple step after it is written, is can be run in a browser. From that moment the browser will take care of the rest, but before that, it is important to make sure that JavaScript is enabled in the browser (Freeman 2014, 2-3).

The main purpose of improving this application with JavaScript is to add behavior to create better user experiences. With HTML it is only possible to create some static content on the page without any extra style. By CSS there are more possibilities to make the website looks more beautiful. The website’s capability is still somehow restricted with HTML and CSS but with JavaScript, the website becomes more dynamic and interactive because JavaScript can add behavior to static content on the page. For small web application with several pages may be it is not necessary to use JavaScript but JavaScript comes very handily when it is time to create a large scale of applications especially because of the reusability of JavaScript function and variables for some repetitive behavior on the web. For example, there is one ordered list element which contains a list of students, the number of students is huge and it changes over time, the best solution for this is just using JavaScript functions of course, with several lines of code the problem will be solved, in another case, this type of behavior has to be added manually by using CSS animation, of course, it is not the ideal solution but it is possible and so much work to do. It is the fact that the function which is used to add animation can be used in any other applications if the behavior is similar, this is where the code reusability comes in handy when working with JavaScript. In other words, it helps HTML and CSS to make their life easier, eventually, it will make life much easier by helping with doing some huge amount of repetitive works (Freeman 2014, 1-5).

JavaScript is continuously growing since 1995. Now industry already started to use the new standard version of JavaScript, it is called ES2015 or ES6. The fact is that currently web browsers or other devices not fully support the ES6 yet but it has to be used with BabelJS transpiler. It helps with transforming the ES6 code into normal JavaScript code which can be understood by the browser. Since 1995, ES2015 is the most significant improvement for JavaScript. It added many new features such as classes, inheritance, arrow functions and promises (Kappert 2015).

One of the most important bright side when working with JavaScript is that most of the time it is not only about using pure JavaScript. There are many JavaScript frameworks and JavaScript library which are used to help with abstracting the most complicated logic and also can help with achieving cross-browser compatibility issues. In other words, it would make the developer’s life much easier and increased the development speed much faster. It is the power of reusable code in JavaScript. In fact, it is called JavaScript modularity. That means it is possible to create a module and reuse it in any application as long as it is suitable for the application needs (Kasireddy 2016).

Working with JavaScript means working with JavaScript modules. Modules are the essential part of JavaScript programming. Using modules is similar to using the chapter to divide the book content for good reason. In JavaScript programming, modules are used to divide the application code into separate sections and it becomes very easy to read and manage. Except that there are some advantages of using modules (Kasireddy 2016).

JavaScript modules are independent. It can be used in many applications as soon as it matches the requirements of the target application. This, on the other hand, saves a lot of energy and time from a development perspective. It can be maintained in different application environment without the needs of any changes from other parts of the applications (Kasireddy 2016).

Without the use of JavaScript modules it is very hard to avoid having same variable names in one code base. But when using JavaScript modules there is no need to worry about the naming issues since every module has its own namespace. Save time, energy for developers and save cost for the company. The main powerful part of JavaScript is its module’s reusability. By this way, sharing the code with others becomes easier (Kasireddy 2016).

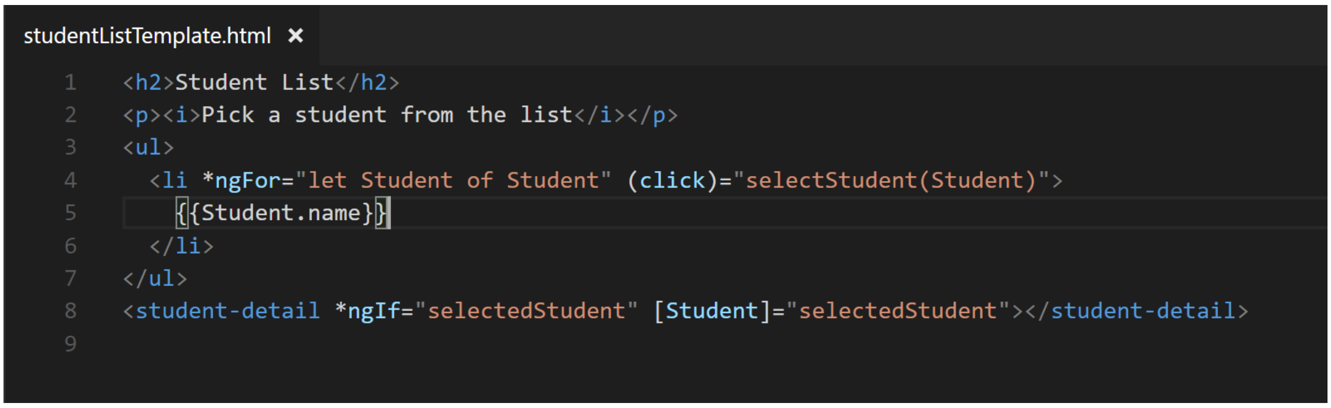
2.2 AngularJS overview

AngularJS is a JavaScript framework for building client-side web applications, actually, the main purpose is to build a single page web application. It has eight main building blocks, such as modules, components, templates, metadata, data binding, directives, services, dependency injection. Following are the more details about each of them before deep diving into the real case study (Shrivastava 2014).

Modules are the core of Angular applications. Angular also has its own module library actually that made the Angular more powerful. Every Angular application at least has one module called the root module which is the module that contains all the other modules. The angular application normally starts to run from root module (Shrivastava 2014).

The component is simple to understand that it is some part of the page or it is called a view. Sometimes in development, it is good to divide the whole page into separate small view and using component to control this specific view. It is very easy to use component. It can be inserted into some other component’s templates file as an HTML element (Shrivastava 2014).

The template is HTML file which has its own component or directive which it belongs to and it tells Angular to render this HTML file as the view of that specific component or directive. It is totally reusable and one template file can be used for many components as soon as it matches the component needs. Angular template is HTML file without any *<!doctype HTML>* declaration or starting and closing HTML tag. It is purely HTML code (Shrivastava 2014). GRAPH 1 is the basic example of the Angular template file, it is a pure HTML file.



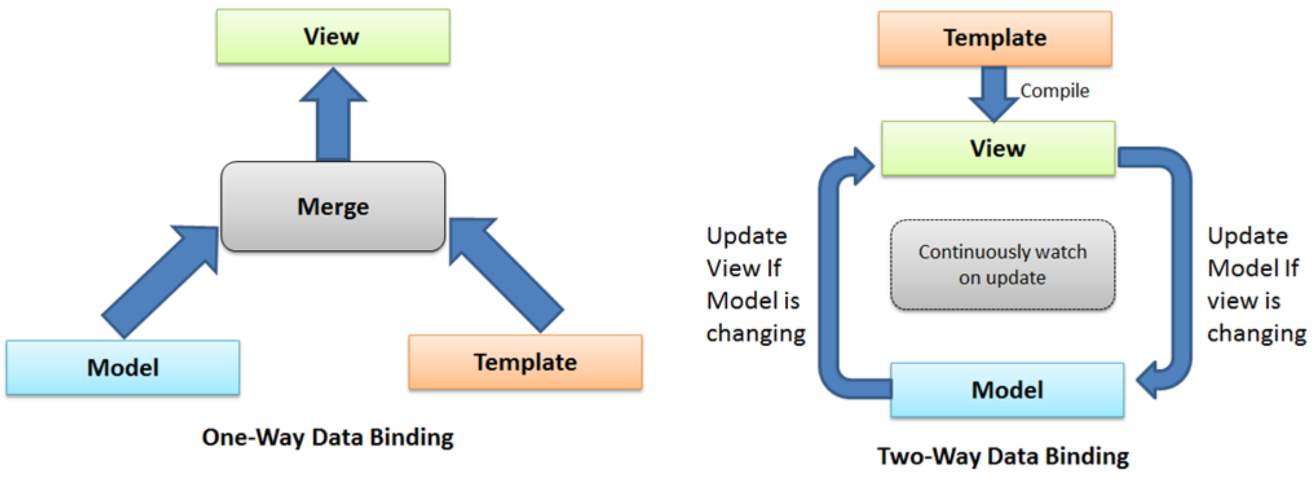
studentListTemplate.html ×

```
<h2>Student List</h2>
<p><i>Pick a student from the list</i></p>
<ul>
  <li *ngFor="let Student of Student" (click)="selectStudent(Student)">
    {{Student.name}}
  </li>
</ul>
<student-detail *ngIf="selectedStudent" [Student]="selectedStudent"></student-detail>
```

GRAPH 1. Student list template file example

Metadata is responsible for telling the Angular to how to process class. In the latest version of angular, a component can be seen as a class. The main thing is that Angular does not know it is a component if it is not mentioned as a component using component decoration with metadata. It is the communication tool between the current application and angular itself (Shrivastava 2014).

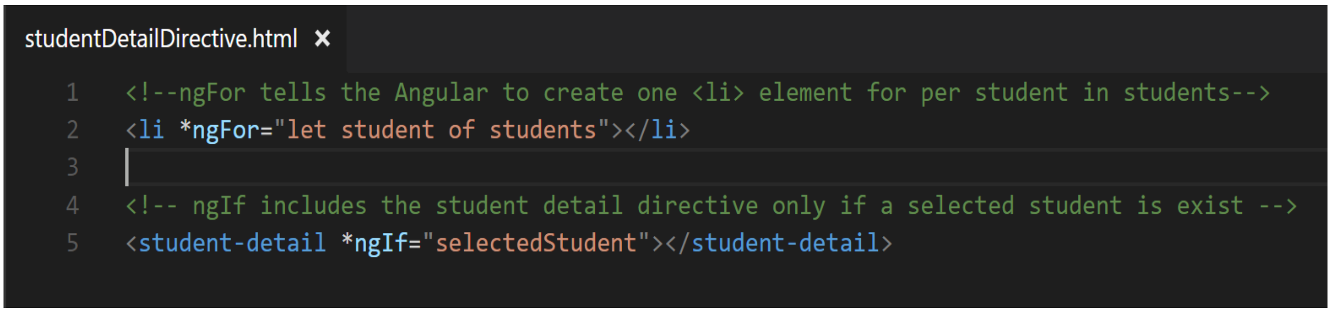
Data binding is one of the unique mechanism of Angular. It helps synchronize data between model and view. When the model changes, the view reflects the changes; when the view changes the model reflects changes. When the model change, the view is reflected also in two-way data binding. (Trivedi 2015). GRAPH 2 is the illustration of two main features of AngularJS framework, the one-way data-binding, and the two-way data-binding. Both are the core features of AngularJS.


View
Merge
Model
Template
One-Way Data Binding
Template
Compile
View
Update View If Model is changing
Continuously watch on update
Update Model If view is changing
Model
Two-Way Data Binding

GRAPH 2. One-way data binding and two-way data binding illustration

The directive is the instruction to tell Angular how to render the Angular templates. In simple words, a directive is the newly created HTML element by Angular. It works as a normal HTML element. It has properties and attributes like the normal HTML elements have. There are two types of directives: structural directives and attribute directives (Wahlin 2014).

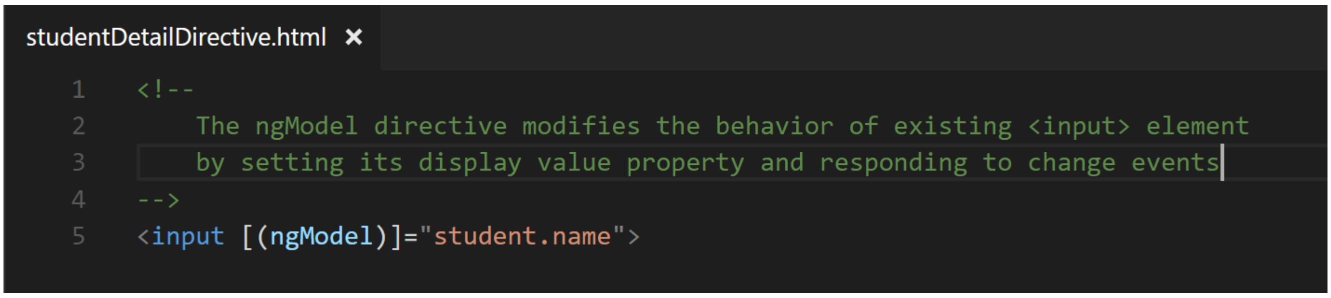
Structural directives alter layout by adding, removing, and replacing elements in DOM. GRAPH 3 is the simple example of built-in structural directives, *ngFor* tells the Angular to create as many <*li*> elements based on the number of students. *NgIf* includes student detail based on the conditional expression which assigned to it.


studentDetailDirective.html

```
<!--ngFor tells the Angular to create one <li> element for per student in students-->
<li *ngFor="let student of students"></li>

<!-- ngIf includes the student detail directive only if a selected student is exist -->
<student-detail *ngIf="selectedStudent"></student-detail>
```

