



Interview

Questions And Answers

Shailendra Chauhan

Founder and CEO - Dot Net Tricks
Corporate Trainer and Microsoft MVP



Node.js Interview Questions and Answers

All rights reserved. No part of this book can be reproduced or stored in any retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, uploading on server and scanning without the prior written permission of the author.

The author of this book has endeavored to ensure the accuracy of the information described in this book. However, the author cannot guarantee the accuracy related to any interpretation the reader may assume from reading it.

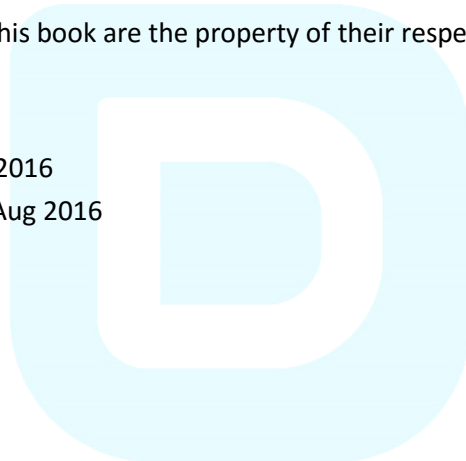
The author will not be liable for any damages, incidental or consequential caused directly or indirectly by this book or any of the information provided in it.

Further, readers should be aware that the websites or reference links listed in this book may have changed or disappeared between when this book was written and when it is read.

All other trademarks referred to in this book are the property of their respective owners.

Release History

- Initial Release 1.0.0 - 8th Jul 2016
- Second Release 1.0.1 - 12th Aug 2016



Dedication

I dedicate this book to my mother Vriksha Devi and my wife Reshu Chauhan, who inspired me to write this Book. They deserve to have their name on the cover as much as I do for all IT professionals, their support made this possible.

It is my deepest sincerity that I say to all my family members Virendra Singh(father), Jaishree and Jyoti(sisters), Saksham and Pranay(sons) and to my friends along with all of you who follow my blog www.dotnet-tricks.com without your continued encouragement and belief in me this book would not have been feasible.

Shailendra Chauhan



Introduction

Writing a book has never been an easy task. It takes great effort, consistency with strong determination most importantly must have wisdom over the subject on which you are going to write.

What Where Author Qualification to Write This Book

Shailendra Chauhan is rewarded as **MVP** by Microsoft for his exceptional contribution in Microsoft **Visual Studio and Development Technologies**. With more than 7 years in hand experience over Microsoft technologies and collection of other technologies including **JavaScript, MVC, AngularJS, Node.js, Ionic, MongoDB and NoSQL Databases** to name but a few.

His inspiration for authoring come from his enthusiasm for technology, analytic and initiative nature. Being a **trainer, architect and blogger** and passion for helping people inspire him for writing.

Enormous feedback and support from his previous books which appreciated by all user inspire him to write Node.js interview questions and answers book.

What This Book Is

Node.js Interview Questions and Answers book aims to save your time and make you more productive Node.js programmer. This book will provide a comprehensive guide for creating, debugging, testing and deploying production-ready Node.js applications. In this book each topic has been explained using appropriate questions with best appropriate answers and suitable examples.

What You'll Learn

This book will help you to prepare yourself for interview on Node.js technology as well as it guides you to learn Node.js core modules and Node.js frameworks - Express, Koa, Socket.IO, and Node.js ORM/ODM - sequelize/Mongoose and Node.js testing frameworks - Mocha, Chai and JS task runners - Grunt, Gulp.

Share to Help Others

Hope you will enjoy this book and find it valuable for yourself. The author is truly delighted about this if you will share this book among others because he wants it to reach as many techy people as possible.

Keep Connected with Us

We always post about latest technologies updates on our website so that techy people keep themselves up to date. That's why, we please to suggest you to subscribe yourself on www.dotnettricks.com.

To get the latest release on Node.js, we encourage you to follow the official Node.js website at www.nodejs.org

Our best wishes always with you for your interview and growth!

About the Author

Shailendra Chauhan - An Entrepreneur, Author, Architect, Corporate Trainer, and Microsoft MVP



With more than 7 years in hand experience, **Shailendra Chauhan** is a polymath in the domains of **Microsoft .NET technologies** and an array of other technologies including **JavaScript, AngularJS, Node.js, Ionic and NoSQL Databases** to name but a few.

His extensive knowledge has been broadcast in a number of **articles-of-the-day**, and reported in **daily-community-spotlight** along with being listed in [Recommended Resources for MVC](#) section on the Official **Microsoft ASP.NET Community Site**. He has rewarded as **Microsoft MVP** for his exceptional contributions in Microsoft **Visual Studio and Development Technologies**.

Shailendra is the author of some of most popular e-books which encompass technical Interview on [AngularJS Interview Questions and Answers](#), [LINQ Interview Questions and Answers](#) and [MVC Interview Questions and Answers](#). Furthermore, he is a technical reviewer for book on [ASP.NET MVC 4 Mobile App Development](#).

Shailendra Chauhan is renowned for sharing his working experience, research and knowledge through his reputed and widely subscribed to blogs - www.dotnet-tricks.com and www.dotnettricks.com. Specifically, his blog www.dotnet-tricks.com provides a vast storehouse of knowledge and support resources in the field of .NET technologies worldwide and is listed as a [non-Microsoft resource](#) in The Microsoft Official Community Site. His blogs and training approach provide an easily accessible and highly innovative learning experience for people everywhere, and from a range of backgrounds and skill levels thereby offering the ultimate in training around the world.

Moreover, and to his credit he has delivered **200+ training sessions** to professionals world-wide in Microsoft .NET technologies and other technologies including JavaScript, AngularJS, Node.js, Ionic and NoSQL Databases. In addition, he provides Instructor-led online and classroom training programs for all above technologies.

Shailendra's strong combination of **technical skills and solution development for complex application architecture with proven leadership and motivational skills** have elevated him to a world renowned status, placing him at the top of the list of most sought after trainers.

"I always keep up with new technologies and learning new skills to deliver the best to my students" says Shailendra Chauhan, he goes on to acknowledge that the betterment of his followers and enabling his students to realize their goals are his prime objective and a great source of motivation and satisfaction.

Shailendra Chauhan - **"Follow me and you too will have the key that opens the door to success"**

How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something which strikes you as odd, or you find an error in the book please drop a line via e-mail.

Our e-mail addresses are listed as follows:

- shailendra@dotnettricks.com
- info@dotnettricks.com

Follow Dot Net Tricks

We sincerely hope you will enjoy reading this book and it will help you to get better understanding of the topic.

You can also follow us on [facebook](#), [twitter](#), [linkedin](#), [google plus](#) or subscribe to [RSS feed](#).

Thank you for reading!!



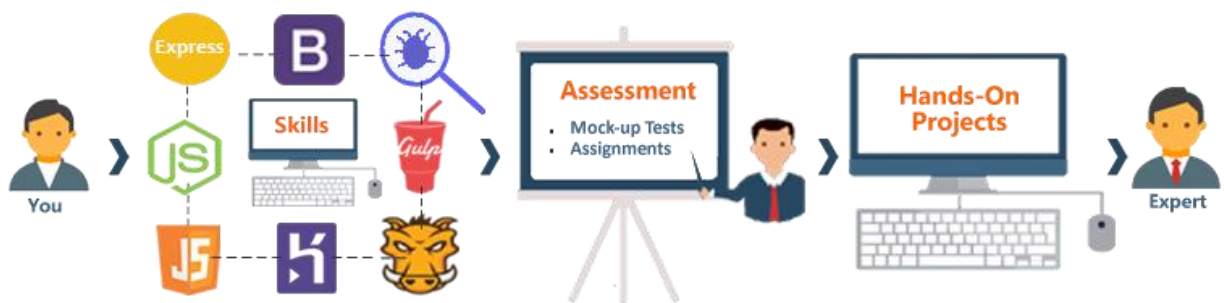
Our Most Accepted Training Programs by Professionals

Node.js Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The Node.js Development course is primarily designed for UI Developer(s)/Web Developer(s) who want to learn how to develop high performance, event-driven, real-time and scalable networking and web applications. In this course, you will learn the Node.js, Express, Bootstrap, Unit Testing and JavaScript Task runner (Grunt, Gulp) fundamental like JavaScript event loop, express routes, unit test cases, HTTP web server, handle requests & responses, event-driven communications, handling databases and much more...



Course objectives

At the completion of this course, attendees will be able to;

- Describe JavaScript weird parts like prototype, objects, lexical scope, dynamic scope and closure.
- Explore Node.js basic and advanced in-depth concepts.
- Create and consume REST API.
- Create HTTP web server and handling requests & responses.
- Handle Sessions, Cookies, and Session Stores.
- Perform Node.js database operations with SQL and NoSQL Databases.
- Write Unit Test cases using Jasmine/Mocha and Chai.
- Use HTML Template engines like Handlebars and Jade.
- Install and Publish Node's Package Manager – NPM.
- Use JS Task runner like Gulp or Grunt.
- Publish their Node.js App on cloud server Heroku.

START LEARNING WITH A FREE DEMO SESSION

CALL +91 987 174 9695

AngularJS Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The AngularJS Development course is primarily designed for UI Developer(s)/Web Developer(s) who want to learn how to create single page web application using AngularJS. In this course, you will learn the JavaScript, Angular fundamental like bootstrap process, directives, statements, routes, filters, services, components etc. that help you to build rich, scalable, extensible and high performance single page web application and much more...



Course objectives

At the completion of this course, attendees will be able to;

- Understand most popular JavaScript MVW Framework
- Understand AngularJS basic and advanced in-depth concepts
- Create custom directives, filters and different types of scopes in directive
- Use Bower to manage front-end libraries and frameworks
- Create and consume REST service
- Optimize web pages performance
- Secure application based on user roles
- Scale SPA for Phone, Tablets, Laptop & Desktop using Bootstrap
- Publish their App

START LEARNING WITH A FREE DEMO SESSION

CALL +91 987 174 9695

Hybrid Mobile Apps Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The Hybrid Mobile Apps Development course is primarily designed for UI Developer(s)/Web Developer(s) who want to learn how to create cross platform mobile apps using Ionic, Cordova and AngularJS. In this course, you will learn how to use AngularJS and Ionic for creating widgets, data listing, spinners etc. and creating mobile apps using mobile camera, audio, video and database and finally publish it on Google Store, Apple Store.



Course objectives

At the completion of this course, attendees will be able to;

- Explore most popular JavaScript MVW Framework.
- Understand AngularJS basic and advanced in-depth concepts.
- Create custom directives, filters and different types of scopes in directive.
- Create and consume REST API.
- Learn Mobile apps development options and advantages of Hybrid mobile apps.
- Explore Ionic HTML, CSS and JS components.
- Use gestures, and tools for building highly interactive mobile apps.
- Use Ionic CLI, Ionic Creator, Ionic Lab and Ionic View
- Understand Cordova basic and advanced in-depth concepts.
- Access mobile native features like camera, contacts, calendar etc.
- Publish mobile app on Google Store, Apple Store.

START LEARNING WITH A FREE DEMO SESSION

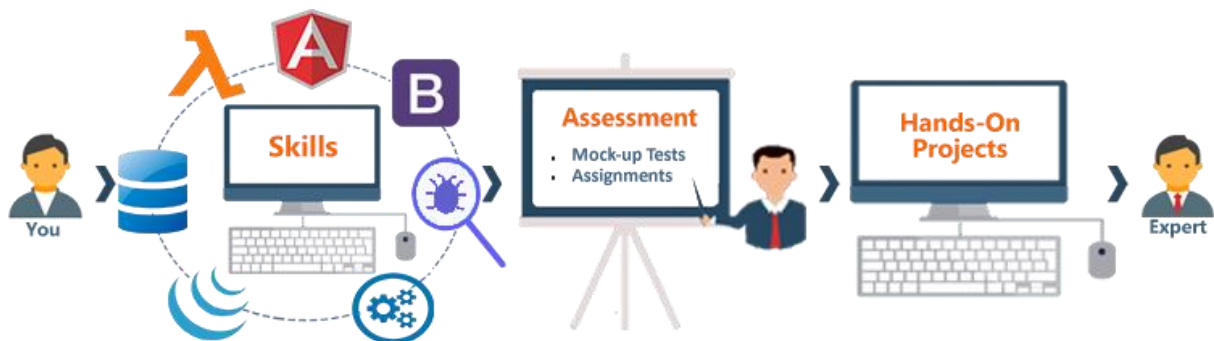
CALL +91 987 174 9695

ASP.NET MVC with AngularJS Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The ASP.NET MVC with AngularJS Development course is primarily designed for .NET Beginner(s)/Professional(s) who want to learn how to create web applications using ASP.NET MVC. In this course, you will learn how to create web pages, custom validation attributes, custom helpers, querying database using Entity Framework ORM, making interactive using AngularJS and jQuery, mobile friendly application and finally publish it on IIS.



Course objectives

At the completion of this course, attendees will be able to;

- Create custom html helpers and validations attributes
- Use jQuery to make user friendly web pages
- Query database using Entity Framework ORM code first approach
- Design and develop scalable architecture
- Create DAL Layer using Entity Framework Code First approach
- Scale application for Phone, Tablets, Laptop & Desktop using Bootstrap
- Create interactive app using AngularJS
- Create REST service using Web API
- Do errors logging by using ELMAH
- Write Unit Tests using MS Test, xUnit and Moq
- Optimize web page performance
- Secure application based on user roles
- Publish their Apps on IIS

START LEARNING WITH A FREE DEMO SESSION

CALL +91 987 174 9695

ASP.NET Core Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The ASP.NET Core course is primarily designed for .NET Beginner(s)/Professional(s) who want to learn how to create web applications using cross-platform ASP.NET Core. In this course, you will learn how to create web pages, custom validation attributes, tag helpers, custom helpers, middleware, querying database using Entity Framework Core, using JavaScript Task runners - grunt and gulp, mobile friendly application and finally publish it.