GRAPH 3. Student detail directive file for ngFor directive

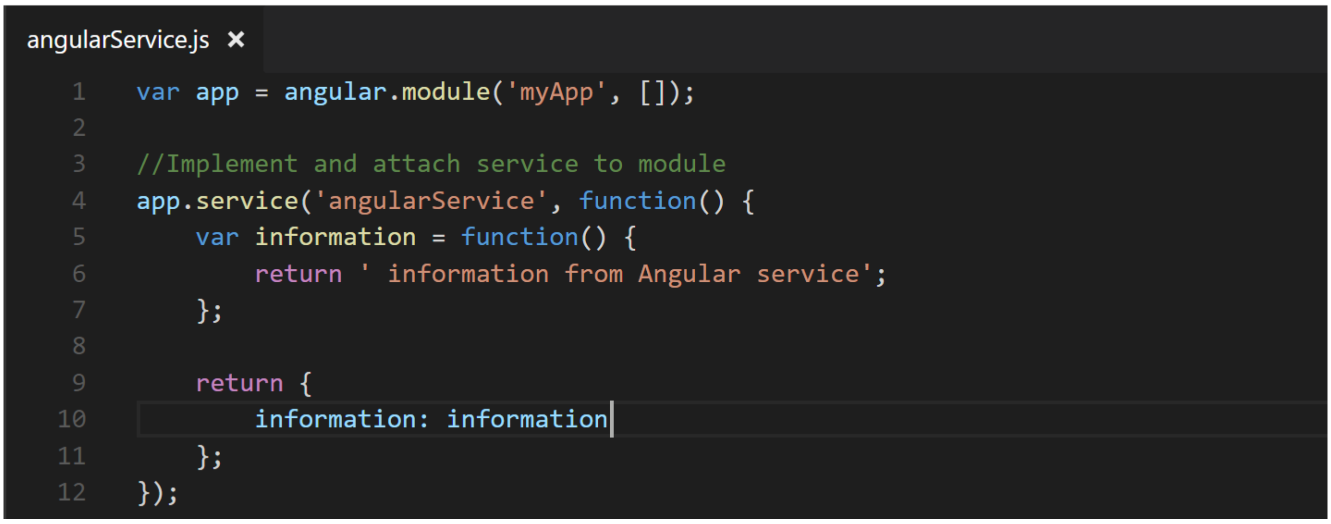
Attribute directives alter the appearance or behavior of an existing element. In other words, it extends the existing HTML features and adds more flexibility for the DOM element. Most of the directives are applied during the run-time. (Wahlin 2014). GRAPH 4 is the simple example of attribute directives. The *ngModel* directive modifies the behavior of existing <*input*> element by setting its display value property and responding to change events.


studentDetailDirective.html

```
<!--
    The ngModel directive modifies the behavior of existing <input> element
    by setting its display value property and responding to change events
-->
<input [(ngModel)]="student.name">
```

GRAPH 4. Student detail directive file for ngModel directive

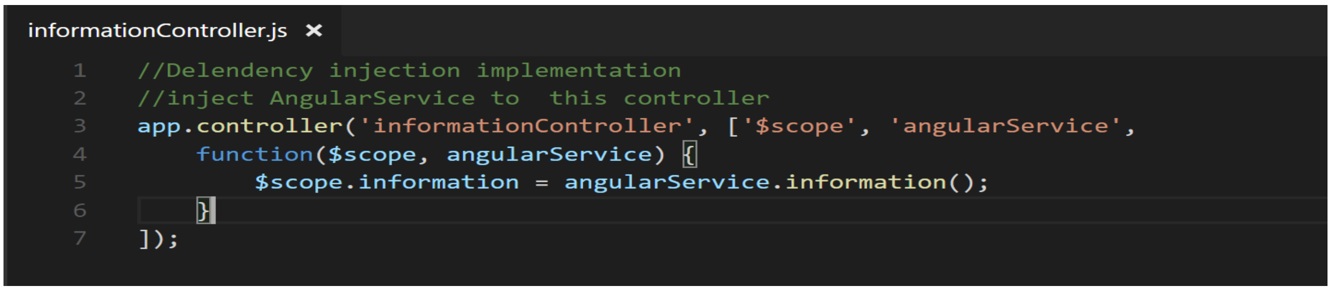
Service is a value, feature or function that can be available application wide in the current application. Service can be used by other directives, components and also other services, but only in the current application. There are many built-in services in Angular, and the good thing is that it is possible to create custom directive (Wahlin 2014). GRAPH 5 is the code snippet for newly created service called *angularService*, service is implemented by attaching it to its module (Kayal 2014). This service is attached to the *myApp* module. And it can be used by any other components by simply injecting it to the component's controller.


angularService.js

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

//Implement and attach service to module
app.service('angularService', function() {
    var information = function() {
        return ' information from Angular service';
    };

    return {
        information: information
    };
});
```

GRAPH 5. Angular service example

Dependency Injection is a software design pattern which indicates the dependency level between software modules. In Angular, it has a very common usage for utilizing the service in various components, directives or in other services in application wide. Normally dependency injection occurs when there is a need for services in the controller. GRAPH 6 is the example of the newly created controller and usage of dependency injection by injecting the service created above to this controller.


informationController.js

```
//Delendency injection implementation
//inject AngularService to  this controller
app.controller('informationController', ['$scope', 'angularService',
    function($scope, angularService) {
        $scope.information = angularService.information();
    }
]);
```

GRAPH 6. Information controller file for controller and dependency injection example

3 CASE STUDY: CREATING AN ANGULARJS BASED WEB APPLICATION

AngularJS is selected for this case study because it has been gaining very much popularity as a single page application framework and most importantly it is backed up with Google, one of the biggest player in the tech industry. The originally thought was using other JavaScript frameworks but the recent fast development and popularity of AngularJS were the main reason to recommend it to anyone who wants to create single page application for multiple reasons. First, AngularJS work properly and it will exist in the future a lot more than other frameworks. Second, it is the framework that used by many companies. Third, since Google is supporting this framework for growth, there is no doubt that it is going to grow more popularity with the powerful community (Krill 2013).

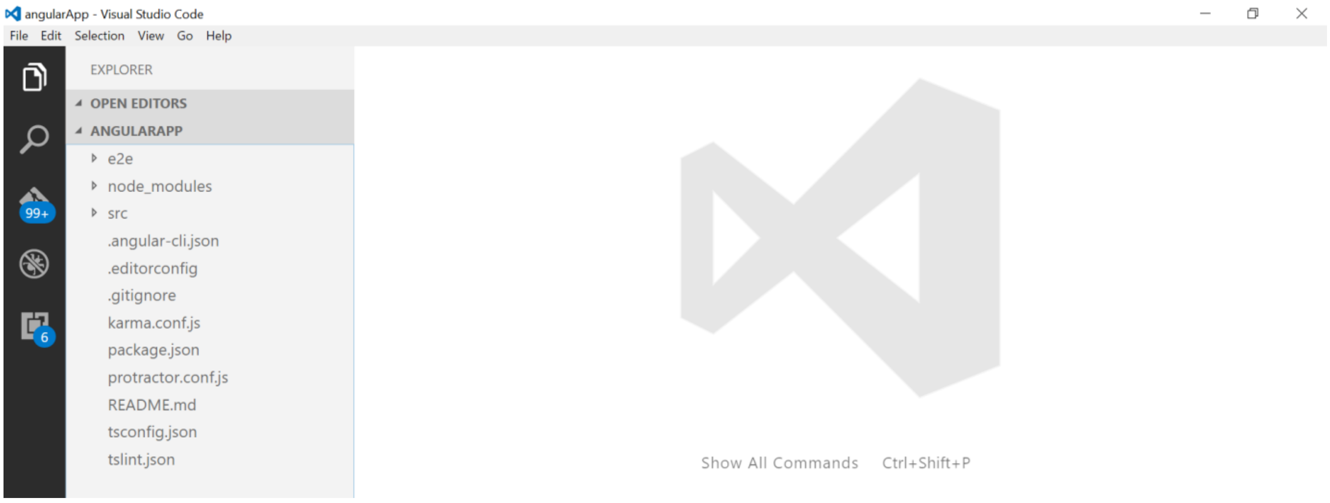
3.1 Installing Node.js and seting up the project folder

Before everything starts, it is necessary to download and install the latest version of Node.js on the local machine. The NPM which is node package manager comes with Node.js (Node.js Foundation. 2017). Node.js can be downloaded from its home page. After installation finished using command line tools to check if Node.js and NPM are installed successfully by simply checking their version installed using the command *node –version* and *npm –version.*

After the installation of Node.js was successful, it is time to install the Angular CLI, which stands for *Angular Command Line Interface*. It makes the application creation process much easier. Normally, NPM can be used to install Angular CLI by simply using the command *npm install –g @angular/cli.* The new angular app can be easily created using the command *ng new*. For example, *ng new angularApp* command creates an angular app called angularApp. Running the newly created angular app is really simple with the command ng serve. (Silva 2017).

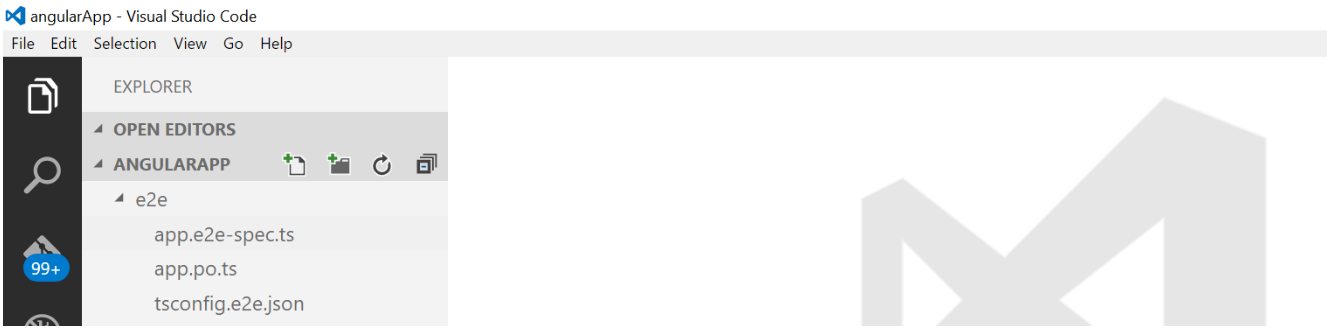
3.2 Project folder introduction

There are many ways of creating the project folder based on the application needs. In order to make the application easily maintainable and scalable in the future, it is really necessary to create a well-structured project folder in a standard manner. A well-structured project folder is also can be easily understandable and readable (Kukic 2014). The GRAPH 7 is the illustration of the file structure in the project folder. It includes not only the source file of this application but also some other configuration for testing, editor, git, code formatting and node modules managing.



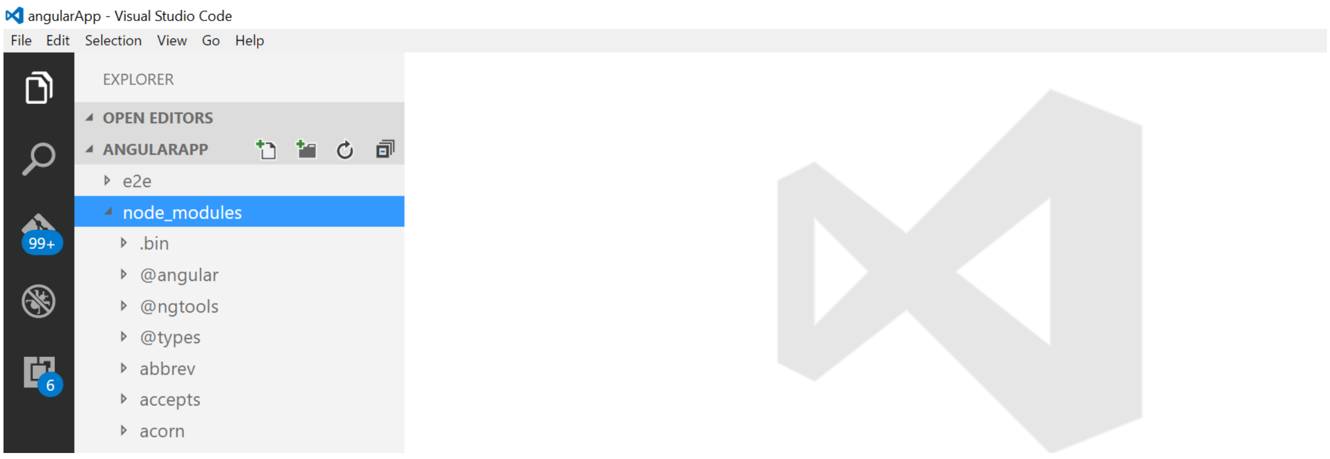
GRAPH 7. Project folder file structure

In AngularJS application, an automated end to end tests case is an important part of the continuous integration. Delivering a new release of an application becomes troublesome without a good end to end test cases. Normally, the manual regression testing can take weeks depending on the complexity level of the application. That implicates that code freeze has to happen a long time before the product is ready to be deployed or ready to deliver. Missing the automated end to end test cases eventually slow down the development or improvement of the application, no, it is really necessary to be taken care of. The GRAPH 8 shows the list of files in an *e2e* folder in which all the files related to *End-to-End* testing can be placed (Schenker 2014).



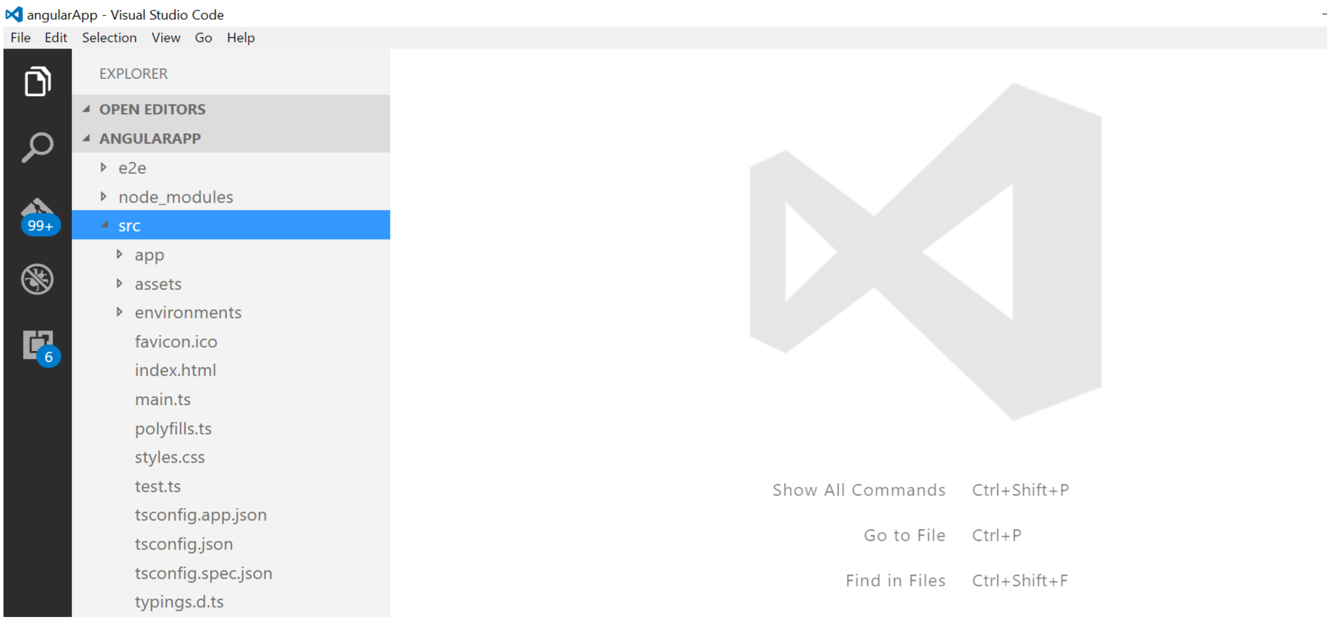
GRAPH 8. List of files in *e2e* folder

In node modules folder there is list of modules which are used by this project. By the way, the module is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout this application. Each module in this folder has its own development context, so it works fine without interfering with other modules or polluting global scope. And, all modules can be placed in node modules folder and can be used by listing it in the *package.json* file (Kim 2015). The GRAPH 9 is about the *node_modules* folder, which includes all the Node modules this Angular app needs during the development and production.



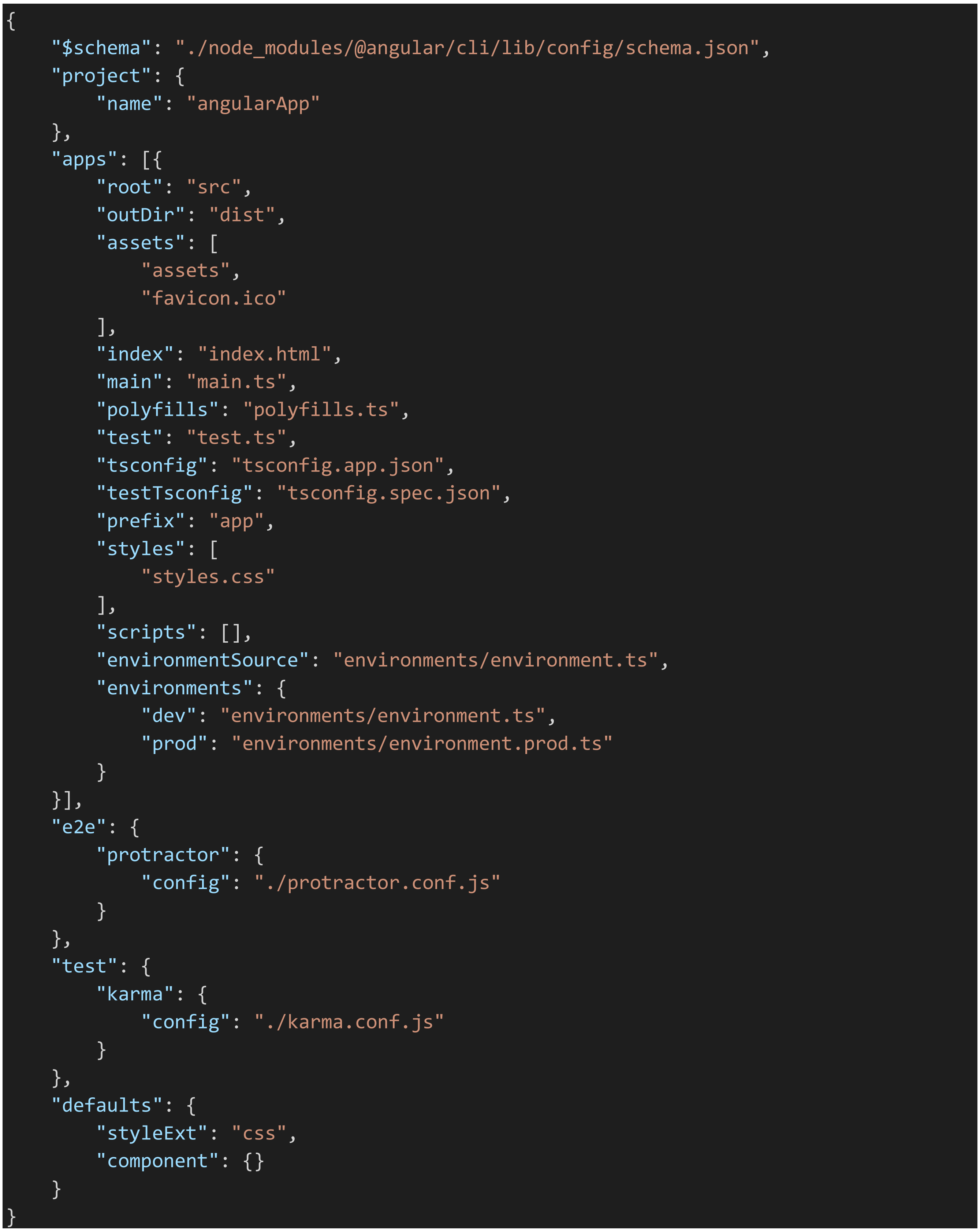
GRAPH 9. List of modules in *Node_modules* folder

In project folder, there is one folder called *src* folder. It is the place where all the application related files, directories are located. It puts all the application specific local files and directories in an *app* folder. It puts all the assets like images and media files in an *asset* folder. It puts some environment specific files in the *environments* folder. It puts all the other application-wide global files outside these three files. The GRAPH 10 illustrates the file structure of ***src*** folder which includes all the global source files and global configuration files of this Angular app.



GRAPH 10. List of files and folders in *src* folder

Angular cli is a command line interface. It is used to scaffold and build angular apps using node.js style called commonjs. Scalable project structure definitely needs it, and it handles all common tedious tasks for a developer out of the box. It is really easy to use and it has many features. When creating an application, it is necessary to create a configuration file about *angular-cli*. Based on the application need, it has a different configuration. In order to configure the angular-cli tool for this application, there is one angular-cli.json file created (Silva 2017). The GRAPH 11 is the editable code snippet for the *angular-cli.json* file, it includes more detailed configuration information about this Angular app.



```
{
    "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
    "project": {
        "name": "angularApp"
    },
    "apps": [{
        "root": "src",
        "outDir": "dist",
        "assets": [
            "assets",
            "favicon.ico"
        ],
        "index": "index.html",
        "main": "main.ts",
        "polyfills": "polyfills.ts",
        "test": "test.ts",
        "tsconfig": "tsconfig.app.json",
        "testTsconfig": "tsconfig.spec.json",
        "prefix": "app",
        "styles": [
            "styles.css"
        ],
        "scripts": [],
        "environmentSource": "environments/environment.ts",
        "environments": {
            "dev": "environments/environment.ts",
            "prod": "environments/environment.prod.ts"
        }
    }],
    "e2e": {
        "protractor": {
            "config": "./protractor.conf.js"
        }
    },
    "test": {
        "karma": {
            "config": "./karma.conf.js"
        }
    },
    "defaults": {
        "styleExt": "css",
        "component": {}
    }
}
```

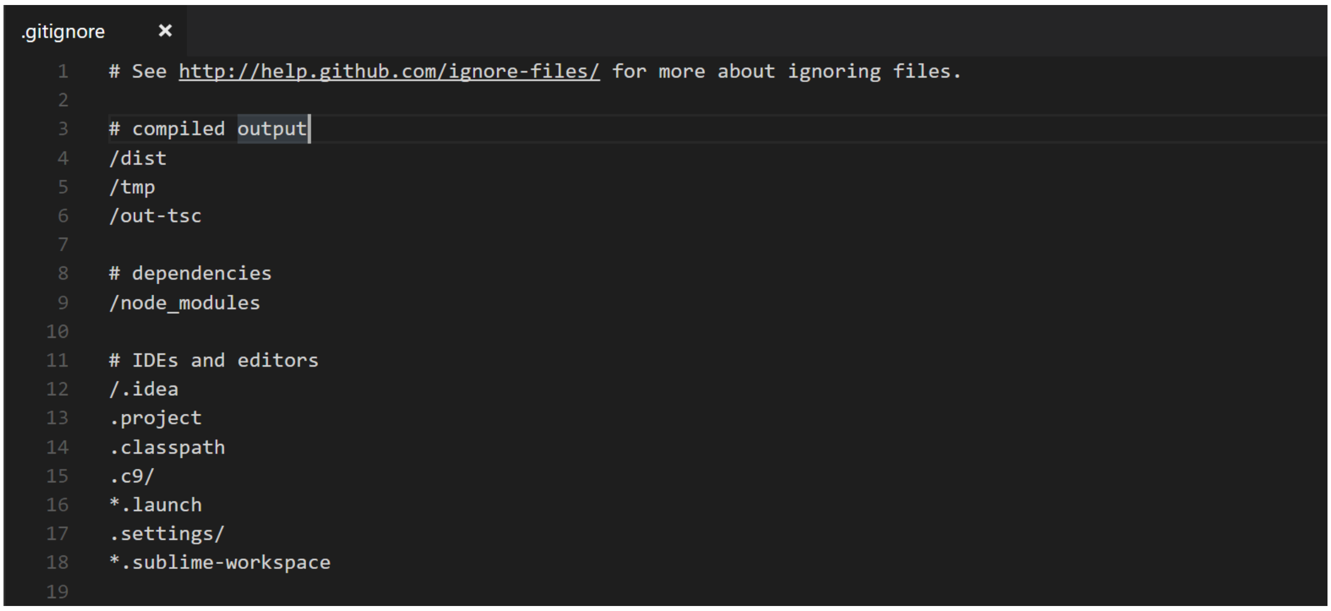
GRAPH 11. Current app configuration

In order to make the code style consistent between different code editors, it is a good idea to have an *EditorConfig* file. It helps developers define and maintain consistent coding styles between different editors and IDEs. The *EditorConfig* project consists of a file format for defining coding styles and a collection of *text editor plugins* that enable editors to read the file format and adhere to defined styles. *EditorConfig* files are easily readable and they work nicely with version control systems (Ensinger 2014). The GRAPH 11 is the editable code snippet for the *.editorconfig* file, it includes more detailed configuration information about the editor which is used for this application.