Course objectives

At the completion of this course, attendees will be able to;

- Explain ASP.NET Core Goals and History
- Develop ASP.NET Core apps with Visual Studio Code (on Windows, Mac, or Linux)
- Understand ASP.NET Core startup process
- Understand ASP.NET Core middleware
- Configure ASP.NET Core MVC
- Use Dependency Injection in ASP.NET Core
- Work with Entity Framework Core
- Handle Errors in ASP.NET Core
- Create REST Service
- Write Unit Tests for MVC apps
- Design and develop scalable architecture
- Scale application for Phone, Tablets, Laptop & Desktop using Bootstrap
- Secure application based on user roles
- Use JS Task runner like Gulp or Grunt
- Publish their Apps

START LEARNING WITH A FREE DEMO SESSION

CALL +91 987 174 969

MEAN Stack Development Training by Shailendra Chauhan

About the Course

[Download Course Agenda](#)

The MEAN Stack Development course is primarily designed for UI Developer(s)/Web Developer(s) who want to learn how to develop high performance, event-driven, real-time and web applications. In this course, you will learn the MongoDB, ExpressJS, AngularJS and NodeJS fundamental like database, express routes, bootstrap process, directives, statements, routes, filters, services, HTTP web server, handle requests & responses, handling databases, mobile friendly application and finally publish it.



Course objectives

At the completion of this course, attendees will be able to;

- Explore most popular JavaScript MVW Framework
- Create custom directives, filters and different types of scopes in directive
- Create and consume REST API
- Create HTTP web server and handle requests & responses
- Handle Sessions, Cookies, and Session Stores
- Install and work with MongoDB and Node.js
- Use HTML Templating engines like Jade and EJS.
- Use Node's Package Manager – NPM.
- Design schema using MongoDB and writing queries.
- Use JS Task runner like Gulp or Grunt.
- Publish their Node.js App on cloud server Heroku.

START LEARNING WITH A FREE DEMO SESSION

CALL +91 987 174 969

Table of Contents

Node.js Interview Questions and Answers	1
Dedication	2
Introduction	3
About the Author	4
How to Contact Us	5
Our Most Accepted Training Programs by Professionals	6
Node.js Development Training by Shailendra Chauhan	6
About the Course	6
Course objectives	6
AngularJS Development Training by Shailendra Chauhan	7
About the Course	7
Course objectives	7
Hybrid Mobile Apps Development Training by Shailendra Chauhan	8
About the Course	8
Course objectives	8
ASP.NET MVC with AngularJS Development Training by Shailendra Chauhan	9
About the Course	9
Course objectives	9
ASP.NET Core Development Training by Shailendra Chauhan	10
About the Course	10
Course objectives	10
MEAN Stack Development Training by Shailendra Chauhan	11
About the Course	11
Course objectives	11
Node.js	17
Q1. What is Node.js?	17
Q2. Who has developed Node.js?	17
Q3. What is io.js?	17

Q4.	What is Node.js foundation?	18
Q5.	What is NPM?	18
Q6.	What is V8 JavaScript Engine?	18
Q7.	Why to use JavaScript on Server Side?	19
Q8.	What IDEs you can use for Node.js development?	19
Q9.	How Node.js is different from others Server Side Frameworks like ASP.NET, Php, JSP and Ruby etc.? ..	19
Q10.	Where you can deploy Node application?	20
Q11.	Which types of application can be developed using Node.js?	20
Q12.	Who are using Node.js for their development?	21
Q13.	What platforms Node.js supports?	21
Q14.	What are the advantages of Node.js?	21
Q15.	What are the limitations of Node.js?	22
Q16.	Where you can deploy Node.js web application?	22
Q17.	Explain Blocking and non-blocking operations?	22
Q18.	Explain Node.js Event Loop?	23
Q19.	What is callback?	24
Q20.	What is Callback Hell and how to avoid it?	24
Q21.	Explain Node.js Architecture?	25
Q22.	What is Module?	26
Q23.	What are core modules in Node.js?	26
Q24.	What is REPL Terminal?	27
Q25.	What are the useful commands of REPL Terminal?	28
Q26.	What are the useful shortcuts of REPL Terminal?	28
Q27.	What are Node packages and how to install/uninstall them?	28
Q28.	What are the various options to download the version of a node module?	30
Q29.	What is package.json file in node project?	30
Q30.	What is the difference between Package dependencies and development dependencies?	32
Q31.	What is the difference between module.exports and exports in Node.js?	32
Q32.	How to share a variable between Node.js modules?	33
Q33.	What are global variables in Node.js?	33
Q34.	What are buffers?	34
Q35.	How to create buffers?	34

Q36.	How to decode buffers?	34
Q37.	What are Streams?	35
Q38.	What are the different types of Streams?.....	35
Q39.	What are Events and Event Emitters?	35
Q40.	What events are supported by readable streams?	36
Q41.	Give an example of readable stream with non-flowing mode?	36
Q42.	Give an example of readable stream with flowing mode?	36
Q43.	What events are supported by writable streams?	37
Q44.	Give an example of writable stream?.....	37
Q45.	What is Streams piping?	38
Q46.	How to debug the code in Node.js?	38
Q47.	How to debug the code using Node.js Built-In debugger?.....	38
Q48.	How to debug the Node.js code using Node Inspector?.....	39
Q49.	How to debug the Node.js code using IDE debuggers?	40
Q50.	How to handle errors in Node.js?.....	41
Q51.	What are uses of path module in Node.js?	42
Q52.	What functions are provided by path module in Node.js?	42
Q53.	What is File System module in Node.js?	42
Q54.	What functions are provided by fs module to perform I/O operations on a file?	43
Q55.	What functions are provided by fs module to perform I/O operations on a directory?	46
Q56.	How to check a file/directory existence in node.js?	47
Q57.	Which types of network application you can build using node.js?	47
Q58.	How to create TCP Server and TCP Client?.....	47
Q59.	What is socket?.....	48
Q60.	What are WebSockets?	49
Q61.	How to create an Http Server using Node.js?	49
Q62.	What are Node.js Http module limitations?	49
Q63.	What is SSL/TLS?.....	49
Q64.	How to create an Https Server using Node.js?.....	50
Q65.	How to create a self-signed certificate in Node.js?	51
Q66.	What is socket.io?	51
Q67.	How to create a simple chat app using socket.io?	52

Q68.	What is middleware?.....	54
Q69.	What is connect?	54
Q70.	What are various node.js web development frameworks?	54
Q71.	What are various node.js REST API frameworks?	55
Q72.	What is Express?.....	55
Q73.	Why to use Express?.....	56
Q74.	How to create an Http server using Express?.....	56
Q75.	How to define routes using Express?	56
Q76.	What is the use of Express Router Class?.....	57
Q77.	What is template engine?	57
Q78.	What template engines you can use with express?	57
Q79.	What are the advantages of template engines?	58
Q80.	What is Koa?	58
Q81.	How to create an Http server using Koa?.....	58
Q82.	What is Unit Testing?	59
Q83.	Why to do Unit Testing?	59
Q84.	What are node.js unit testing frameworks?.....	59
Q85.	What is Test Driven Development (TDD)?.....	59
Q86.	What is Behavior Driven Development (BDD)?.....	60
Q87.	What is Acceptance Test Driven Development (ATDD)?.....	60
Q88.	What is Jasmine?	61
Q89.	What is Mocha?.....	61
Q90.	What are the most popular Node.js ORM?	61
Q91.	What are JavaScript task runners?	61
Q92.	What are Grunt and Gulp?	62
Q93.	What is the difference between Grunt and Gulp?	63
Q94.	How Grunt and Gulp deal with Task Automation?.....	63
Q95.	What is MEAN stack?.....	64
Q96.	Why to use MEAN stack?	64
Q97.	What is MongoDB?.....	65
Q98.	Why to use MongoDB?.....	65
Q99.	What is AngularJS?	65

Q100. What is the recommended folder structure for MEAN app development?	66
Other Free E-Books	67



Node.js

Q1. What is Node.js?

Ans. Node.js is a server side JavaScript environment for developing web applications like as ASP.NET, JSP, Php etc. It is an open-source and cross-platform framework based on Google's V8 JavaScript Engine.



It is used to build fast and scalable network applications as well as data-intensive real-time web applications. All versions of Node.js are starting from 0.1.0 releases to 0.1.x, 0.2.x, 0.3.x, 0.4.x, 0.5.x, 0.6.x, 0.7.x, 0.8.x, 0.9.x, 0.10.x, 0.11.x, and 0.12.x. Before merging of Node.js and io.js, it's last versions was Node.js v0.12.9.

What Node.js is not?

- Node.js is not a JavaScript library, but it is a platform to execute JavaScript on server side.
- Node.js programs are written in JavaScript but there is no DOM manipulation provided by Node.js.

Q2. Who has developed Node.js?

Ans. Node.js was developed by **Ryan Dahl** and other developers working at Joyent. It was first released in 2009 supporting only Linux. In 2011, windows version was released.

Q3. What is io.js?

Ans. io.js is a fork of the Node.js project which was created in December 2014. It was created to accelerate the development and predicted releases of code under an "open governance model". Since, Node.js was governed by Joyent Inc.



All versions of io.js are starting from 1.0 releases to 1.x, 2.x and 3.x. Before merging of Node.js and io.js, it's last versions was io.js v3.3.1.

Note: In Sep 14, 2015, Node.js and io.js are merged into a single code base known as Node.js version 4.0. In this way, the much-awaited Node.js version 1.0 never happened.

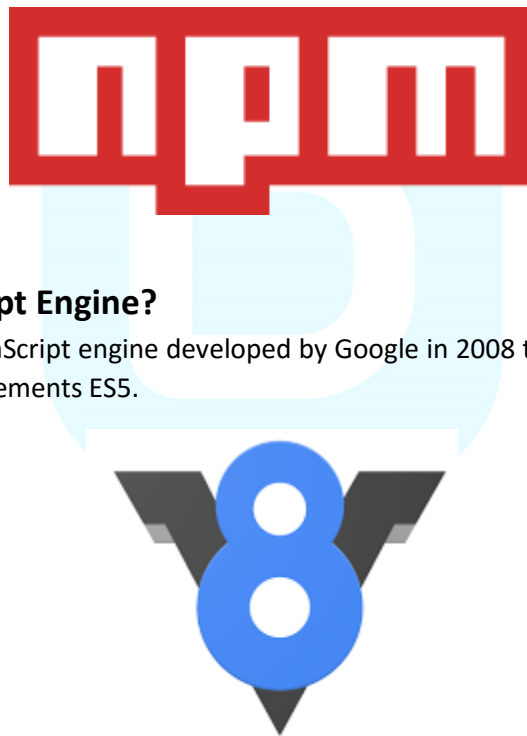
Q4. What is Node.js foundation?

Ans. The Node.js foundation is an independent foundation to take care of development and releases of Node.js. It has developers from IBM, Microsoft, PayPal, Joyent, Fidelity, SAP and other companies.

In Sep 14, 2015, the Node.js foundation announced the combined release of Node.js and io.js into a single code base known as Node.js version 4.0. It has a features of Node.js and io.js including a lot of new features of ES6.

Q5. What is NPM?

Ans. NPM stands for Node Package Manager. It's an online repository of node packages. It was released in 2011 to share and update open-source libraries like jQuery, AngularJS, React etc.



Q6. What is V8 JavaScript Engine?

Ans. V8 is an open source JavaScript engine developed by Google in 2008 to be used in Chrome browser. It is written in C++ language and implements ES5.

V8 JavaScript Engine

Key Points about V8 JavaScript Engine

- It can be run standalone or can be embedded into any C++ application.
- It uses just-in-time compilation (JIT) to execute JavaScript code.
- It compiles JavaScript to native machine code (IA-32, x86-64, ARM, or MIPS ISAs) before execution.
- It is used by many open source projects like Node.js and MongoDB to execute JavaScript on server side.

Q7. Why to use JavaScript on Server Side?

Ans. There are following reasons to use JavaScript on Server Side:

- Unified language for both front-end and back-end.
- Increase programmer productivity.
- Code reusability.
- Exchange of data using JSON.
- JavaScript with V8 engine performs faster than Php, Ruby, Python, JSP and ASP.NET.

Q8. What IDEs you can use for Node.js development?