```
# Editor configuration, see http://editorconfig.org
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 2
insert_final_newline = true
trim_trailing_whitespace = true

[*.md]
max_line_length = off
trim_trailing_whitespace = false
```

GRAPH 12. Code editor configuration

Inside project folder, there are some files that are not changed all the time. So it is a good idea to do not upload these files to the Git when working with version control system. For example, all the list of node modules are not changed, so it is better to ignore it when uploading the project to a version control system via Git. In order to decrease the transferred file size, every application might have one file called *.gitignore* file. The GRAPH 13 is an editable code snippet for the *.gitignore* file. It contains the list of files which can be ignored by *Git* when uploading the source code to the version control system like *GitHub, GitLab* or any other version control systems.

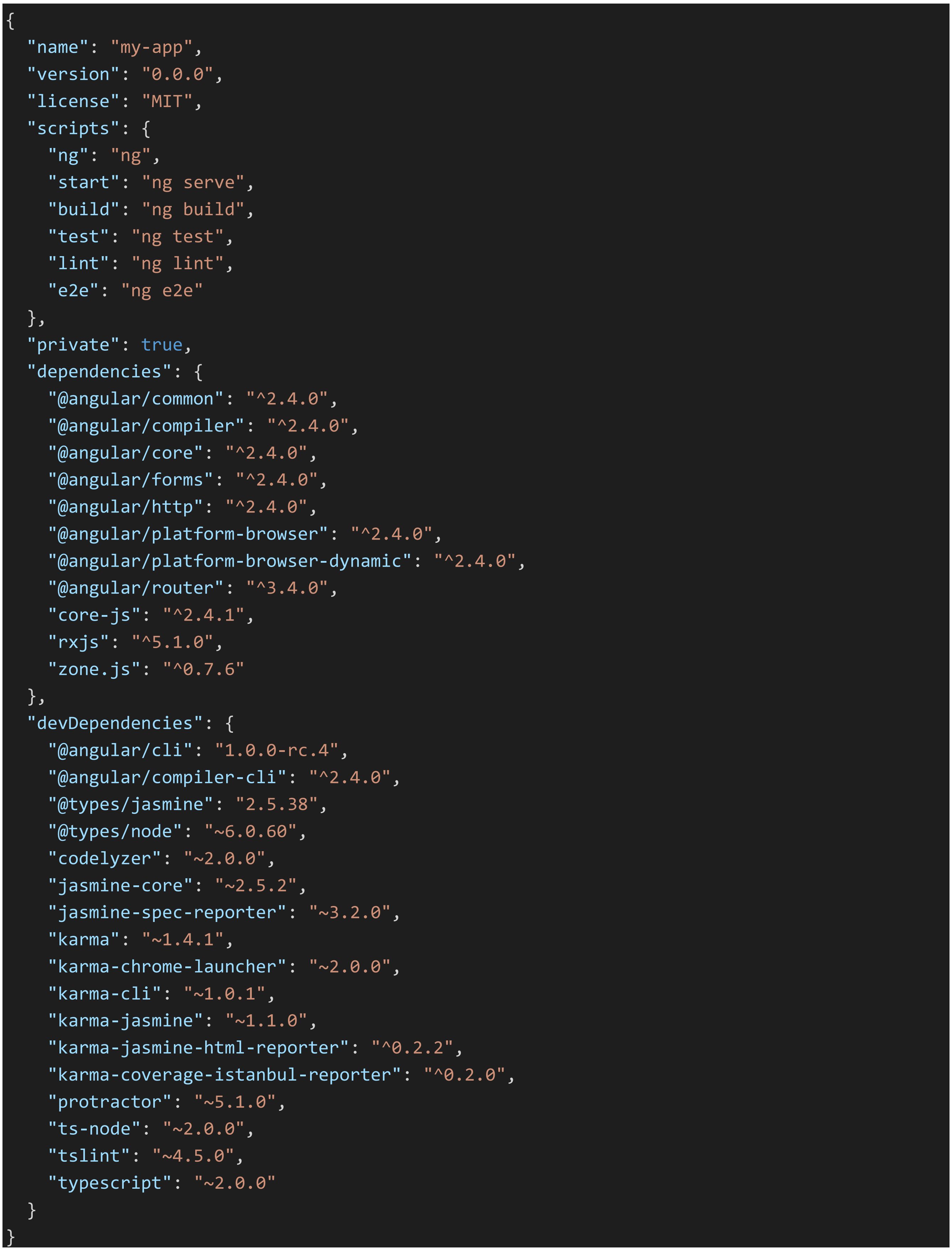


.gitignore

```
# See http://help.github.com/ignore-files/ for more about ignoring files.

# compiled output
/dist
/tmp
/out-tsc

# dependencies
/node_modules

# IDEs and editors
/.idea
.project
.classpath
.c9/
*.launch
.settings/
*.sublime-workspace

```

GRAPH 13. Angular .gitignore file example

Every application has one file to list all the node modules with its name and version specified in it, usually in the project root folder, called *package.json*. This file also holds various metadata relevant to the project. This specific file is used to give information to NPM that allows it to identify the project as well as handle the project's dependencies. It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both NPM and to the end users of the package. The *package.json* file is normally located in the root directory of a Node.js project (Nicoreed 2011). The GRAPH 15 is an editable code snippet for the *package.json* file. It contains the list of Node modules with its name and version.



```json
{
  "name": "my-app",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/common": "^2.4.0",
    "@angular/compiler": "^2.4.0",
    "@angular/core": "^2.4.0",
    "@angular/forms": "^2.4.0",
    "@angular/http": "^2.4.0",
    "@angular/platform-browser": "^2.4.0",
    "@angular/platform-browser-dynamic": "^2.4.0",
    "@angular/router": "^3.4.0",
    "core-js": "^2.4.1",
    "rxjs": "^5.1.0",
    "zone.js": "^0.7.6"
  },
  "devDependencies": {
    "@angular/cli": "1.0.0-rc.4",
    "@angular/compiler-cli": "^2.4.0",
    "@types/jasmine": "2.5.38",
    "@types/node": "~6.0.60",
    "codelyzer": "~2.0.0",
    "jasmine-core": "~2.5.2",
    "jasmine-spec-reporter": "~3.2.0",
    "karma": "~1.4.1",
    "karma-chrome-launcher": "~2.0.0",
    "karma-cli": "~1.0.1",
    "karma-jasmine": "~1.1.0",
    "karma-jasmine-html-reporter": "^0.2.2",
    "karma-coverage-istanbul-reporter": "^0.2.0",
    "protractor": "~5.1.0",
    "ts-node": "~2.0.0",
    "tslint": "~4.5.0",
    "typescript": "~2.0.0"
  }
}
```

GRAPH 14. List of dependencies

Every application folder contains one *README* file. It is a text file containing general information about how to use the application under different environments, such as guideline about running the application, guideline about connecting the server, testing guideline. README files typically contain instructions and additional help as well as details about project updates. It is very useful for people who did not write this code but wanted to use the source code of this application (Computer Hope 2017). The GRAPH 17 is a *readme.my* file. It contains some information about how to run, build and test this application.

```
# MyApp

This project was generated with [Angular CLI](https://github.com/angular/angular-
cli) version 1.0.0-rc.4.

## Development server

Run `ng serve` for a dev server. Navigate to `http://lo-
calhost:4200/`. The app will automatically re-
load if you change any of the source files.
## Code scaffolding

Run `ng generate component component-name` to generate a new compo-
nent. You can also use `ng generate directive/pipe/service/class/module`.

## Build

Run `ng build` to build the project. The build arti-
facts will be stored in the `dist/` directory. Use the `-prod` flag for a produc-
tion build.

## Running unit tests
Run `ng test` to execute the unit tests via [Karma](https://karma-runner.github.io).

## Running end-to-end tests
Run `ng e2e` to execute the end-to-end tests via [Protractor](http://www.protrac-
tortest.org/).
Before running the tests make sure you are serving the app via `ng serve`.

## Further help
To get more help on the Angular CLI use `ng help` or go check out the [Angu-
lar CLI README](https://github.com/angular/angular-cli/blob/master/README.md).
```

GRAPH 15. Running, building and testing guideline

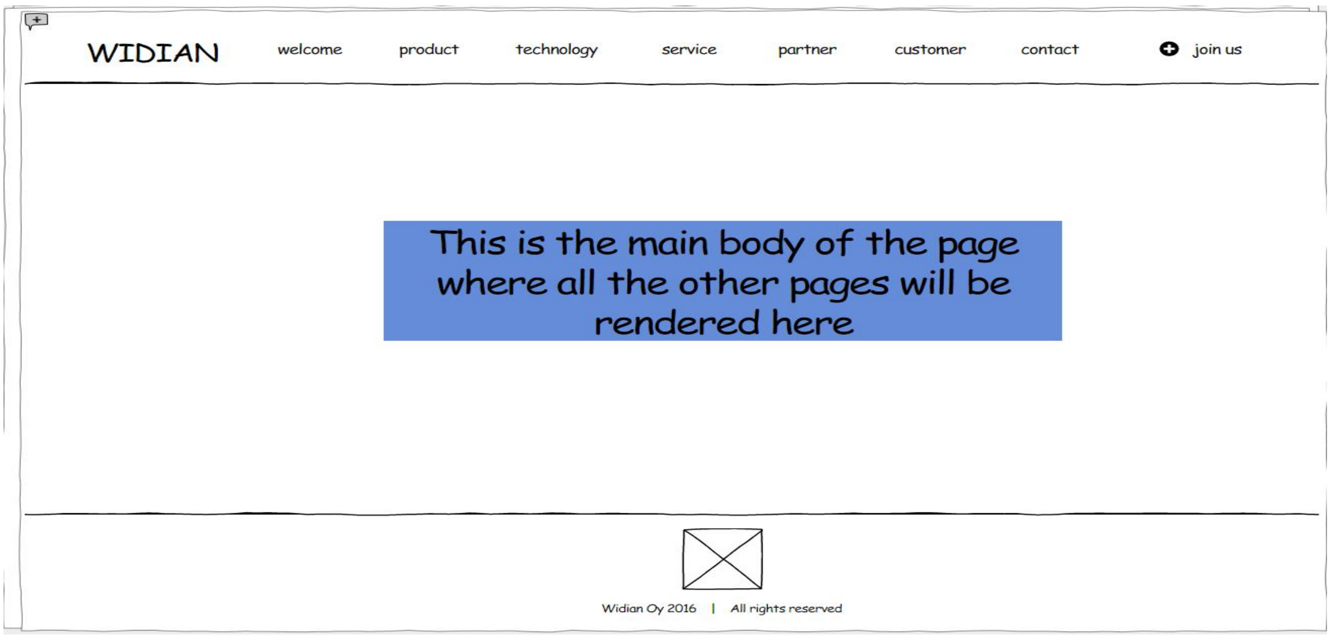
3.3 More about source folder

The *src* folder in the application project folder is the main folder in which the developers are going to spend the most of the time during the creation of this application. It includes all the source code and assets which directly belong to this application. It also includes some configuration files that are specific to this application. In other words, all the dynamic files and folders are located in the *src* folder. All the static files and folders are normally located outside *src* folder. That is why most of the global configuration is done in the file which is located outside *src* file. Here are more explanations about every folder and file in this folder.

The *App* folder contains the current app's modules and components. Every Angular app has at least one component called root component. Here in the app folder, the root component named *app.component* is created for use. It has template file with .html extension, it has style file with *.css* extension, it has test case file with *.spec.ts* extension, it has Typescript class file with *.ts* extension and it has modules files with *.modules.ts* extension. The *Assets* folder contains assets like images, videos, local databases. The *Environment* folder contains some Typescript file which configures the production and development environment.

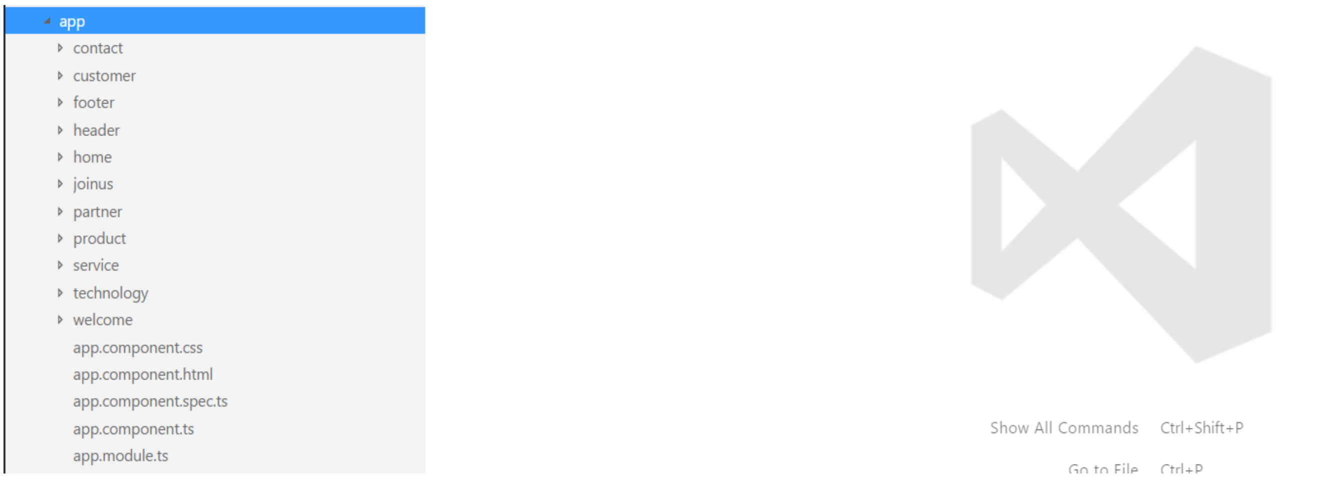
3.4 Design concept and component creation process

This case study application has a simple page layout: common header and footer, main body area in which all the main page content will be rendered. It has eight pages. Every page can be created as one component. In this case, it has *Header* component for the *header* area, which is a common header. It has *Footer* component for the *footer* area, which is also a common footer. It has a *Welcome* component for the *welcome* page. It has *Product* component for the *product* page. It has *Technology* component for the *technology* page. It has *Servic*e component for the *service* page. It has *Partner* component for the *partner* page. It has *Customer* component for the *customer* page. It has *Contact* component for the *contact* page. It has to *Join us* component for the *join us* page. GRAPH 24 is the simple design and wireframe for this Angular app.


WIDIAN
welcome
product
technology
service
partner
customer
contact
join us
This is the main body of the page where all the other pages will be rendered here
Widian Oy 2016 | All rights reserved

GRAPH 16. The simple design and wireframe for this Angular app

Creating a new component is really simple with Angular CLI. The first thing to do is to navigate to *app* folder and simply run the command *ng g –component < component name comes here>*. This is the new feature available in the latest version of AngularJS. This simple command can save time, energy and most importantly, it makes the application creation process more consistent and less troublesome. (Silva 2015). The GRAPH 25 is the illustration of the file structure of project folder after all the components are added.



```
app
  contact
  customer
  footer
  header
  home
  joinus
  partner
  product
  service
  technology
  welcome
  app.component.css
  app.component.html
  app.component.spec.ts
  app.component.ts
  app.module.ts
```

GRAPH 17. New components list in project folder

3.5 Creating file for routing and service

In order to connect the database, service needs to be created in an Angular application. Creating a service is very easy with Angular CLI tool. Just using this *ng g service <name of service>* command is enough to create service (Silva 2015). By default, there are two files in newly created service folders, one is for testing, which has *.spec.ts* extension and the other one is the actual code for the services, which is a normal typescript file with a *.ts* extension. Normally service can be created in the App folder. After that two files will be added to the project folder. The GRAPH 26 is the illustration of name structure of newly created *service* file and its *testing* file.

get-json.service.spec.ts
get-json.service.ts

GRAPH 18. List of files in service folders

Creating a single page application means reducing the page refresh when navigating between pages. In Angular, page navigation always handled by the router. Routing is one of the best features in AngularJS. And it is really easy to use, simply by injecting as a dependency injection. The routing file contains all the configuration about router of the current application (Sevilleja 2013). There is no command for creating router. Normally it can be created manually. The routing file also can be created in *App* folder (Nvamba 2016). The GRAPH 27 is the illustration of name structure of the newly created *routing* file.

app.routing.ts

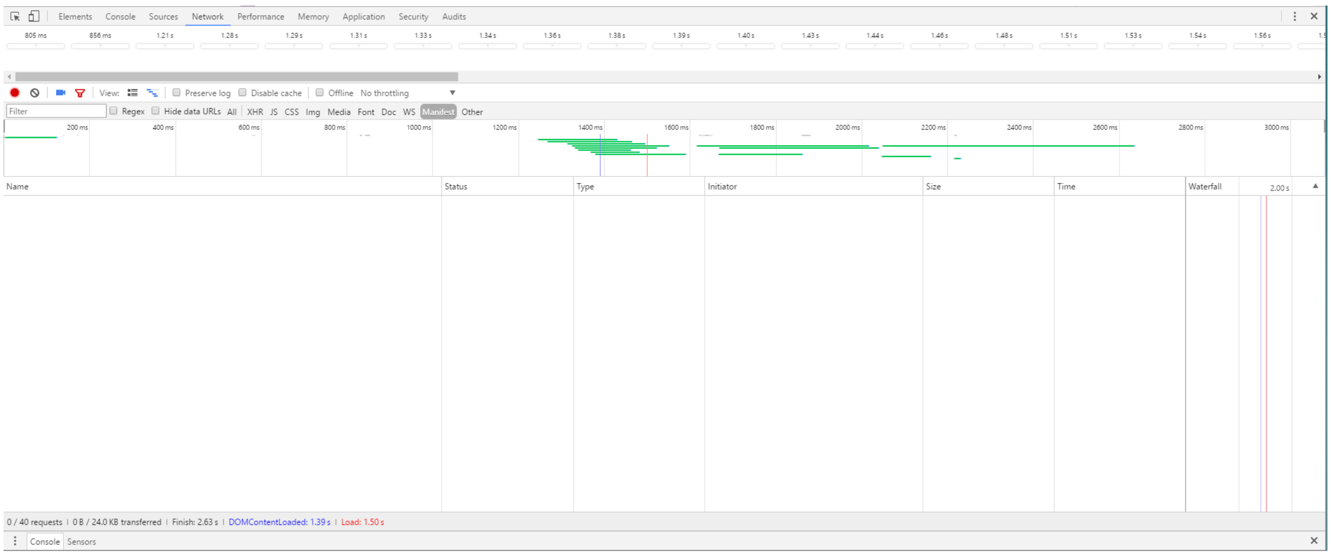
GRAPH 19. Routing file

After all, the components, services, and routing files are ready. The last thing to do now is to create the template code, style code and testing code for every component. But before that implementing the services and router is a must since it is useful when navigating between pages using a router and get some data for data binding from the database. Since this is not a step-by-step tutorial for beginners so the implementation process is omitted. The demo link for the completed application and the source code link of this application will be provided in the reference list. Anyone can use it only for study purpose.

3.6 Collecting page loading event information

In order to prove the fact that Angular based web application can be cost effective, it is planned to do this page loading test with Chrome's development tool. It is a set of web authoring and debugging tools built into Google Chrome. It is the tool that provides web developers deep access into the internals of the browser and their web application. Use it to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization. It is the bridge between developers and browser. This page loading test is implemented based on the following steps (Basques 2017).

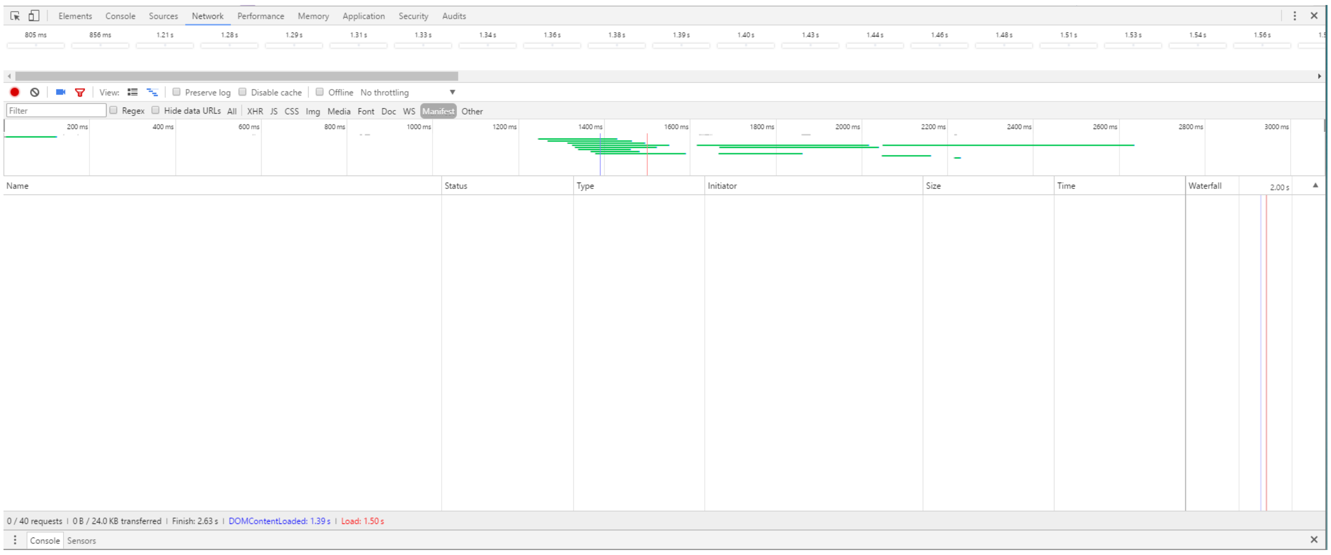
First, load the *home* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 28 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 20. *Home* page loading event information

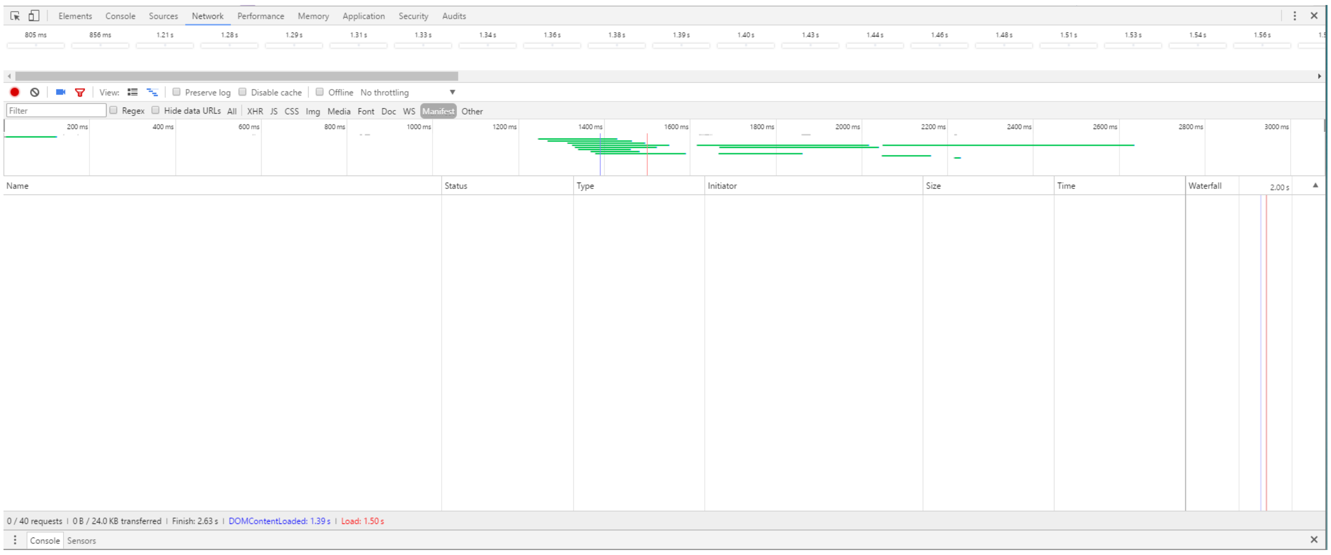
Second, navigate to the *product* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 29 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s

GRAPH 21. *Product* page for loading information

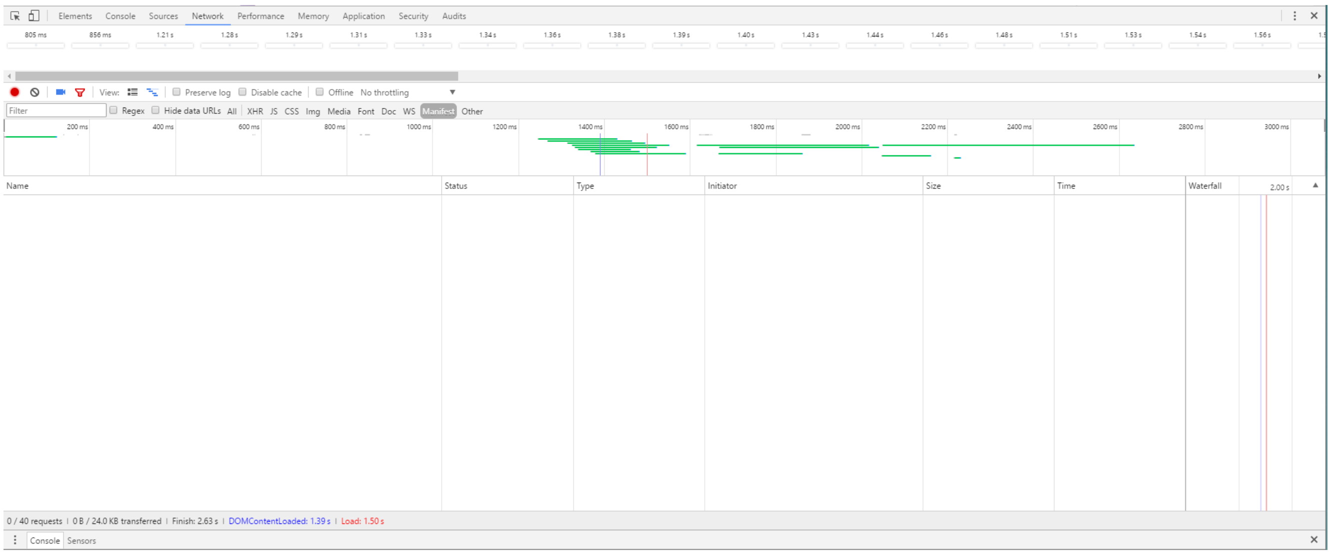
Third, navigate to the *technology* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 30 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 22. *Technology* page loading information

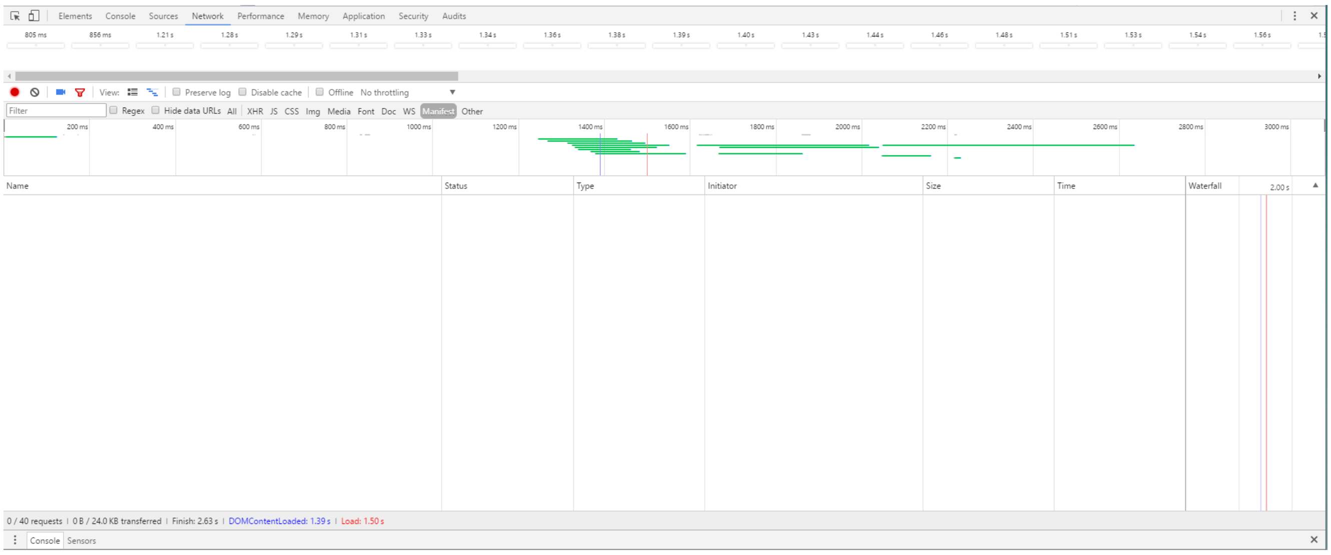
Fourth, navigate to the *service* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 31 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s