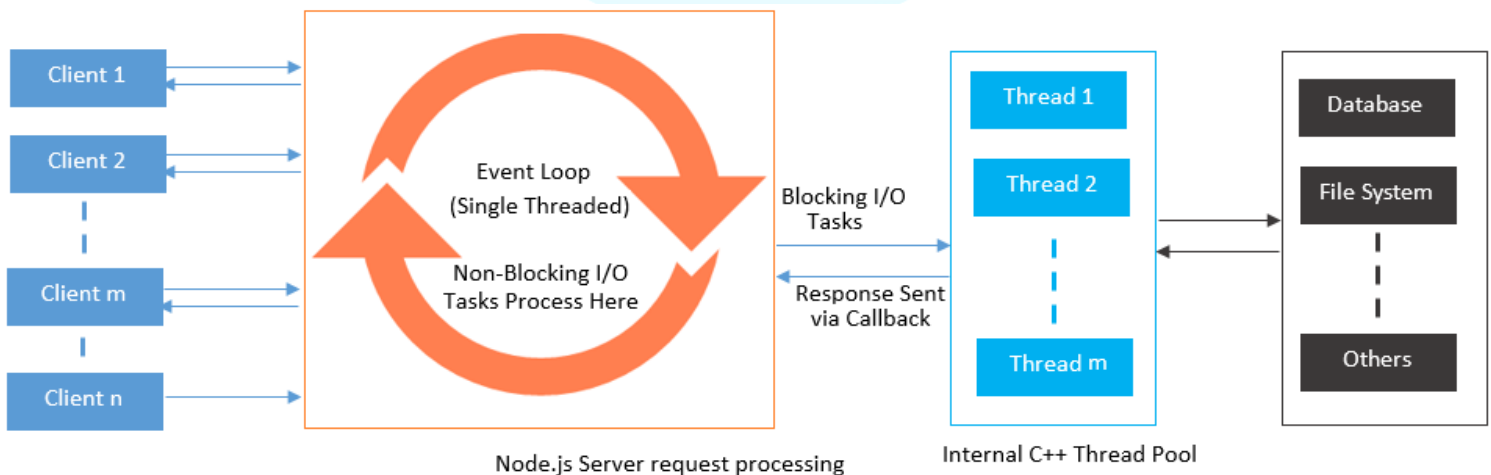
Ans. AngularJS development can be done with the help of following IDEs:

1. Visual Studio 2013, 2015 or higher
2. Visual Studio Code
3. Atom
4. Node Eclipse
5. WebStorm
6. Sublime Text

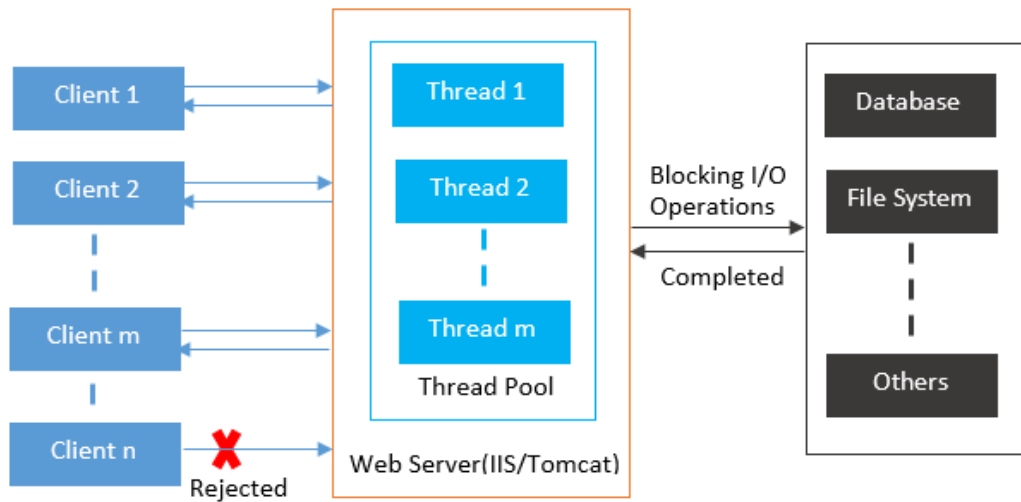
Q9. How Node.js is different from others Server Side Frameworks like ASP.NET, Php, JSP and Ruby etc.?

Ans. Today, Node.js is the most popular and widely used server side framework for small, large and any sized web app and web application development.

Node.js is different from existing server-side frameworks because it is based on asynchronous events via JavaScript callback functionality and uses the JavaScript as a programming language. Moreover, everything inside Node.js runs in single thread.



While existing server-side framework like ASP.NET, JSP and Php etc. are based on multiple threads web server (IIS/Tomcat). In multiple threads system, there is a limit of maximum number of threads, beyond which the throughput decreases.



Multi-Threaded Web Server request processing

There are following issues with Multi-threaded systems:

- Under heavy load a multi-threaded web server consumes a large amount of memory.
- Most of the time threads wait till some I/O operations finish.
- Context-switching and scheduling increases drastically with large number of threads.

Q10. Where you can deploy Node application?

Ans. Node.js app cannot be deployed on your existing hosts like shared web hosting etc. You can use VPS or dedicated servers to install node and run your application.

The easiest way to deploy your node application is to use a scalable service like Heroku, which is completely free and you only need to pay when you are using more resources.

Q11. Which types of application can be developed using Node.js?

Ans. Node.js can be used to develop following types of application:

- E-Commerce Web Applications
- Social Media Applications
- Proxy Server
- Real-time Services
- Real-time data Applications like Multiplayer Games, Stock Trading, Chat App etc.
- Data Streaming Applications
- Network Applications
- High Concurrency Applications
- File Uploading Tools
- Process Monitoring Tools
- HTTP Web Server

Q12. Who are using Node.js for their development?

Ans. Following companies are using Node.js for their development:

- Walmart
- E-bay
- PayPal
- Microsoft
- LinkedIn
- Yahoo
- Google
- SAP
- IBM
- Strong Loop
- Dropbox

Q13. What platforms Node.js supports?

Ans. Node.js supports following platforms:

1. Linux
2. Windows
3. Mac OS X
4. SunOS

Q14. What are the advantages of Node.js?

Ans. The advantages of node.js are given below:

Open Source – Node.js is open source, so it's free to use and no need to pay for license. There are also many open source modules supported by Node.js.

JavaScript as Programming Language – It uses JavaScript as a programming language for both front-end and back-end which increase programmer productivity and code reusability.

Scalable – You can scale your Node.js application by using two ways – Horizontal Scaling and Vertical Scaling, which helps you to improve your application performance.

- In Horizontal scaling you can add more nodes to your existing system.
- In Vertical scaling you can add more resources to a single node.

Better Performance – It provides better performance, since Node.js I/O operations are non-blocking. Also, it uses V8 JavaScript engine to execute JavaScript code. V8 engine compiles the JS code directly into machine code which make it fast.

Caching Support – Node.js supports caching of modules. Hence, when a Node.js modules is requested first time, it is cached into the application memory. So next calls for loading the same module may not cause the module code to be executed again.

Lightweight and Extensible – Node.js is based on JavaScript which can be executed on client side as well as server side. Also, it supports exchange of data using JSON which is easily consumed by JavaScript. This makes it light weight as compared to other frameworks.

Node.js is open source. Hence you can extend it as per your need.

REST API Support – Using Node.js you can also develop RESTful services API easily.

Unit Testing – It supports unit testing out of box. You can use any JS unit testing frameworks like Jasmin to test your Node.js code.

Server Development – Node.js has some built-in API which help you to create different types of Server like HTTP Server, DNS Server, TCP Server etc.

Community Support – Node.js has a wide community of developers around the world. They are active in development of new modules or packages to support different types of applications development.

Q15. What are the limitations of Node.js?

Ans. There are following limitations of Node.js:

1. It doesn't support multi-threaded programming.
2. It doesn't support very high computational intensive tasks. When it executes long running task, it will queue all the incoming requests to wait for execution, since it follows JavaScript event loop which is single threaded.
3. Node is not good for executing synchronous and CPU intensive tasks.

Q16. Where you can deploy Node.js web application?

Ans. The easiest way to deploy your Node.js web application is using Cloud server hosting like Windows Azure, Aws, Google, Heroku etc.

Note

- Node.js app cannot be deployed on the shared web hosting.
- Node.js application you can be deploy on VPS and dedicated servers after installing Node.js.

Q17. Explain Blocking and non-blocking operations?

Ans. By default, In Node.js all I/O operations like file I/O, database operations, network operations etc. are non-blocking. When these operations are completed, they are notified to the main thread using callback functions. You can also write blocking code for these operations using blocking version of Node.js functions which have *Sync* word as suffix.

Let's understand the blocking code and Non-blocking code with the help of example.

Blocking Code: The given code is a blocking code, since it will not continue the execution of the next lines of code until the read operation is completed.

```
var fs = require('fs');  
  
//blocking code
```

```
var data = fs.readFileSync('text.txt', 'utf8'); //wait for reading
console.log(data);
console.log("Done!");

/* Output
 * Hello Node.js
 * Done!
 */
```

Un-blocking Code: The above blocking code can be converted into non-blocking code with the help of callback function. Hence, the given code will continue the execution of the next lines of code without waiting for read operation to be complete.

```
var fs = require('fs');

//un-blocking code with the help of callback function
fs.readFile('text.txt', 'utf8', function (err, data) { //doesn't wait and will
  read file asynchronously
    console.log(data);
  });
console.log("Done!");

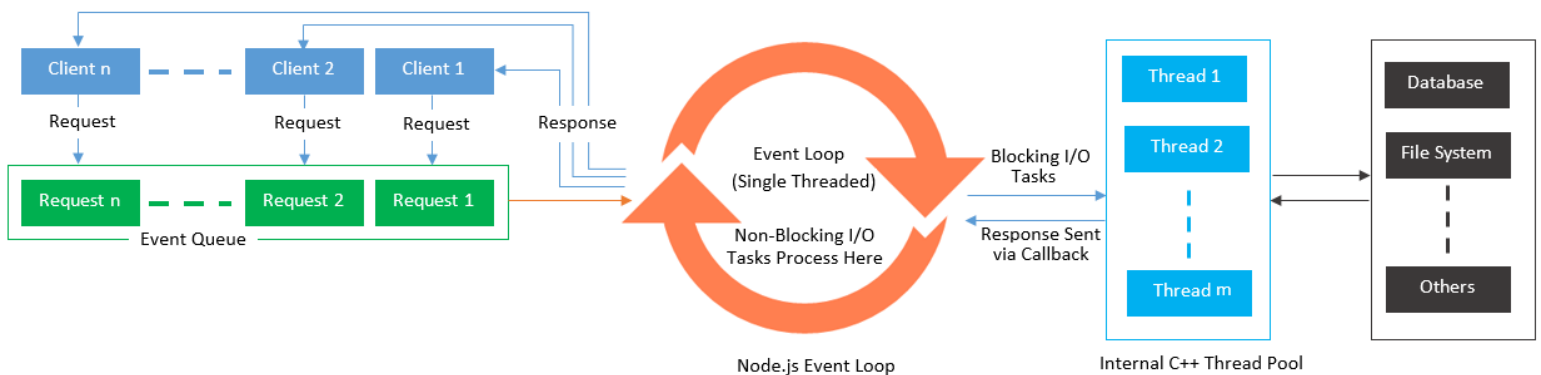
/* Output
 * Done!
 * Hello Node.js
 */
```

Q18. Explain Node.js Event Loop?

OR

Explain Node.js Code Execution Process?

Ans. Node.js execution model is based on JavaScript Event loop which is single threaded and based on JavaScript callback mechanism. The event loop facilitates the Node.js to handle concurrent requests. Let's understand the Node.js execution model with the help of following diagram.



The steps of code execution are given below:

1. Clients send their request to Node.js Server.
2. Node.js Server receives those requests and places them into a processing Queue that is known as “*Event Queue*”.
3. Node.js uses JavaScript Event Loop to process each client request. This Event loop is an indefinite loop to receive requests and process them. Event Loop continuously checks for client’s requests placed in Event Queue. If requests are available, then it process them one by one.
4. If the client request does not require any blocking IO operations, then it process everything, prepare the response and send it back to the client.
5. If the client request requires some blocking IO operations like interacting with database, file system, external services then it uses C++ thread pool to process these operations.

Thread pool process steps are as:

- Event loop checks for thread availability from internal thread pool. If thread is available, then picks up one thread and assign the client request to that thread.
- Now, this thread is responsible for handling that request, process it, perform blocking IO operations, prepare the response and send it back to the Event Loop.
- Finally, the Event Loop sends the response to the respective client.

Q19. What is callback?

Ans. A callback is a function which passed as an argument to an asynchronous function, that describes what to do after the asynchronous operation has completed. Callbacks are used frequently in node.js development.

```
var fs = require('fs');

//callback function to read file data
fs.readFile('text.txt', 'utf8',
  function (err, data) { //callback function
    console.log(data);
  });
```

Q20. What is Callback Hell and how to avoid it?

Ans. JavaScript callback hell occurs when you are dealing with multiple nested callback functions which make the code difficult to read and debug. This problem might occur in your JavaScript code when you are dealing with nested asynchronous code in JavaScript.

Example of Nested callback function

```
async1(function () {
  async2(function () {
    async3(function () {
      ....
    });
  });
});
```

How to Avoid Callback

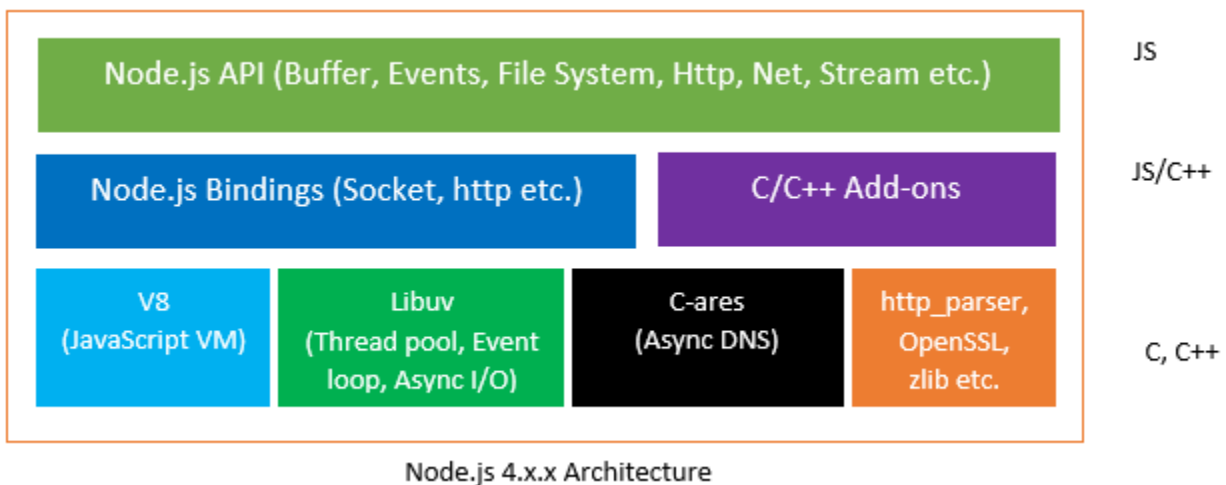
You can avoid callback hell by avoiding deeply-nested callbacks. You can also use promise object to call your async function one by one.

```
// Promise approach
var task1 = async1();
var task2 = task1.then(async2);
var task3 = task2.then(async3);
task3.catch(function () {
    // handle thrown errors from task1, task2, task3 here
});

// Promise approach with chaining
async1(function () {
    //...
}).then(async2)
  .then(async3)
  .catch(function () {
    // handle thrown errors here
  });
```

Q21. Explain Node.js Architecture?

Ans. Node.js has mainly two types of components – core components and node.js API (modules). The core components are written in C and C++, and node.js API are written in JavaScript. Diagram for Node.js architecture is given below:



Node.js API – These are written in JavaScript and directly exposed to outer world to interact with Node.js internal components.

Node.js Binding – These are Core API, which bind the JavaScript with C / C++ libraries.

C/C++ Add-ons – You can also develop your Node.js Add-ons using C/C++ to work with Node.js.

V8 – It is Google’s open source JavaScript engine, written in C++. Actually, it is a JavaScript VM which compile the JavaScript code into native machine code instead interpretation. It is the fastest JIT (Just-In-Time) compiler for JavaScript.

Libuv – It is a multi-platform support C++ library which is responsible for handling thread pool, event loop and async I/O operations in Node.js. In Node.js, blocking I/O operations are delegated to LibUV modules which has a fixed size C++ thread pool to handle these operations. When these operations are completed, they are notified to Event loop.

C-ares – It is a C library for handling async DNS request, name resolves and multiple DNS queries in parallel.

http_parser – It is a C library for parsing HTTP request and response.

OpenSSL – It is a C library for the implementation of Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. It also provides all the necessary cryptography methods like hash, cipher, decipher, sign and verify etc.

Zlib – It is a C library for data compression and decompression.

Q22. What is Module?

Ans. A module is a collection of JavaScript code which encapsulate related code into single unit of code. Node.js has a simple module loading system. A developer can load a JavaScript library or module into his program by using require method as given below:

```
var http = require('http');
```

Q23. What are core modules in Node.js?

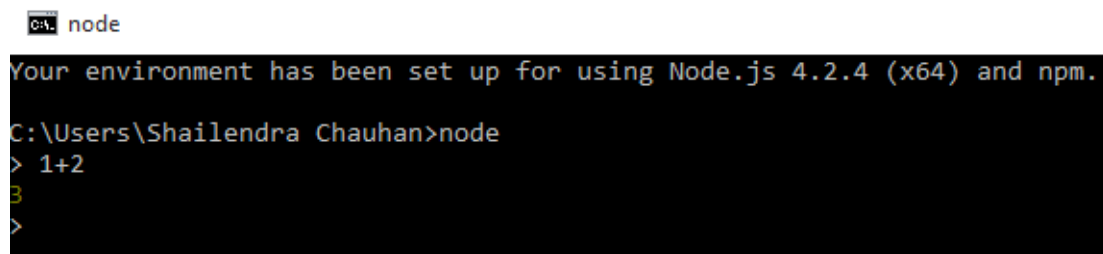
Ans. Node.js comes with many core modules which are compiled to binary when node.js is installed. There is a list of node.js core modules with links to documentation.

Module	Description
Assert	It is used by Node.js for testing itself. It can be accessed with <code>require('assert')</code> .
Buffer	It is used to perform operations on raw bytes of data which reside in memory. It can be accessed with <code>require('buffer')</code> .
Child Process	It is used by node.js for managing child processes. It can be accessed with <code>require('child_process')</code> .
Cluster	This module is used by Node.js to take advantage of multi-core systems, so that it can handle more load. It can be accessed with <code>require('cluster')</code> .
Console	It is used to write data to console. Node.js has a Console object which contains functions to write data to console. It can be accessed with <code>require('console')</code> .
Crypto	It is used to support cryptography for encryption and decryption. It can be accessed with <code>require('crypto')</code> .
Debugger	It is used for code debugging. To use this, start Node.js with the debug argument and for debugging add <code>debugger;</code> statement in your code.

DNS	It is used to perform operations like lookup and resolve on domain names. It can be accessed with <code>require('dns')</code> .
Events	It is used for events handling in node.js. In node.js, events are emitted by other node objects. It can be accessed with <code>require('events')</code> .
File System	It is used to perform operations on files. It can be accessed with <code>require('fs')</code> .
HTTP	It is used to create Http server and Http client. It can be accessed with <code>require('http')</code> .
Net	It used to create TCP server and client which can communicate over a network using TCP protocol. It can be accessed with <code>require('net')</code> .
OS	It is used to provide basic operating system related utility functions. It can be accessed with <code>require('os')</code> .
Path	It is used to perform operations on paths of files. It can be accessed with <code>require('path')</code> .
Process	It is a global object and provides information related to the program execution. You do not need to load it using <code>require()</code> method.
Query String	It is used to deal with query strings.
Stream	It is used to stream data between two entities. It can be accessed with <code>require('stream')</code> .
Timers	All of the timer functions are global and deals with time. You do not need to load it using <code>require()</code> method.
Url	It is used for URL resolution and parsing. It can be accessed with <code>require('url')</code> .
Util	It is primarily designed to support the needs of Node.js's internal APIs. It is also useful for debugging. It can be accessed with <code>require('util')</code> .
Vm	It provides an access to V8 virtual machine to compile and execute JavaScript code. It can be accessed with <code>require('vm')</code> .
Zlib	It is used to compress and decompress data. It can be accessed with <code>require('zlib')</code> .

Q24. What is REPL Terminal?

Ans. REPL stands for Read-Eval-Print-Loop. It is an interface to run your JavaScript code and see the results. You can access REPL by simply running node.js command prompt and simply run command *node*.



```

C:\Users\Shailendra Chauhan>node
> 1+2
3
>

```

Here, we are adding two numbers 1 and 2 which results into 3.

Q25. What are the useful commands of REPL Terminal?

Ans. Following is the list of useful commands while working with REPL.

Command	Description
.help	It shows REPL options and commands.
.break	It helps you to exit from multiline expression when you get stuck.
.clear	It is an alias for .break.
.exit	Exit the REPL
.load	It will help you to load JS code form a file into the REPL session.
.save	It will save all evaluated command in the REPL session.

Q26. What are the useful shortcuts of REPL Terminal?

Ans. Following is the list of useful shortcuts while working with REPL.

Command	Description
Ctl + d	It will terminate the REPL session.
Up/Down Keys	It will iterate through command history.
Tab	It will display the list of current available commands.

Q27. What are Node packages and how to install/uninstall them?

OR

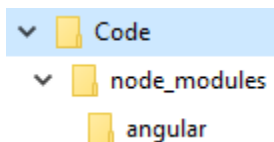
What are local packages and global packages?

Ans. Node packages are open source libraries that are added into your node project. These packages are installed in `node_modules` folder somewhere in your machine. There are two types of Node packages:

Local Packages - Local packages are installed at project level. These are installed in `node_modules` folder of the parent working directory i.e. project level folder.

For example, you can download the latest version of AngularJS into your project by running following command.

```
npm install angular
```



If you want to download latest version of AngularJS, you have to specify the version number of that package as:

```
npm install angular@1.3.5
```

You can see the list of all installed node packages within the parent working directory by using following command:

```
npm ls
```

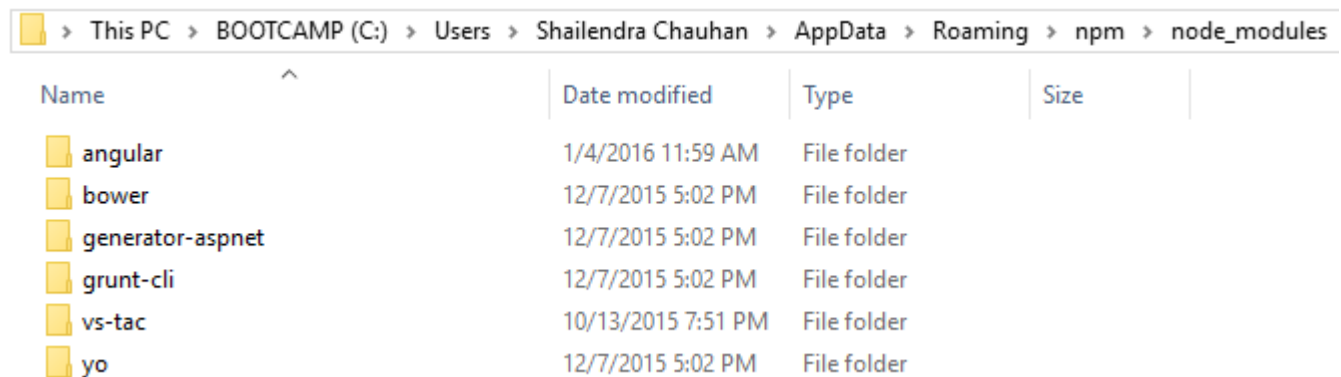
```
C:\DOT NET TRICKS\Training\Courses\NodeJS Training\Code>npm ls
C:\DOT NET TRICKS\Training\Courses\NodeJS Training\Code
└─ angular@1.4.8
```

Global Packages - Global packages are installed at global location. Typically, these are installed in `/users/username/AppData/Roaming/npm/node_modules` folder in Windows OS.

For example, you can download the latest version of AngularJS by running following command.

```
npm install angular -g
```

In my case it is installed at `C:\Users\Shailendra Chauhan\AppData\Roaming\npm\node_modules\angular` location.



This PC > BOOTCAMP (C:) > Users > Shailendra Chauhan > AppData > Roaming > npm > node_modules				
Name	Date modified	Type	Size	
angular	1/4/2016 11:59 AM	File folder		
bower	12/7/2015 5:02 PM	File folder		
generator-aspnet	12/7/2015 5:02 PM	File folder		
grunt-cli	12/7/2015 5:02 PM	File folder		
vs-tac	10/13/2015 7:51 PM	File folder		
yo	12/7/2015 5:02 PM	File folder		

You can see the list of all installed node packages within the parent working directory by using following command:

```
npm ls
```

```
C:\DOT NET TRICKS\Training\Courses\NodeJS Training\Code>npm ls -g
C:\Users\Shailendra Chauhan\AppData\Roaming\npm
└─ angular@1.4.8
```

Update Packages

A node package can be updated by using following command.

```
npm update angular
```

You can also update global packages by providing the `-g` option with commands.

```
npm update angular -g
```

Uninstalling Packages

A node package can be uninstalled or removed by using following two commands.


```
npm uninstall angular
npm rm angular
```

You can also remove global packages by providing the -g option with commands.

```
npm uninstall angular -g
npm rm angular -g
```

Q28. What are the various options to download the version of a node module?

Ans. When you create a Node project, you have to add node modules with version number as dependencies into your project *package.json* file. There are following ways to specify the version number for a node module. Let's say you want to add a version of AngularJS.

Version Number	Description
angular or angular@"" or angular@""* or angular@"latest"	Latest version
angular@"1.*" or angular@"1.x"	Any version that starts with 1. Takes the latest.
angular@1.3.5 or angular@"=1.3.5"	Choose an exact version 1.3.5
angular@>1.3.5"	Greater than version 1.3.5 Could break your code.
angular@>=1.3.5"	Greater than or equal to version 1.3.5
angular@<1.3.5"	Smaller than version 1.3.5
angular@<=1.3.5"	Smaller than or equal to version 1.3.5
angular@~1.3.5"	Greater than or equal to version 1.3.5, but less than the next major version i.e. any version compatible with 1.3.5
angular@"^1.3.5"	Any version which is compatible with 1.3.5. This will call versions up to the next major version like 2.0.0. Could break your code.
angular@"^1.3.5 - 1.4.8"	Greater than or equal to version 1.3.5 and less than or equal to version 1.4.8.

Q29. What is package.json file in node project?

Ans. Node module dependencies are defined in *package.json* of your node project. The simple structure of package.json file is given below:

```
{
  "name": "module-name",
  "version": "1.0.0",
  "description": "An example module",
  "author": "shailendra@dotnettricks.com",
  "dependencies": {
    "angular": "1.3.5"
  },
  "devDependencies": {
    "mocha": "~1.8.1"
  }
}
```

It can be created by using following node command. After running this command, you have to follow some steps which are shown in the screen shot.

```
C:\DOT NET TRICKS\Training\Courses\NodeJS Training\Code>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

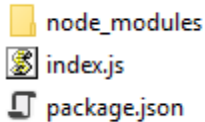
Press ^C at any time to quit.
```

```
name: (Code) my_node_app
version: (1.0.0)
description: This is first node app
entry point: (index.js)
test command:
git repository:
keywords: node, angular
author: Shailendra Chauhan
license: (ISC)
About to write to C:\DOT NET TRICKS\Training\Courses\NodeJS Training\Code\package.json:

{
  "name": "my_node_app",
  "version": "1.0.0",
  "description": "This is first node app",
  "main": "index.js",
  "dependencies": {
    "angular": "^1.4.8"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "node",
    "angular"
  ],
  "author": "Shailendra Chauhan",
  "license": "ISC"
}

Is this ok? (yes) yes
```

In this way, your package.json file has been created within your project folder.