GRAPH 23. *Service* page loading information

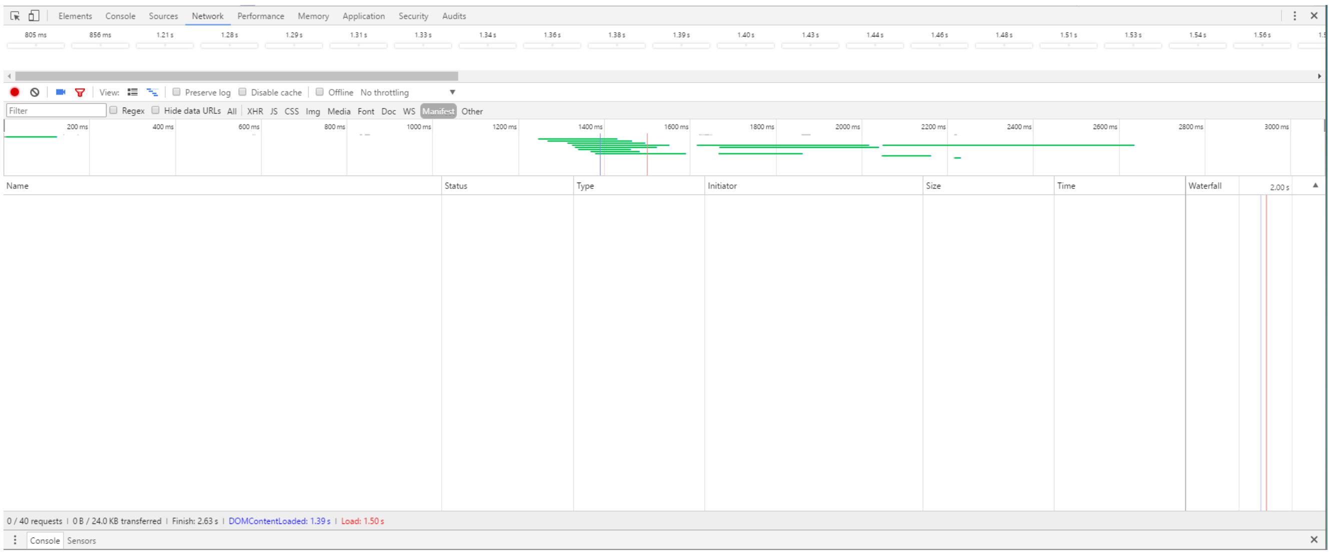
Fifth, navigate to the *partner* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 32 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 24. *Partner* page loading information

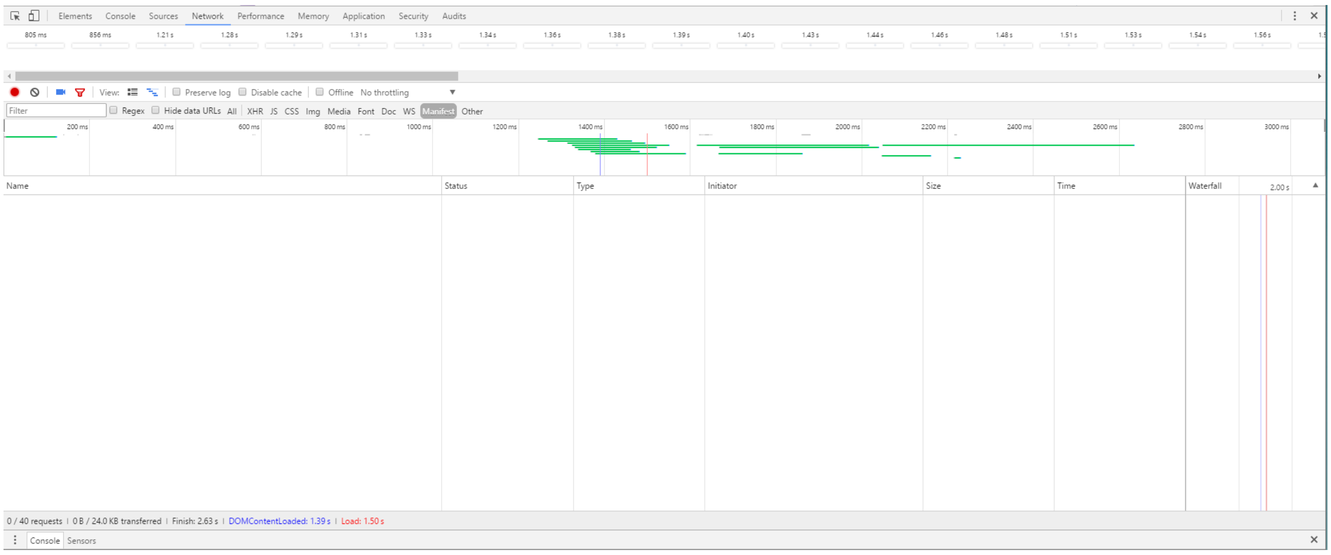
Sixth, navigate to the *customer* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 33 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 25. *Customer* page loading information

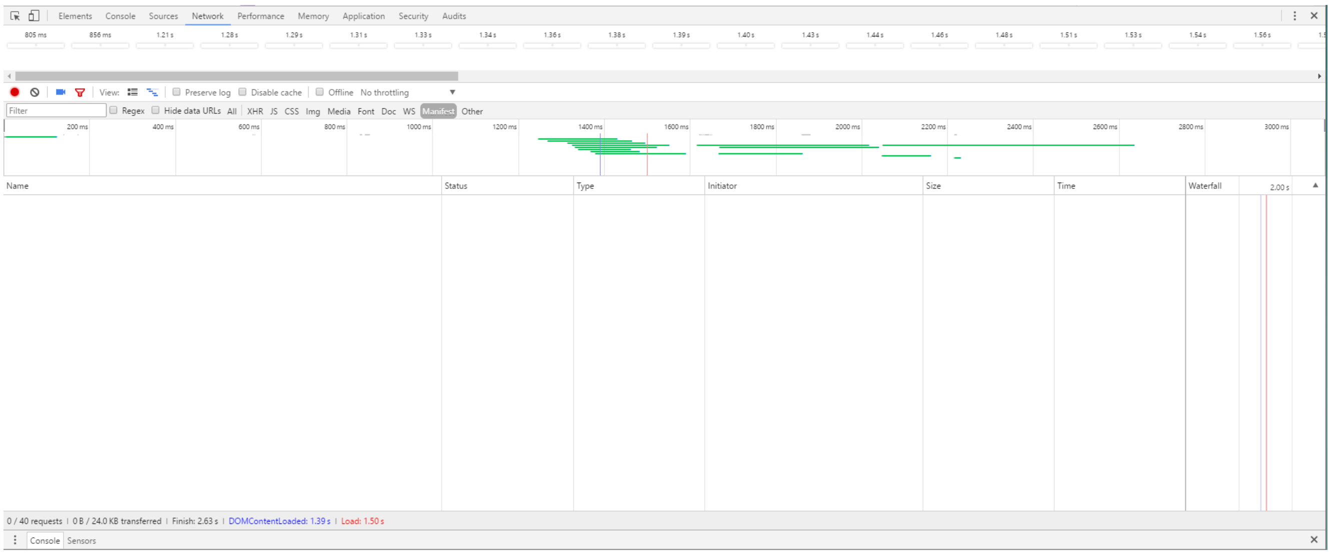
Seventh, navigate to the *contact* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 34 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s

GRAPH 26. *Contact* page loading information

Eighth, navigate to the *join us* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 35 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 27. *Join us* page loading information

TABLE 1 records some detailed numerical information about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load. Based on the page loading test data from above screen captures, the number of requests change on page changes, transferred file size changes on page changes, time for total loading time which is the time user did not interact with web pages at all changes since it depends on how long time the test is done, time for loading the initial markup does not change, time for loading all the page contents also does not change. There is one thing to mention that the test data may be different for the same page at the different time because of the external factors like a server.

TABLE 1. Page loading information for *angular App*

	Number of Requests	**Transferred files/kb**	**Finish/s**	**DOMContentLoaded/s**	**Load/s**
Welcome	22	5.9	1.92	1.58	1.60
Product	23	7.2	37.53	1.58	1.60
Technology	26	8.8	66	1.58	1.60
Service	27	9.9	102	1.58	1.60
Partner	30	11.0	126	1.58	1.60
Customer	33	12.5	150	1.58	1.60
Contact	34	13.3	168	1.58	1.60
Join us	37	20.2	192	1.58	1.60

3.7 Analyzing the page loading event information

In this section, all the data collected from page loading test are analyzed a little more deeply. There are three variables which have consistent change, such as the *Number of Request*, *Transferred files*, *Finish*. The *Number of Request* column shows the total requests which the application supposed to send to the server in order to load the specific page. So whenever the page changes its value changes since every page needs to get different data that is why it sends different requests. The *Transferred files* column indicates the total size of data exchanged between client and server on per request. Based on the different request the exchanged data also changes that is why its value changes on per site changes. The *Finish* column records the time from start till user totally stop interacting with the current website. The value of this column depends on when user totally stops interacting with the website, so its value changes based on the user interactivity time (Basques 2017).

The *DOMContentLoaded* column records the time from start till the initial markup of a page has been parsed. The value in the DOMContentLoaded section will not change on page changes because in AngularJS application the initial markup parsing mostly happens once on the initial loading, so whenever the page changes there is no pages reloading happens after the first load. That is the main reason why the value of DOMContentLoaded does not change when the page changes. Because it does not need to reload the website content again. This is a unique advantage of AngularJS application compared to applications which are created with other frameworks. This will highlight and support the main idea in this thesis (Basques 2017).