Q30. What is the difference between Package dependencies and development dependencies?

Ans. Package dependencies and development dependencies, both are defined in package.json file.

Package Dependencies

The dependencies field of package.json file will have all packages listing on which your node project is dependent.

```
"dependencies": {  
  "angular": "1.4.8",  
  "jQuery": "^2.1.4"  
}
```

To do listing of your node module as a dependencies package you need to use either `--save` flag or `--production` flag with node command to install package.

Development Dependencies

The devDependencies field of package.json file will have those packages listing which are only required for testing and development.

```
"devDependencies": {  
  "mocha": "~1.8.1"  
}
```

To do listing of your node module as a dependencies package you need to use `--dev` flag with node command to install package.

Q31. What is the difference between module.exports and exports in Node.js?

Ans. A module is a JavaScript object having an object type property, *exports* (*module.exports*).

```
var module = { exports: {} };
```

An exports is a JavaScript variable that can be set to a value module.exports.

```
var exports = module.exports;
```

Hence, a simplified way to view a JavaScript file in Node could be like this:

```
var module = { exports: {} };  
var exports = module.exports;  
  
//TO DO:  
return module.exports;
```

In this way, if you set a property on exports (say *exports.foo = 5;*) then it will be set to *module.exports.foo* as well since in JavaScript objects are passed as references. It means *exports* and *module.exports* represent the same object.

But if you will set *exports* variable to a value other than *module.exports* then *exports* and *module.exports* will not represent the same object.

Q32. How to share a variable between Node.js modules?

Ans. In Node.js, there is no global context. Each java script code file(module) has its own context. Hence, defining a variable at the top level scope in a module will not be available in other modules.

In Node.js to share variables between multiple node modules you have to include the same module in each js file.

```
//module1.js
var module1 = {
  sayHello: function(){
    return 'Hello Node.js';
  }
};

module.exports = module1;
```

Let's access module1 into module2.

```
//module2.js
var module1 = require('./module1.js');

console.log(module1.sayHello());
```

Q33. What are global variables in Node.js?

Ans. Node.js provides you some global variables which are available by default. These are documented in the Node.js API docs: *globals* and *process*. Here is a list of global variables provided by Node.js.

Variable Name	Description
__filename	Returns the absolute path of the currently executing file
__dirname	Returns the absolute path of the directory containing the currently executing file
process	An object which represents the current running process
process.env	An object representing the user environment for the current process
process.stdin, process.stdout, process.stderr	Streams which are corresponded to the standard input, standard output and standard error output for the current process
Process.argv	An array containing the command line arguments

Q34. What are buffers?

Ans. JavaScript language has no mechanism for reading or manipulating streams of binary data. So, Node.js introduced the Buffer class to deal with binary data. In this way, Buffer is a Node.js special data type to work with binary data. A buffer length is specified in bytes. By default, buffers are returned in data events by all Stream classes. Buffers are very fast and light objects as compared to strings. A buffer act like an array of integers, but cannot be resized.

Q35. How to create buffers?

Ans. There are following methods to create buffers in Node.js.

Method	Description
<code>new Buffer(array)</code>	Allocates a new buffer using an array.
<code>new Buffer(str[, encoding])</code>	Allocates a new buffer containing the given string. encoding is optional and defaults to 'utf8'.
<code>new Buffer(size)</code>	Allocates a new buffer of size bytes.
<code>new Buffer(buffer)</code>	Copies the passed buffer data into a new Buffer instance.
<code>new Buffer(arrayBuffer)</code>	Allocates a new buffer using <i>buffer</i> property of a TypedArray or a new ArrayBuffer().

Example:

```
//creating buffer from size=2
var buf = new Buffer(2);
buf[0] = 0;
buf[1] = 1;

console.log(buf); //<Buffer 00 01>
console.log(buf.length); //2

//creating buffer from array
var arr = [1, 2, 3];
var buff = new Buffer(arr);
console.log(buff) //<Buffer 01 02 03>

//creating buffer from string
var buffer = new Buffer('Hello World!');
console.log(buffer); //<Buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64 21>
console.log(buffer.length); //12
```

Q36. How to decode buffers?

Ans. Conversion between Buffers and JavaScript string objects requires an explicit encoding method like ascii, utf8, base64, binary, hex etc.

```
//creating buffer from string
var buffer = new Buffer('Hello World!');

//by default encoding is utf8
```

```
console.log(buffer.toString()); //Hello World!
console.log(buffer.toString('ascii')); //Hello World!
console.log(buffer.toString('binary')); //Hello World!
console.log(buffer.toString('hex')); //48656c6c6f20576f726c6421
console.log(buffer.toString('base64')); //SGVsbG8gV29ybGQh
```

Q37. What are Streams?

Ans. Typically, Stream is a mechanism for transferring data between two points. Node.js provides you streams to read data from the source or to write data to the destination. In Node.js, Streams can be readable, writable, or both and all streams are instances of EventEmitter class.

```
var http = require('http');

var server = http.createServer(function (req, res) {
  // here, req is a readable stream
  // here, res is a writable stream
});
```

Q38. What are the different types of Streams?

Ans. Node.js supports four types of streams as given below:

- **Readable** - Used for read operation.
- **Writable** - Used for write operation.
- **Duplex** - Used for both read and write operations. Both operations are independent and each have separate internal buffer.
- **Transform** - A type of duplex stream where the output is computed based on input. Both operations are linked via some transform.

Q39. What are Events and Event Emitters?

Ans. Node.js core API are based on asynchronous event-driven architecture. Node.js uses *events* module for event handling. The events module exposes *EventEmitter* class to deal with events.

In Node, objects that generate events are called event emitters. All objects that emit events are instances of the EventEmitter class. Event Emitters are based on publish/subscribe pattern. EventEmitter class provides Listeners for listing events and Emitters for emitting events.

An example which creates an event emitter is given below:

```
var events = require("events");
var emitter = new events.EventEmitter();

//listening 'docall' event
emitter.on("docall", function () {
  console.log("ring ring ring")
});

//emitting 'docall' event
emitter.emit("docall");
```


Q40. What events are supported by readable streams?

Ans. A readable stream supports following five events.

- **data** - This is event is emitted when the data is available to read in flowing mode. In flowing mode, you can pause and resume streams.
- **readable** - This is event is emitted when the data is available to read in non-flowing mode.
- **end** - This is event is emitted when there is no more data to read.
- **close** - This is event is emitted when the stream and any of its underlying resources (like a file descriptor) have been closed. This event also indicates that no more events will be emitted.
- **error** - This is event is emitted when there is any error while receiving or writing data.

Q41. Give an example of readable stream with non-flowing mode?

Ans. The *readable* event is used to read data in non-flowing mode. An example is given below:

```
var fs = require("fs");
var stream = fs.createReadStream('./input.txt');

//non-flowing mode: stream pause is not possible
stream.on('readable', function(){
    // read chunk by chunk
    var chunk;
    while ((chunk = stream.read()) !== null) {
        console.log(chunk);
    }
});

stream.on('end', function () {
    console.log('finished'); //no more data to read
});
stream.on('close', function () {
    console.log('closed'); //file has been closed
});
stream.on('error', function (err) {
    console.log(err.stack);
});
```

Q42. Give an example of readable stream with flowing mode?

Ans. The *data* event is used to read data in flowing mode. An example is given below:

```
var fs = require("fs");
var stream = fs.createReadStream('./input.txt');

//flowing mode: stream pause is possible
stream.on('data', function (chunk) {

    console.log(chunk); //reading chunk
    //pausing stream
    stream.pause();
});
```

```

    console.log('wait for 1 second.....');
    setTimeout(function () {
        console.log('data start flowing again.....');
        stream.resume();
    }, 1000);
});

stream.on('end', function () {
    console.log('finished'); //no more data to read
});
stream.on('close', function () {
    console.log('closed'); //file has been closed
});
stream.on('error', function (err) {
    console.log(err.stack);
});

```

Q43. What events are supported by writable streams?

Ans. A writable stream supports following five events.

- **drain** - This is event is emitted when
- **pipe** - This is event is emitted when.
- **unpipe** - This is event is emitted when.
- **finish** - This is event is emitted when.
- **error** - This is event is emitted when there is any error while receiving or writing data.

Q44. Give an example of writable stream?

Ans. An example of writable stream is given below:

```

var fs = require("fs");
var data = 'This data is from writable stream!';

// create a writable stream
var ws = fs.createWriteStream('output.txt');

// write the data to stream with encoding to be utf8
ws.write(data, 'utf8');

// mark the end of file:no more data to read
ws.end();

ws.on('finish', function() {
    console.log("write completed.");
});
ws.on('error', function(err){
    console.log(err.stack);
});

```

Q45. What is Streams piping?

Ans. Streams Piping is a way to pass output of one stream as an input to another stream. Node.js provides `pipe()` function for this purpose.

The `pipe()` function attaches a readable stream to a writable stream for passing the data from one stream to another stream. Let's understand it with an example, here we are reading the data from *input.txt* and writing it to another file *output.txt*.

```
var fs = require('fs');

var rs = fs.createReadStream('input.txt');
var ws = fs.createWriteStream('output.txt');

rs.pipe(ws); //streams piping

//Handling events: end and error
rs.on('end', function () {
    console.log("copy has been completed.");
    ws.end(); //mark end
});
rs.on('error', function (err) {
    console.log(err.stack);
});
```

Q46. How to debug the code in Node.js?

Ans. The static languages like C# and Java have tools like Visual Studio and Eclipse respectively for debugging. Node.js is based on JavaScript and in order to debug your JavaScript code we have `console.log()` and `alert()` statements. But Node.js supports other options as well for code debugging. It supports following options:

1. **The Built-In Debugger** – A non GUI tool to debug the Node.js code.
2. **Node Inspector** - A GUI tool to debug the Node.js code with the help of chrome or opera browser.
3. **IDE Debuggers** – IDE like as WebStorm, Visual Studio Code and eclipse enide etc., support the Node.js code debugging environment.

Q47. How to debug the code using Node.js Built-In debugger?

Ans. It is a non GUI tool to debug the Node.js code. It is used by inserting debugger statements into the code and run your app in debug mode using command: *node debug app.js*

```
//app.js
for (var i = 0; i < 10; i++) {
    debugger;
    console.log(i);
}
console.log("done debugging!");
```

The main node debug commands are as follows:

- **continue** – `cont`, `c` (continue until the next debugger/break point)

- **step** – next, n (step to the next statement)
- **step in** – step, s (step inside the function call)
- **step out** – out, o (step outside the function call)

Using above listed command you can debug your code easily.

Q48. How to debug the Node.js code using Node Inspector?

Ans. It is a GUI tool to debug the Node.js code with the help of chrome or opera browser. It is based on the Blink Developer Tools (formerly WebKit Web Inspector). It only works in Chrome and Opera browser.

In order to use Node Inspector for debugging, you need to install it globally into your machine using NPM

```
C:\>npm install -g node-inspector
```

Once it gets installed, run your Node app by running following command using terminal:

```
C:\Node_debugging>node-debug app.js
```

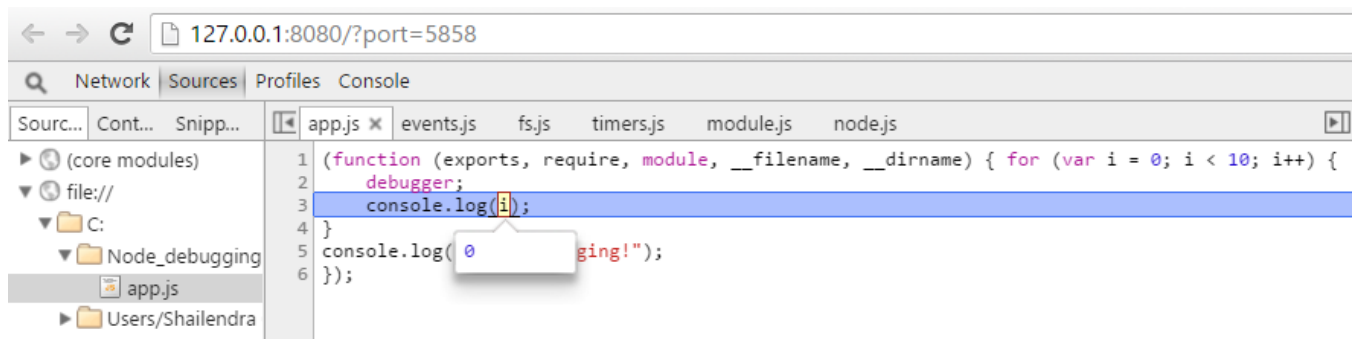
After running above command, you will see the output as given below:

```
Node Inspector v0.12.8
Visit http://127.0.0.1:8080/?port=5858 to start debugging.
Debugging `app.js`

Debugger listening on port 5858
```

It will open the Chrome (assuming chrome is your default browser) immediately and will show your app code in debug mode. If it doesn't not open the browser, simply open your chrome browser and navigate to the address from the console output (<http://127.0.0.1:8080/?port=5858>).