The *Loads* column records the time from start till the page fully loaded. The value in the Loads section will not change on page changes because when the AngularJS application launched the whole page loaded on the initial load and mostly it happens once, so whenever the page changes there is no other page reload. That is the main reason why the value of Loads does not change when the page changes. Because the web application already fully loaded on the initial load and no need to reload again, this is also a unique advantage of AngularJS application compared to applications which are not created as a single page application. This will also highlight and support the main idea in this thesis (Basques 2017).

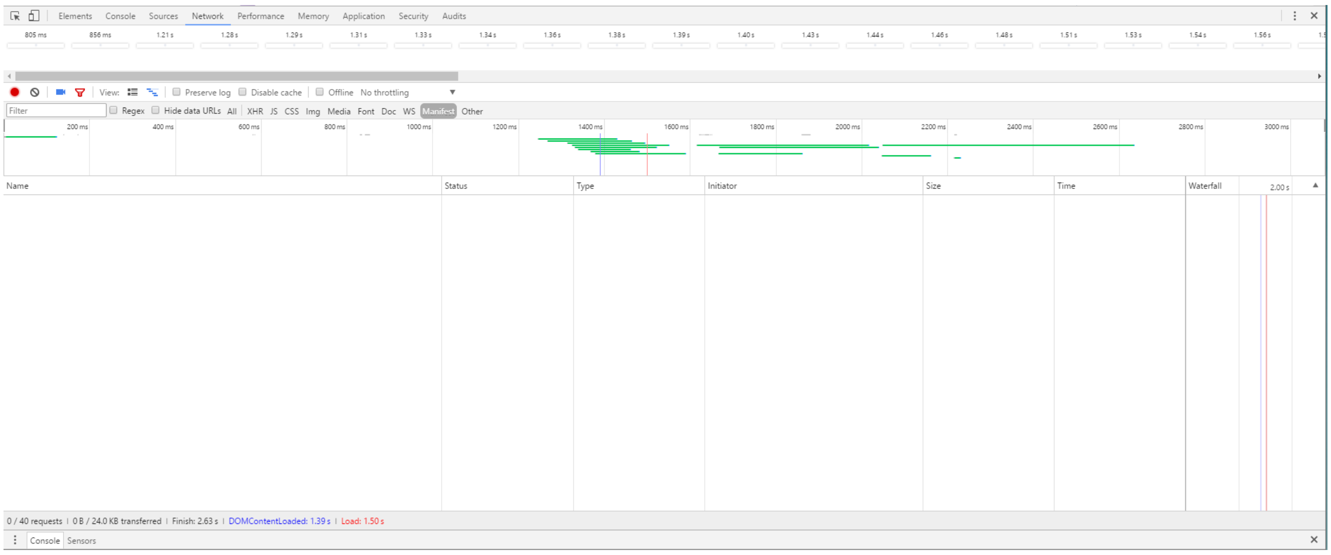
Based on all the information above, it can be easily concluded that Angular based web application does not reload the page when navigation occurs between pages after the initial load is finished. Less page reloading means less data transfer between client and server that is why it can reduce the required bandwidth amount. Eventually, it reduces the hosting cost. (Low 2016).

4 CASE STUDY: CREATING NON-ANGULARJS BASED WEB APPLICATION

In order to highlight the main idea more clearly, in this chapter the page load test is implemented for a web application which did not use AngularJS. The main Idea is to check whether it reloads the entire page when navigating between pages. Because in the end, it is going to be clarified that non-angular based web application has more page reloads compared to angular based web application. This application is created with popular JavaScript framework called Express.js. It is a server-side JavaScript framework (Ornbo 2012). This is the link for the source code of this application: https://github.com/AihaitiAbudureheman/infourmo-new-website

4.1 Collecting page loading event information

First, load the *home* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 36 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



Elements Console Sources Network Performance Memory Application Security Audits

☐ Preserve log ☐ Disable cache ☐ Offline No throttling

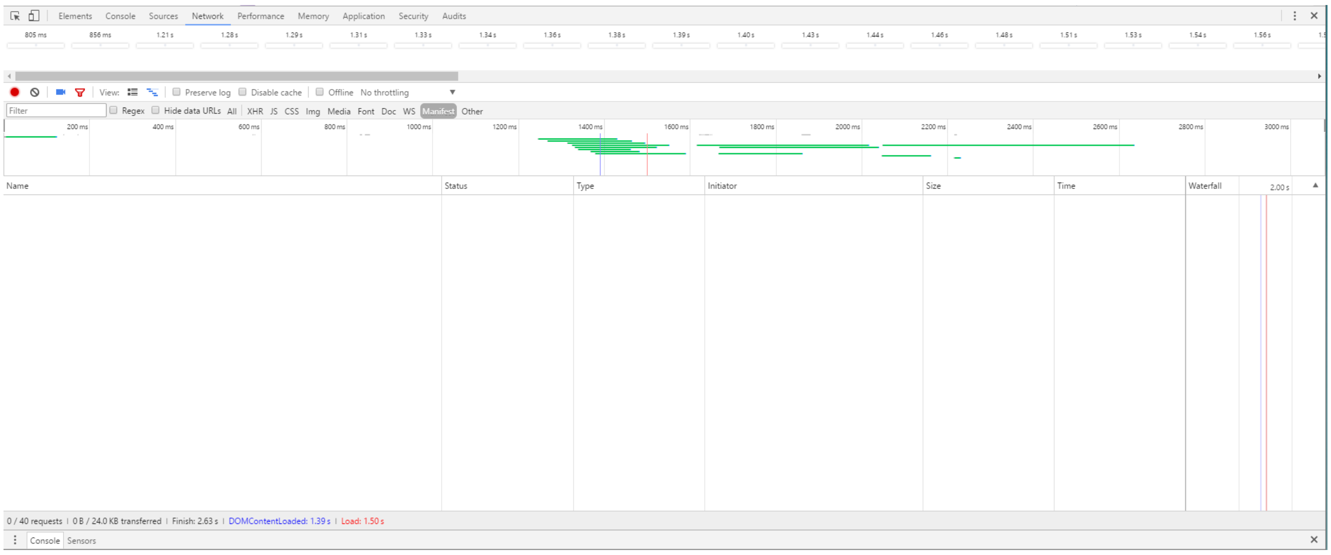
Filter ☐ Regex ☐ Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Status	Type	Initiator	Size	Time	Waterfall

0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s

GRAPH 28. *Home* page loading event information 2

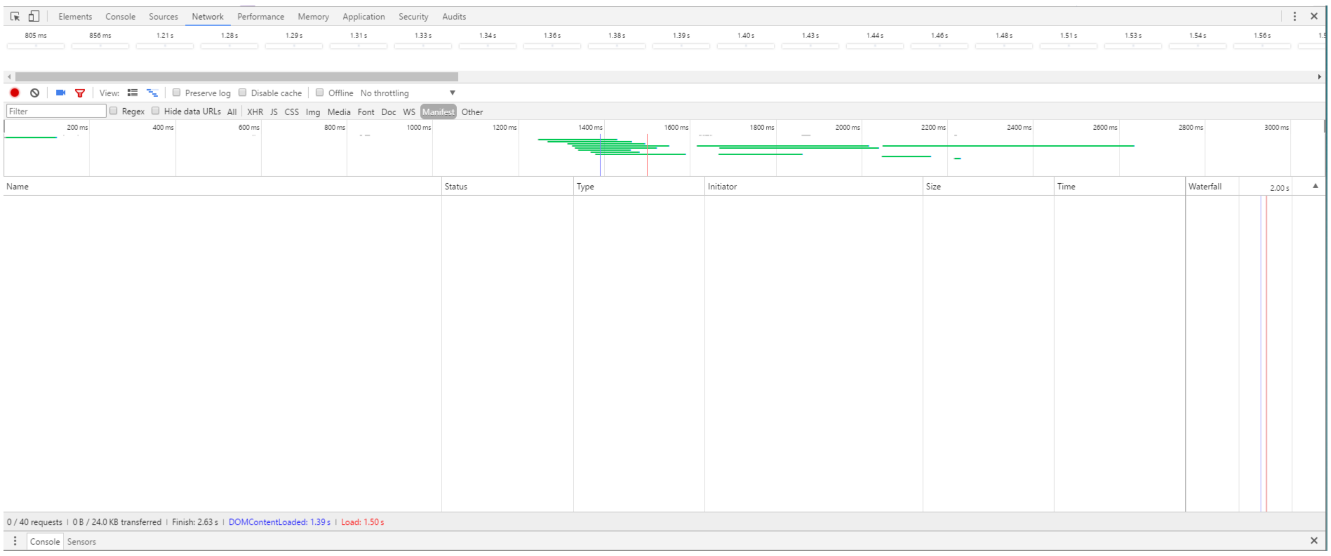
Second, navigate to the *Our Service* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 37 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s

GRAPH 29. *Our Service* page loading information

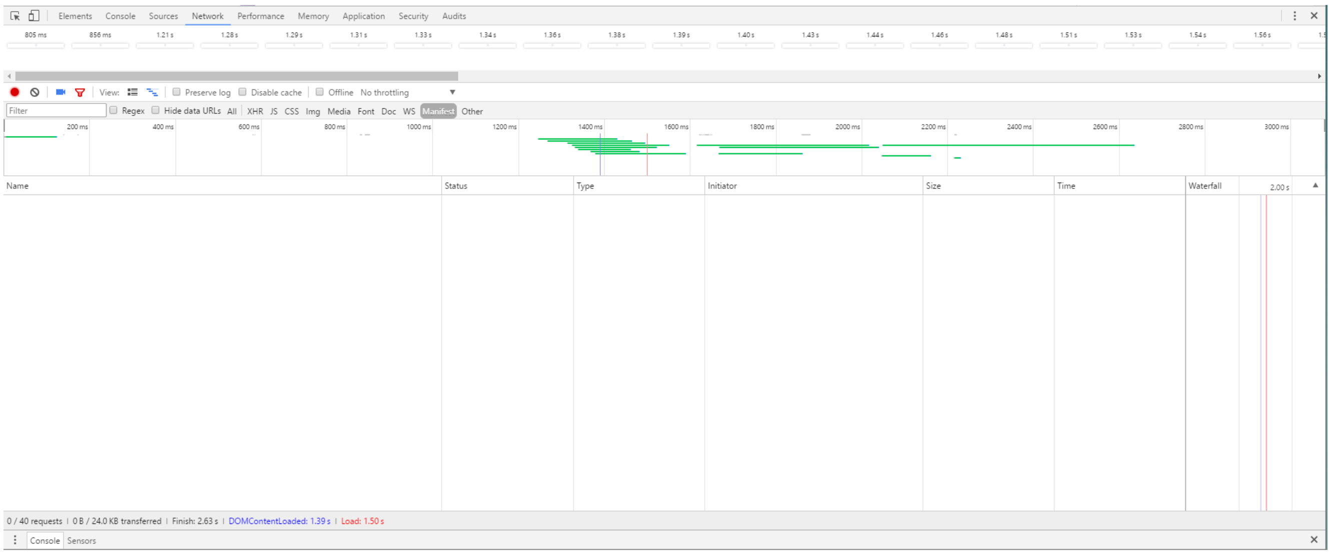
Third, navigate to the *Customer* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 38 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 30. *Customer* page loading information

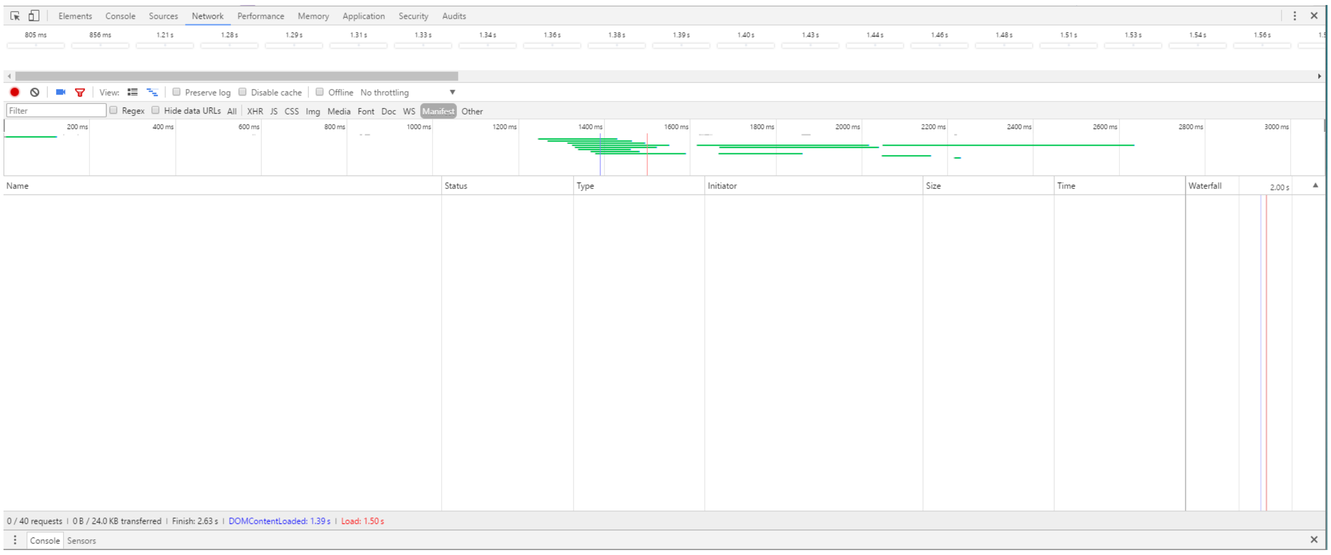
Fourth, navigate to the *Jobs* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 39 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 31. *Jobs* page loading information

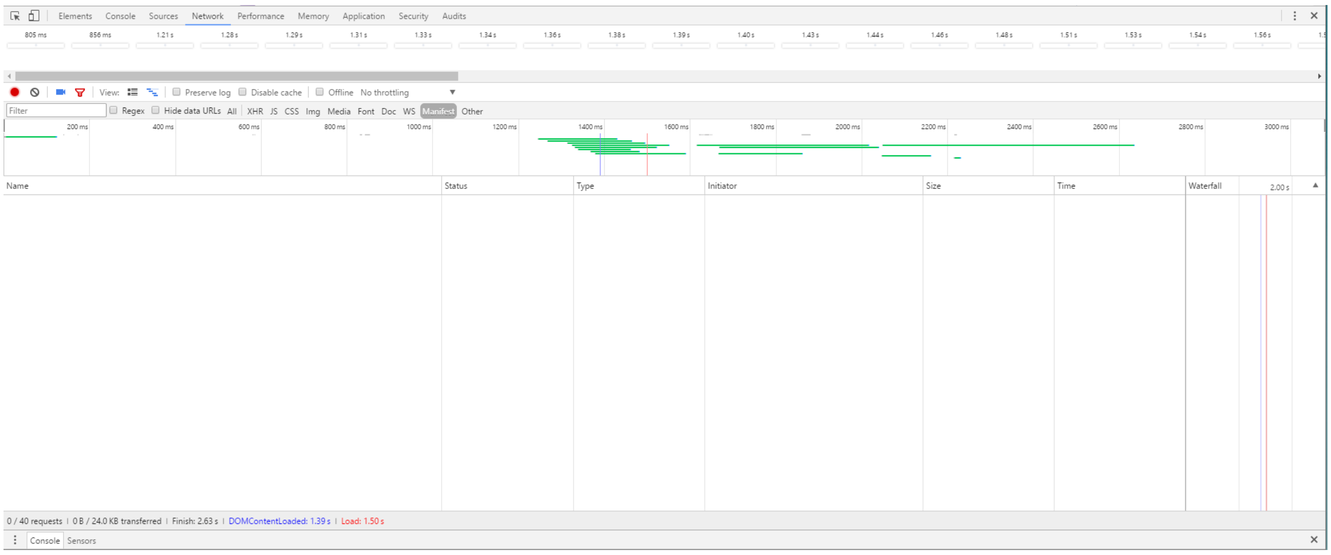
Fifth, navigate to the *News* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 40 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 32. *News* page loading information

Sixth, navigate to the *About Us* page and open the Chrome DevTool. The Chrome DevTools has many options to use and discover more about the application. It has nine sections, such as Elements, Console, Source, Network, Performance, Memory, Application, Security, and Audits. In this page load test, all the relevant data are collected from Network section, which has all the loading information related to this current application. Every time page changes, loading information is updated as well (Basques 2017). The GRAPH 41 is the screen capture of this page loading event. It includes some details about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load.