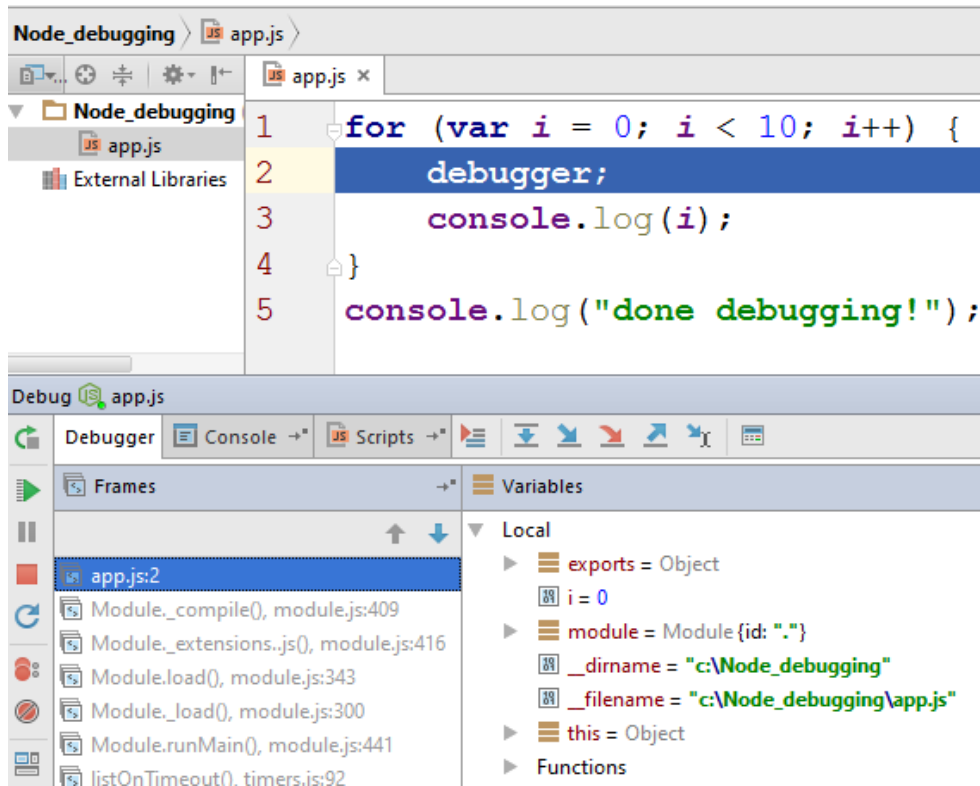
Now just enjoy the nice visual debug environment for debugging your Node.js code as given below.



Q49. How to debug the Node.js code using IDE debuggers?

Ans. IDE like as WebStorm, Visual Studio Code and eclipse enide etc., support the Node.js code debugging environment. In order to debug your code in WebStorm and Visual Studio code, you have to set up your Node.js project. After that run it and debug it as given below:

WebStorm IDE Debugging:



Visual Studio Code IDE Debugging:



Q50. How to handle errors in Node.js?

Ans. Error handling is a most important part of building robust Node.js app. If you ignore errors or not dealing with errors in proper way, your node.js app might crash or be left in an inconsistent state. You can divide the errors mainly into two categories: **developer errors** and **operational errors**.

Developer errors are bugs in the program like typing errors in variables or functions name, passing wrong data, syntax errors etc.

Operational errors are the issues which typically occur on run time like request timeout, failed to connect to server, invalid user input or database errors etc.

There are following basic patterns you can use for errors handling in Node.js:

- **Try/Catch** - The most commonly used way to handle errors in any programming language is try/catch blocks. Also, try/catch block executes in synchronous fashion; hence suitable for compiled languages like C#, Java etc.

```
try {  
    throw new Error('throw error');  
} catch (e) {  
    console.log(e);  
}
```

By default, Node.js supports asynchronous operations. It is difficult to write synchronous operations in Node.js. So, if you will use try/catch with Node.js asynchronous code, at the time of throwing the exception it will not be in the stack. This will generate an uncaught exception. Hence, you should use try/catch block with Node.js to handle errors for synchronous code.

- **Callback** – This way is used to handle errors in any callback without using nested try catch statements. In this way you have to pass a callback, *function(err, result)* which will be invoked when the asynchronous operation completes. Here *err* parameter returns the error and *result* is the result of completed operations, depending on whether the operation succeeded or failed.

```
//request in express  
app.get('/', function (req, res) {  
    db.query('sql_query', function (err, result) {  
        if (err) {  
            res.writeHead(500);  
            res.end();  
            console.log(err);  
        }  
        else {  
            res.send(result);  
        }  
    });  
});
```


- **Event Emitters** – In Node.js you can use Event Emitters to emit errors. Event emitters come with native support for *error event*. So, instead of throwing the errors emit it when the errors happen and the caller will listen for *error event* on the emitter.

This way is useful when your single operation might create multiple errors or multiple results and when a lot of different asynchronous operations are happening.

```
var events=require('events');
var emitter = new events.EventEmitter();

function async() {
    process.nextTick(function () {
        // This emits the "error" event
        emitter.emit("error", new Error("Something went wrong"));
    });
    return emitter;
}
//calling async
var event = async();
event.on("error", function (error) {
    // When the "error" event is emitted, this is called
    console.error(error);
});
```

Q51. What are uses of path module in Node.js?

Ans. Node.js provides path module to normalize, join, and resolve file system paths. It also used to find relative paths, extract components from paths and determine the existence of paths.

Note - The path module simply manipulates strings and does not interact with the file system to validate the paths.

Q52. What functions are provided by path module in Node.js?

Ans. Node.js path module provides following functions to manipulate path strings:

- **path.normalize()** - Ensure a predictable format when a source is untrusted or unreliable.
- **path.join()** - Build a single path form the path segments.
- **path.dirname()** - Provide the directory name out from a path.
- **path.basename()** - Provide the final path segment.
- **path.extname()** - Use to slice from the last period(.) to the end of the path string.
- **path.resolve()** - Resolve a list of path instructions into an absolute path.

Q53. What is File System module in Node.js?

Ans. Node.js provides file system module (fs) to perform file I/O and directory I/O operations. The fs module provides both asynchronous or synchronous way to perform file I/O and directory I/O operations.

The synchronous functions have “Sync” word as suffix with their names and return the value directly. In synchronous file I/O operation, Node doesn’t execute any other code while the I/O is being performed.

By default, fs module functions are asynchronous, it means they return the output of the I/O operation as a parameter to a callback function.

Q54. What functions are provided by fs module to perform I/O operations on a file?

Ans. Node.js fs module provides following functions to perform file I/O operations:

- **fs.open(path, flags, [mode], callback)** - Open a file at given path and callback will receive any exceptions as first argument and a file descriptor as second argument.
- **fs.close(fd, callback)** - Close a file descriptor. The callback receives one argument as an exception.

```
var fs = require("fs");
var path = "./fs.txt";

//open file at given path to read ('r'- read mode)
fs.open(path, 'r', function (err, fd) {
  fs.fstat(fd, function (err, stats) {
    var totalBytes = stats.size;
    var bytesRead = 0;
    var buffer = new Buffer(totalBytes);

    //Each call to ensure that chunk size is not too small; not too large
    var chunkSize = Math.min(512, totalBytes);
    while (bytesRead < totalBytes) {
      //for reading last chunk
      if ((bytesRead + chunkSize) > totalBytes) {
        chunkSize = totalBytes - bytesRead;
      }
      //reading the file byte by byte
      fs.read(fd, buffer, bytesRead, chunkSize, bytesRead, error);
      bytesRead += chunkSize;
    }
    function error(err) {
      if (err) {
        console.log("File read aborted!!");
        console.log(err);
        fs.close(fd, function () {
          console.log("File closed!!");
        });
      }
    }
    fs.close(fd, function () {
      console.log("File closed!!");
    });
    console.log("File read completed!!");
    console.log("Total bytes read: " + totalBytes);
    //display on screen
    console.log(buffer.toString());
  });
});
```

- **fs.rename(oldName, newName, callback)** - The callback receives one argument as an exception.
- **fs.unlink(path, callback)** - Delete the file at given path. The callback receives one argument as an exception.
- **fs.truncate(path, len, callback)** – Truncate the file at given path by len bytes. If len is greater, the file length is padded by appending null bytes (\x00) until len is reached.
- **fs.chown(path, uid, gid, callback)** - Changes the ownership of the file at path . Use this to set whether a user uid or group gid has access to a file.
- **fs.chmod(path, mode, callback)** - Change the mode (permissions) on a file at path.
- **fs.stat(path, callback)** - Use to read the attributes of a file.

```
var fs = require("fs");
var path = "./fs.txt";
//read the attributes of a file
fs.stat(path, function(err, stats) {
    console.log(stats);
});

//appended
fs.appendFile('./fs-write.txt', '\nAppended Data', function(err) {
    if (err) throw err;
    console.log('Append completed!');
});

//renamed
fs.rename('./fs-write.txt', './fs-write.bak', function (err) {
    if (err) throw err;
    console.log('Rename completed!');
});

//moving file
fs.rename('./fs-write.txt', './content/fs-write.txt', function (err) {
    if (err) throw err;
    console.log('Move completed!');
});

//deleted
fs.unlink('./fs-write.bak', function (err) {
    if (err) throw err;
    console.log('Deleted successfully!');
});

// Change permission to readonly
fs.chmod('./fs-write.txt', '444', function (err) {
    if(err)throw err;
    console.log("changed permission");
});
```

- **fs.read(fd, buffer, offset, length, position, callback)** – Read a file byte by byte.
- **fs.readFile(path, [options], callback)** - Fetch the entire file at once at given path.

- **fs.write(fd, buffer, offset, length, position, callback)** - Write a file byte by byte.
- **fs.writeFile(path, data, [options], callback)** - Write a large chunks of data.
- **fs.appendFile(path, data, [options], callback)** - Similar to fs.writeFile, except that data is appended to the end of the file.
- **fs.createWriteStream(path, [options])** - Return a writable stream object for file at path.
- **fs.createReadStream(path, [options])** - Return a readable stream object for file at path . You can perform stream operations on the returned object.

```
var fs = require("fs");

//Fetching an entire file at once
fs.readFile("./text.txt", "utf8", function(err, data) {
    if(err) {
        console.write(err);
    }
    else{
        console.log(data);
    }
});

//open file and write to it byte by byte
fs.open("fs-write.txt", "w", function (err, fd) {
    var buffer = new Buffer('This is a File System write operation');
    var totalBytes = buffer.length;
    var bytesWrite = 0;

    //Each call to ensure that chunk size is not too small; not too large
    var chunkSize = Math.min(15, totalBytes);
    while (bytesWrite < totalBytes) {
        //for writing last chunk
        if ((bytesWrite + chunkSize) > totalBytes) {
            chunkSize = Math.min(512, (totalBytes - bytesWrite));
        }
        //writing the file byte by byte
        fs.write(fd, buffer, bytesWrite, chunkSize, bytesWrite, error);
        bytesWrite += chunkSize;
    }
    function error(err) {
        if (err) {
            console.log("File write aborted!!");
            console.log(err.stack);
            fs.close(fd);
        }
    }
    fs.close(fd);
    console.log("File write completed!!");
    console.log("Total bytes written: " + totalBytes);
});
```

```
//write to file at once
fs.writeFile("fs-write2.txt","fs write operation","utf8", function (err) {
    if(err) {
        return console.log(err);
    }
    else {
        console.log("File has been written");
    }
});
```

Q55. What functions are provided by fs module to perform I/O operations on a directory?

Ans. Node.js fs module provides following functions to perform directory I/O operations:

- **fs.readdir(path, callback)** - Read a directory.
- **fs.mkdir(path[, mode], callback)** - Make a directory.
- **fs.rmdir(path, [callback])** - Remove a directory.

```
var fs = require("fs");

//reading directory attributes
var path = 'C:\Downloads';
fs.readdir(path, function (err, files) {
    if (err) {
        console.log(err);
    }
    else {
        files.forEach(function (file) {
            var filepath = path + "/" + file;
            fs.stat(filepath, function (err, stats) {
                console.log(filepath);
                console.log(stats);
            });
        });
    }
});

// make a new directory
var path = 'C:/test'; //new dir_name
fs.mkdir(path, 0666, function (err) {
    if (err) {
        console.log(err.stack);
    } else {
        console.log('Created test dir');
    }
});

var path2 = 'C:/mytest'; //existing dir_name
//remove directory at given path
```

```
fs.rmdir(path2, function (err) {
  if (err) {
    console.log(err.stack);
  } else {
    console.log('Removed test dir');
  }
});
```

Q56. How to check a file/directory existence in node.js?

Ans. In Node.js 4.x or higher, fs.stat() function is recommended to find out the existence of a file/directory. Both path.exists() and fs.exists() functions have been deprecated from Node.js v0.12.x

```
var fs= require("fs");
fs.stat('./fs-write.txt', function(err, stats) {
  if (err) {
    if (err.code == 'ENOENT') {
      console.log('Does not exist.');
```

Q57. Which types of network application you can build using node.js?

Ans. Node.js is best for developing HTTP based application. But it is not only for developing HTTP based application. It can be used to develop other types of applications. Like as:

- TCP server
- Command-line program
- Real-time web application
- Email server
- File server
- Proxy server etc.

Q58. How to create TCP Server and TCP Client?

Ans. Node.js "net" module is used for TCP network communication. "net" module is used to create both server and client. net.createServer() method is used to create a TCP server. net.connect() and net.createConnection() methods are used to create TCP client.

```
var net = require('net');
//creating TCP Server
var server = net.createServer(function (socket) {
  console.log('client connected');
```

```

socket.on('data', function (data) {
    console.log('client data:', data.toString());
});

socket.on('end', function () {
    console.log('client disconnected');
});

//shutting down the Server
socket.on('close', function () {
    console.log('server closed');
});

socket.on('error', function (err) {
    console.log('Server error:', err);
});
socket.write('Hello World!\r\n');
});

server.listen(8081, function () {
    console.log('server is listening at 8081');
});

```

```

var net = require('net');

//creating TCP client
var client = net.connect({port: 8081}, function () {
    console.log('connected to server!');
});

client.on('connect', function () {
    // writing your "request"
    client.write('Hello from client1');
});

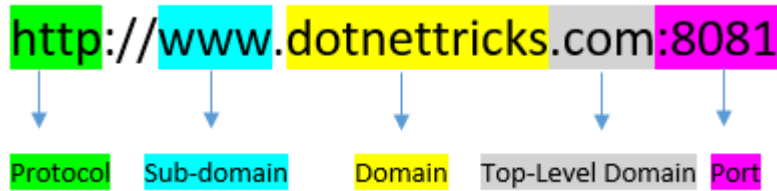
client.on('data', function (data) {
    console.log(data.toString());
    client.end();
});

client.on('end', function () {
    console.log('disconnected from server');
});

```

Q59. What is socket?

Ans. When two applications communicate over a network, they use sockets. A socket is a combination of an Internet Protocol (IP) address and a port number. An IP address is used to uniquely identify a device on a network, so that communication can happen.



Q60. What are WebSockets?

Ans. WebSockets are based on the WebSocket protocol which is introduced with Html5. WebSockets allow two-way communication with a persistent connection over TCP. WebSockets communication looks like HTTP based communication, but the HTTP part is just to create a handshake between the client and the server. Once the handshake is successful, both the client and the server can send messages to each other.

WebSockets communication is message-based, which makes it easier than TCP sockets based communication which is based on streams. WebSockets can be used for communication between browser and server, as well as Peer-to-Peer (P2P), or direct communication between browsers. WebSocket-Node and Socket.IO are third party node.js modules for creating a WebSocket server.

Q61. How to create an Http Server using Node.js?

Ans. Node.js is best for developing HTTP based application. http module is used to create an http server.

```
var http=require("http");

http.createServer(function(req,res){
    res.writeHead(200,{"Content-Type":"text/html"});
    res.write("<h1>Hello, Node.js Http Server!</h1>");
    res.end(); //to end response
}).listen(8081);

console.log("Server is running at http://localhost:8081");
```

Q62. What are Node.js Http module limitations?

Ans. Node.js http module has following limitations:

- No cookies handling or parsing.
- No built-in session support.
- No built-in routing support.
- No static file serving.

Q63. What is SSL/TLS?

Ans. SSL and TLS are cryptographic protocols for data encryption & authentication between client and servers. Secure Sockets Layer(SSL) was originally developed by Netscape and released as SSL 2.0 in 1995 and SSL 3.0 was released in 1996. Transport Layer Security (TLS) is the successor to SSL. TLS 1.0 was introduced in January 1999.

TLS 1.0 also referred to as "SSL 3.1". TLS 1.0 is more secure than SSL v3.0. Current browsers support TLS 1.0 by default. SSL/TLS are used to secure application-specific protocols such as Http, Ftp, Smtip etc.

Q64. How to create an Https Server using Node.js?

Ans. HTTP is a stateless and text-based protocol build on the top of TCP protocol. Https or HTTP Secure is an encrypted version of HTTP and commonly used when dealing with sensitive data.

For running https server, you need to purchase a SSL certificate from a certificate authority or you have to generate one yourself.

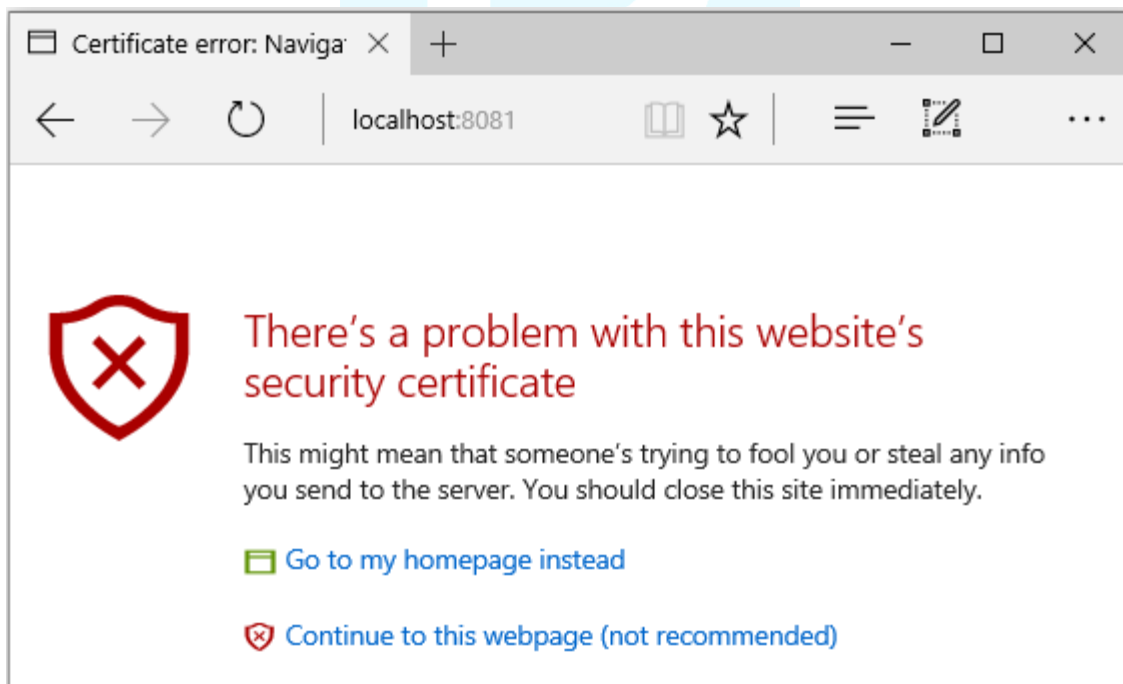
```
var fs = require('fs');
var https = require('https');

// read the private key and certificate at given path
var options = {
  key : fs.readFileSync('./server.key'),
  cert : fs.readFileSync('./server.crt')
};

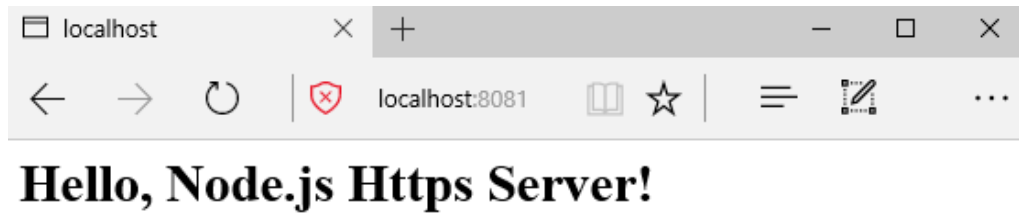
https.createServer(options, function (req, res) {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.write("<h1>Hello, Node.js Https Server!</h1>");
  res.end(); //to end response
}).listen(8081);

console.log("Server is running at : https://localhost:8081");
```

Now run the server and when you will open it in your browser you will get the following error since it is a self-signed certificate.



Just ignore this error and click on option continue to this webpage for checking your https server status. You will find the message in browser as shown below:



Q65. How to create a self-signed certificate in Node.js?

Ans. Here are the steps to generate a self-signed certificate in node.js on windows os:

1. First you have to download and configure OpenSSL into your system. Download it from the given url <https://slproweb.com/products/Win32OpenSSL.html>. In my windows pc it is installed at C:\OpenSSL-Win64 location.
2. Now open command prompt and follow the given steps to generate self-signed certificate and key

```
C:\>cd C:\OpenSSL-Win64\bin
C:\OpenSSL-Win64\bin>openssl.exe

OpenSSL> genrsa -des3 -out server.enc.key 1024

OpenSSL> req -new -key server.enc.key -out server.csr

OpenSSL> rsa -in server.enc.key -out server.key

OpenSSL> x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
```

Note - The self-generated certificates generally trigger warnings in the browser.

Q66. What is socket.io?

Ans. Socket.io is the most popular node.js module for WebSockets programming. It is used for two-way communication in web. It uses events for transmitting and receiving messages between client and server.



socket.emit("eventname",data) event is used for sending messages. socket.on("eventname",callback) event is used for receiving messages.

Q67. How to create a simple chat app using socket.io?

Ans. To create a simple chat app, you need to create a http server using socket.io node module then you need to create a client using socket.io JavaScript library.

Chat Server

Here is an example to create an http server using Express and Socket.IO. Make sure you have installed express and socket.io modules using npm.



```
var express= require('express');
var http = require('http');
var socket_io = require('socket.io');

var app = express();
var server= http.Server(app);
var io = socket_io(server);

io.on('connection', function(socket){
  socket.on('server_msg', function(msg){
    io.emit('client_msg', msg);
  });
});

//setting public folder to server static files - html, css, js and images
app.use(express.static(__dirname + '/public'));

//serve html file
app.get('/', function(req, res){
  res.sendFile(__dirname + '/views/index.html');
});

//running server
server.listen(3000, function(){
  console.log('Server is running on http://localhost:3000');
});
```

Client

Here is an example to create client to interact with server. Make sure you have downloaded socket.io JavaScript library from <http://socket.io> website.

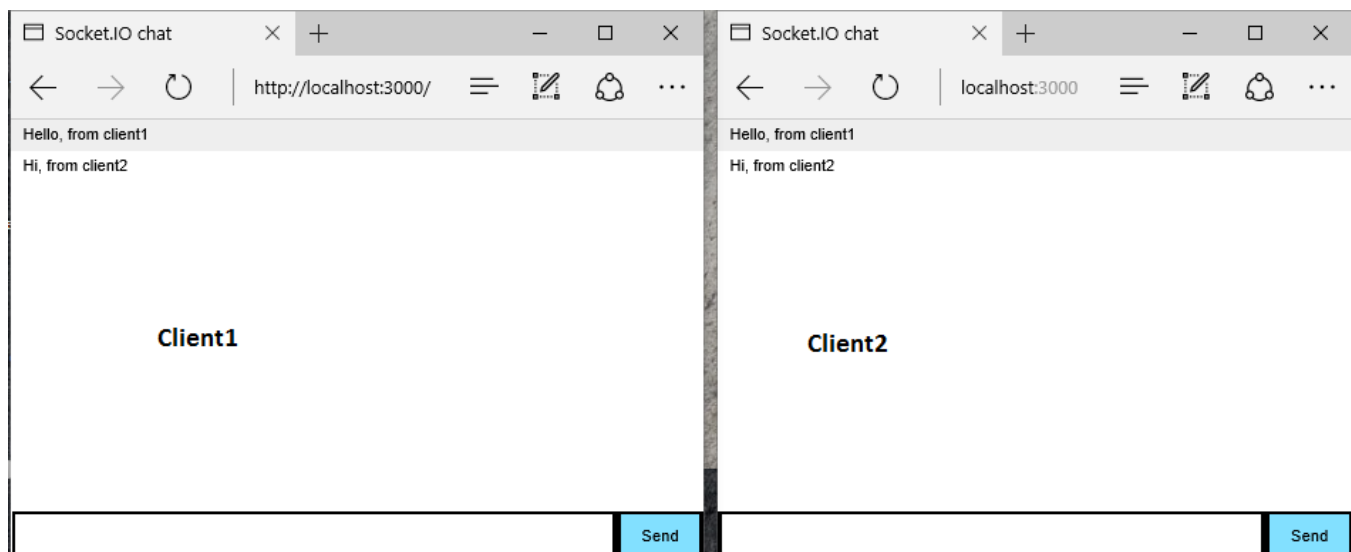
```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>Socket.IO chat</title>
  <link href="css/site.css" rel="stylesheet"/>
  <script src="js/socket.io-1.4.5.js"></script>
  <script src="js/jquery-1.11.1.js"></script>
</head>
<body>
<ul id="messages">
</ul>
<form action="">
  <input id="txt_chat" autocomplete="off"/>
  <button>Send</button>
</form>
<script type="application/javascript">
  //put this code after form tag
  var socket = io();
  $('form').submit(function () {
    var send_msg = $('#txt_chat').val();
    socket.emit('server_msg', send_msg);
    $('#txt_chat').val('');
    return false;
  });
  socket.on('client_msg', function (msg) {
    var receive_msg = $('<li>').text(msg);
    $('#messages').append(receive_msg);
  });
</script>
</body>
</html>

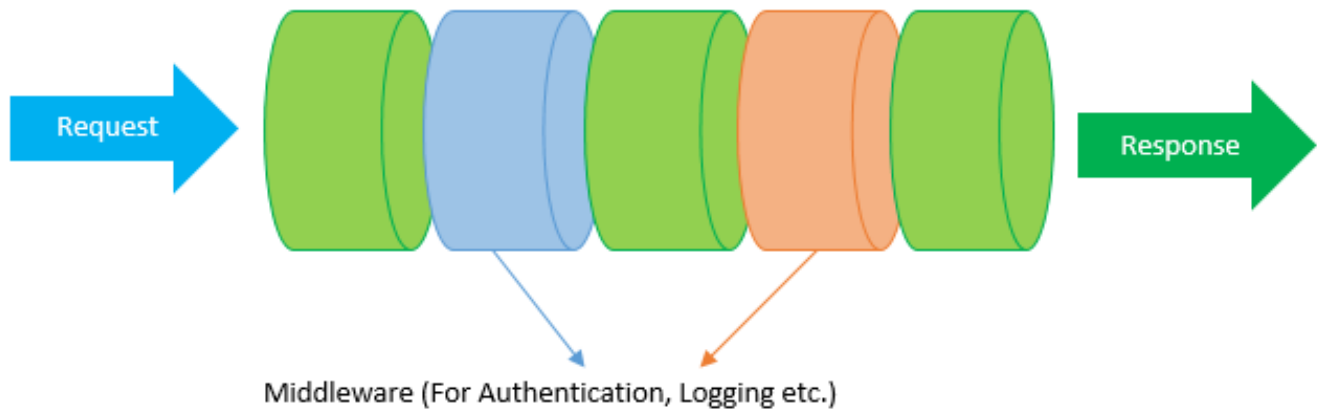
```

Now run the node.js server and open the given url <http://localhost:3000> in two browser windows. These two browsers client can communicate to each other as shown below.