```
0 / 40 requests | 0 B / 24.0 KB transferred | Finish: 2.63 s | DOMContentLoaded: 1.39 s | Load: 1.50 s
```

GRAPH 33. *About Us* page loading information

TABLE 1 records some detailed numerical information about this page loading event, such as a number of requests, the size of transferred files, time to load the page's initial markup and the total time for full page load. Based on the page loading test data from above screen captures, the number of requests change on page changes, transferred file size changes on page changes, time for total loading time which is the time user did not interact with web pages at all changes since it depends on how long time the test is done, time for loading the initial markup does not change, time for loading all the page contents also does not change. There is one thing to mention that the test data may be different for the same page at the different time because of the external factors like a server..

TABLE 2. Page loading information for *non-Angular App*

	Number of Requests	**Transferred files/kb**	**Finish/s**	**DOMContentLoaded/s**	**Load/s**
Home	24	4.5	0.316	0.241	0.321
Our Service	24	109	0.312	0.276	0.388
Customer	99	1100	1.2	0.461	1.29
Jobs	22	1300	0.976	0.283	0.979
News	23	42	0.302	0.245	0.304
About Us	35	7800	1.67	0.3	1.67

4.2 Analyzing page loading event information

In this section, all the data collected from page loading test are analyzed a little more deeply. The *Number of Request* column shows the total requests which are supposed to be sent to the server in order to load the specific page. So whenever the page changes its value changes since every page needs to get different data that is why it sends different requests. The *Transferred Files* column indicates the total size of data exchanged between client and server on per request. Based on the different request the exchanged data also changes that is why its value changes on per site changes. The *Finish* column record the time for loading the current page (Basques 2017).

The *DOMContentLoaded* column records the time from start till the initial markup of the current page has been parsed. The value in DOMContentLoaded section change on page changes because in the Non-AngularJS application the initial markup parsing mostly happens when page changes, so whenever the page changes there is page reloading happens. That is the main reason why the value of DOMContent-Loaded change when the page changes. Because it needs to reload the website content again. This is one of the disadvantages of Non-AngularJS application compared to applications which are created with AngularJS. This highlights and supports the main idea in this thesis (Basques 2017).

The *Loads* column records the time from start till the page fully loaded. The value in Loads section changes on page changes because, in the non-angular application, the whole page loaded happens on the initial load and mostly it happens again when page changes, so whenever the page changes there is page reload happens. That is the main reason why the value of Loads change when the page changes (Basques 2017).

Based on all the information above, it can be easily concluded that non-angular based web application reload the page when navigation occurs between pages after the initial load is finished. More page re-loading means more data transfer between client and server that is why it can increase the required bandwidth amount. Eventually, it increases the hosting cost. (Low 2016).

5 CONCLUSION

The goal of this thesis was to find out what AngularJS is and how does it work to create single page application and to find out how to develop a single page application with AngularJS. In some point, this thesis has reached its goal. Due to the time limitation, there was no more detailed explanation to make this thesis suitable even for beginners, so all the source codes are upload with more explanation to this thesis's public repository and this will help new beginners to explore the idea behind the single page application. Also, the explanation for the non-angular based application is emitted, of course, the main point is to see the page loading information but it will be useful for new beginners to learn something. It is also uploaded to this thesis's public Github repository.

Angular is a good choice for creating single-page applications and especially well-suited for programmers with a background in JavaScript. It basically allows the developer to develop single page application with basic HTML, CSS and JavaScript skills. Which is a technology with rapid growth though. There are not that many latest books available about angular, so most of the resources are from online.

Based on the experiences while developing this angular application, the latest version of angular is really easy to get started with. It provides angular command line interface by which it is really easy to create a clean project skeleton to get started with. For example, there is no need to configure some bundling tools like *Webpack* and testing tools like *Karma* for the application, instead, all the configurations can be done with one line of command using angular command line interface.

In fact, it would be better to have good knowledge in HTML, CSS, and JavaScript before start coding with angular. During the writing process of this thesis, the biggest problem was a lack of good JavaScript knowledge and HTML, especially the deep knowledge of HTML is very important since the angular itself is affecting the document object model by extending the HTML features.

In conclusion, Angular is a popular framework for creating a single-page application. The Angular based application has less page reload compared to other non-angular based application. The less page reloads means less request to the server and that means less requirement for bandwidth. Eventually, it will save a certain amount of money by saving bandwidth. But based on my experiences, there is one real fact that in order to create an angular based application with a good performance it is necessary to consult the people who have more experiences. And one another important thing is that angular is growing rapidly with a great feature, so it is the best option for an application which has many interactions between client and server.

REFERENCES

Basques, K. 2017. Network Analysis Reference.
Available at: https://developers.google.com/web/tools/chrome-devtools/network-performance/reference#controls
Accessed 06 April 2017.

Braithwite, B. 2015. Getting started with Karma for AngularJS testing
Available at: http://www.bradoncode.com/blog/2015/05/19/karma-angularjs-testing/
Accessed 12 April 2017.

Computer Hope 2017 README file
Available at: http://www.computerhope.com/jargon/r/readme.htm
Accessed 02 May 2017

Ensinger, D. 2014. Why I use EditorConfig
Available at: http://davidensinger.com/2013/07/why-i-use-editorconfig/
Accessed 02 May 2017

Freeman, E. 2012. Head First HTML and CSS: A Learner's Guide to Creating Standards-based web pages. 2nd edition. Sebastopol, CA: O'Reilly Media

Freeman, E. 2014. Head First JavaScript Programming: A Brain-Friendly Guide. 1st edition. Sebastopol, CA: O'Reilly Media

Firebase, 2017. Firebase Real-time database.
Available at: https://firebase.google.com/docs/database/
Accessed 06 April 2017.

Hégaret, P., Wood, L., & Robie, J. 2000. What is document object model?
Available at: https://www.w3.org/TR/DOM-Level-2-Core/introduction.HTML
Accessed 13 March 2017.

Krill, P. 2013. What is so special about Google's AngularJS?
Available at: http://www.infoworld.com/article/2612801/javascript/what-s-so-special-about-google-s-angularjs.HTML
Accessed 06 April 2017.

Kappert, L. 2015. ECMAScript 6(ES6): What's New in the Next Version of JavaScript?
Available at: https://www.smashingmagazine.com/2015/10/es6-whats-new-next-version-javascript/
Accessed 15 March 2017.

Kasireddy, P. 2016. JavaScript Modules: A Beginner Guide.
Available at: https://medium.freecodecamp.com/javascript-modules-a-beginner-s-guide-783f7d7a5fcc#.nrcosqxyy
Accessed 16 March 2017.

Kukic, A. 2014. AngularJS Best Practices: Directory Structure
Available at: https://scotch.io/tutorials/angularjs-best-practices-directory-structure
Accessed 01 May 2017.

Kim, C. 2015. Understanding module.exports and exports in Node.js
Available at: https://www.sitepoint.com/understanding-module-exports-exports-node-js/
Accessed 02 May 2017

Kayal, S. 2014. Dependency Injection in Controller of AngularJS.
Available at: https://www.codeproject.com/Tips/806530/Dependency-Injection-in-Controller-of-Angu-larJs
Accessed 20 March 2017.
Loiselle, S. 2013. What is web hosting?
Available at: https://www.godaddy.com/garage/smallbusiness/launch/what-is-web-hosting/.
Accessed 13 March 2017.

Low, J. 2016. How much bandwidth does your site really need?
Available at: http://www.webhostingsecretrevealed.net/blog/web-hosting-guides/how-much-band-width-does-your-site-really-need/
Accessed 06 April 2017.

Microsoft, 2017. Typescript.
Available at: https://github.com/Microsoft/TypeScript.
Accessed 13 January 2017.

Mitchell, B. 2016. What is bandwidth?
Available at: https://www.lifewire.com/what-is-bandwidth-p2-818121.
Accessed 13 March 2017.

Mitchell, B. 2017. What is a server?
Available at: https://www.lifewire.com/servers-in-computer-networking-817380
Accessed 13 March 2017.

Node.js Foundation. 2017. Download Node.js
Available at: https://nodejs.org/en/
Accessed 25 March 2017.

Nicoreed, N. 2011. What is the package.json file?
Available at: https://docs.nodejitsu.com/articles/getting-started/npm/what-is-the-file-package-json/
Accessed 02 May 2017.

Nwamba, C. 2016. Routing Angular 2 Single Page Apps with the Component Router
Available at: https://scotch.io/tutorials/routing-angular-2-single-page-apps-with-the-component-router
Accessed 06 April 2017.

Ornbo, G. 2012. Introducing Express, the Web Framework for Node.js
Available at: http://www.informit.com/articles/article.aspx?p=1858263
Accessed 03 May 2017

Patrick, S. 2015. What is DomContentLoded?
Available at: https://varvy.com/performance/domcontentloaded.HTML
Accessed 22 March 2017.

Scott, C & Ben, Straub. 2014. Pro Git. 2nd Edition. New York: Apress Media LLC

Segue Technologies. 2013. What is Ajax and where it is used in Technology?
Available at: http://www.seguetech.com/ajax-technology/.
Accessed 14 March 2017.

Shrivastava, N. 2014. Introduction to Angular.
Available at: https://www.codeproject.com/Articles/803294/Part-Introduction-to-AngularJS
Accessed 16 March 2017.

Sergei, S. 2016. What is Git and should I use it.
Available at: https://www.quora.com/What-is-git-and-why-should-I-use-it
Accessed 06 April 2017.

Silva, F. 2017. CLI tool for Angular
Available at: https://github.com/filipesilva
Accessed 06 April 2017.

Schenker , G. 2014. AngularJS – Part 14, End to end tests
Available at: https://lostechies.com/gabrielschenker/2014/04/18/angular-jspart-14-end-to-end-tests/
Accessed 01 May 2017.

Sevilleja, C. 2013. Single page apps with AngularJS routing and templating
Available at: https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templating
Accessed 03 May 2017

Trivedi, J. 2015. Data Binding in AngularJS.
Available at: http://www.c-sharpcorner.com/UploadFile/ff2f08/data-binding-in-angularjs/
Accessed 20 March 2017.

Victoria, W. 2013. Revealed, what happens in just ONE minute on the internet: 216,000 photos posted, 278,000 Tweets and 1.8m Facebook likes?
Available at: http://www.dailymail.co.uk/sciencetech/article-2381188/Revealed-happens-just-ONE-minute-internet-216-000-photos-posted-278-000-Tweets-1-8m-Facebook-likes.HTML.

Accessed 14 March 2017.

Wahlin, D. 2014. Creating Custom Angular Directives Part I – The Fundamentals.
Available at: https://weblogs.asp.net/dwahlin/creating-custom-angularjs-directives-part-i-the-funda-mentals
Accessed 20 March 2017.

Wahlin, D. 2013-2014. AngularJS in 60 Minutes. Chandler, Arizona: Wahlin Consultin