Q68. What is middleware?

Ans. A middleware is a chain of functions which are executed before the request is being processed. It can be used for various operations like user authentication, logging etc.



Q69. What is connect?

Ans. Connect is an extensible HTTP server framework built on the top of node's http. It offers higher level APIs for common HTTP server functionality like session management, authentication, logging and more.

You can think connect as a stack of middlewares. You can create a server using connect as given below:

```
var connect = require("connect");
var app = connect();

app.use("/", function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end("This is connect middleware");
});

var port = process.env.port || 1301;
app.listen(port);

console.log("Server is running at http://localhost:" + port);
```

Q70. What are various node.js web development frameworks?

Ans. The best and most powerful node.js web development frameworks to build real time and scalable web applications with ease are given below:

MVC frameworks

- Express
- Koa
- Hapi
- Sails
- Nodal

Full-stack frameworks

- Meteor
- Derby.js
- MEAN.IO
- MEAN.js
- Keystone
- Horizon

Q71. What are various node.js REST API frameworks?

Ans. The best and most powerful node.js REST API frameworks to build a fast Node.js REST API server with ease are given below:

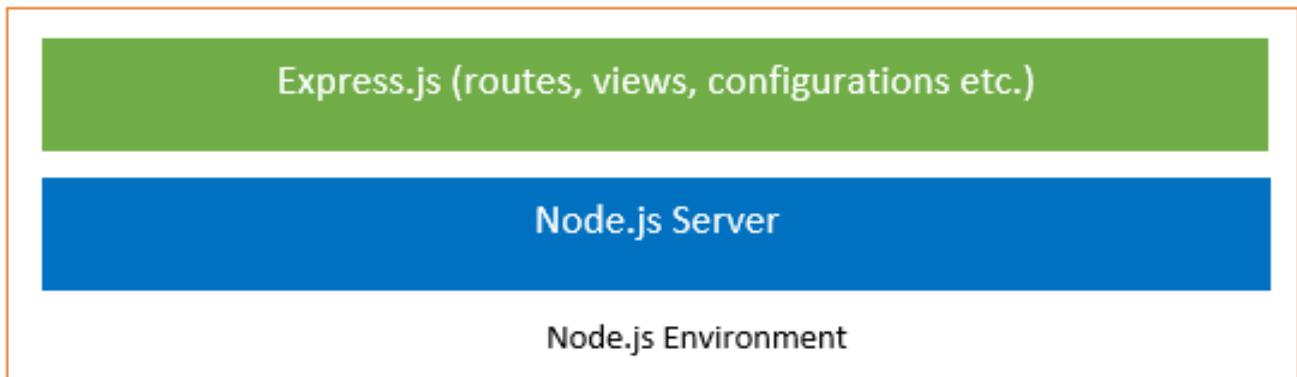
- Restify
- LoopBack
- ActionHero
- Fortune.js

Q72. What is Express?

Ans. Express is a web development framework for Node.js. Express is used to build single (SPA), multipage (MPA) and hybrid web applications and REST API using Node.js.



Express is built on the top of http and connect. Express is based on Ruby's Sinatra framework.



Q73. Why to use Express?

Ans. You should use Express because provides following features:

- Parses the arguments and headers
- Supports Routing
- Supports multiple view engines like Jade, EJS, JSHtml etc.
- Handle Configurations
- Supports Sessions
- Supports Content Negotiation
- Supports Error Handling
- Supports Multiple Databases
 - RDBMS – MySQL, MS SQL etc.
 - NoSQL – MongoDB, Firebase, Redis etc.

Q74. How to create an Http server using Express?

Ans. Express is best for developing web application using Node.js. You can create an http server using express as given below:

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

app.get("/", function (req, res) {
  res.write("Hello, Express");
  res.end();
});
var port = process.env.port || 1305;
app.listen(port);
console.log("Server is running at http://localhost:" + port);
```

Q75. How to define routes using Express?

Ans. Express provides an efficient way to manage and handle routes in a web application. Express provides methods like get(), post(), put(), delete(), and so on to deal with HTTP verbs – get, post, put and delete respectively.

All of these methods take the first parameter as representing URL and a sequence of middleware as arguments. Let's see the examples to create routes in express.

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

//route for home page
app.get("/", function (req, res) {
  res.write("This is Home Page");
  res.end();
});

//route for about us page
app.get("/about", function (req, res) {
```

```
    res.send("<h1>This is About Us Page</h1>");
});

var port = process.env.port || 1305;
app.listen(port);

console.log("Server is running at http://localhost:" + port);
```

Q76. What is the use of Express Router Class?

Ans. Express router class was introduced with Express 4.0. This provides routing APIs like. use, .get, .param, and route to deal with routes in a web application. Following is an example to define routes using router class.

```
var express = require("express");

// get an instance of router
var router = express.Router();
var app = express();

// home page route
router.get("/", function (req, res) {
    res.send("This is a Home Page");
});

// about page route
router.get("/about", function (req, res) {
    res.send("This is a About Us Page");
});

// apply the routes to our application
app.use('/', router);

//running server
var port = process.env.port || 1306;
app.listen(port);

console.log("Server is running at http://localhost:" + port);
```

Q77. What is template engine?

Ans. A template engine allows us to create and render an HTML template with minimal code. At runtime, a template engine executes expressions and replaces variables with their actual values in a template file. In this way, it transforms the template into an HTML file and sent to it the client.

Q78. What template engines you can use with express?

Ans. The popular template engines which you can use with Express are Pug, Handlebars, Mustache, and EJS. The Express application generator uses Pug as its default template engine.

Note - Jade is now known as **pug**.

Q79. What are the advantages of template engines?

Ans. There are following advantages of using template engines:

- Separate program-logic and UI.
- Makes the development easier.
- Improves flexibility.
- Code reusability.
- Easy to modify and maintain.

Q80. What is Koa?

Ans. Koa is a node.js web framework developed by the team of Express. Koa is smaller, more expressive, and more robust framework from Express. Koa uses generators via co to efficiently deal with call backs and increase error-handling capabilities.

koa

next generation web framework for node.js

Koa does not use middleware. It provides a set of methods that make writing servers fast and enjoyable.

Q81. How to create an Http server using Koa?

Ans. You can create an http server using Koa as given below:

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

app.use(function *(){
  this; // context
  this.request; // koa Request
  this.response; // koa Response
  this.body = '<h2>Koa Server</h2>';
});

var port = process.env.port || 1305;
app.listen(port);

console.log("Server is running at http://localhost:" + port);
```

Q82. What is Unit Testing?

Ans. Unit testing is a way to test each piece of your code which is called as unit. The idea behind the unit testing is to ensure the quality of each smaller unit. It verifies the functional behavior of each unit in response to correct and incorrect cases of input data and verify any assumptions made by the code. Unit test cases are usually written by the developer who has written that code unit.



Q83. Why to do Unit Testing?

Ans. Unit tests reduce number of bugs, provide accurate documentation, and improve design. There are following reasons to choose unit testing.

- Allow you to make big changes into code quickly since you know it's tests are working as expected and when you make the changes you need to write and run tests again to get it work.
- Reduce the number of bugs in production code.
- Help you to understand the design of the code since to write tests you need to outline all the conditions with respect to your code and what outputs you are expecting from the code.
- Allow refactoring of code at any time without fear of breaking existing code. In this way it makes the coding process more agile.
- Ensure functional behavior of your code since all the tests have been passed.

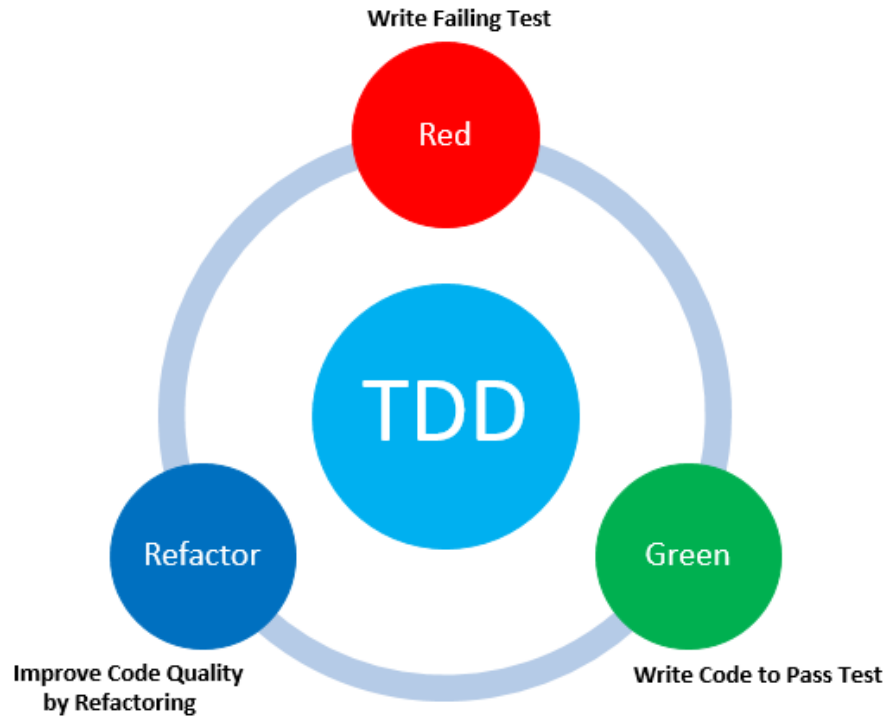
Q84. What are node.js unit testing frameworks?

Ans. Node.js supports various testing libraries and frameworks to follow test-driven development process. The most popular node.js testing libraries and frameworks are given below:

- Node.js assert
- Mocha
- Chai.js
- Should.js
- Nodeunit
- Vows.js
- Jasmine etc.

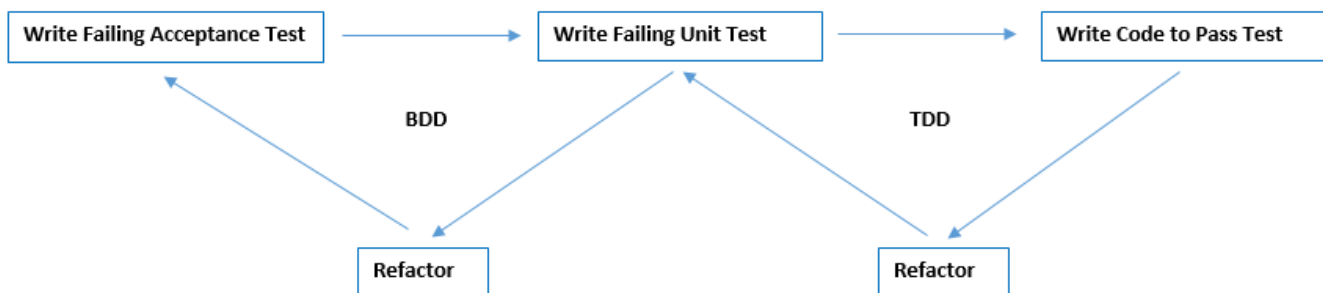
Q85. What is Test Driven Development (TDD)?

Ans. TDD is a software development technique to develop software using unit tests. In this approach, test cases are written before any code. And to make test cases pass, the code is written and then refactor the code to make it maintainable.



Q86. What is Behavior Driven Development (BDD)?

Ans. BDD is an extension/revision of TDD. BDD does focus on the business behaviors rather than implementation aspect. BDD is an Agile software development technique that encourages collaboration between developers, QA and non-technical participants in a software development. In simple language, it is like a story writing.



Q87. What is Acceptance Test Driven Development (ATDD)?

Ans. ATDD is a collaborative practice where whole team (business users, testers, and developers) discuss about acceptance criteria early in the development process. ATDD helps to ensure that all team members understand precisely what needs to be done and implemented.

ATDD is closely related to TDD. In this workflow usually concrete examples are created from a user perspective. Thereafter, acceptance tests are developed and run to see the results of failure with the right code based on these examples.

Q88. What is Jasmine?

Ans. Jasmine is BDD based testing framework that works for both client (browser) and server-side (nodejs) JavaScript. It also includes easy to understand documentation with code snippets.



Q89. What is Mocha?

Ans. Mocha is a JavaScript test runner for both client (browser) and server-side (node.js). With Mocha you can use a variety of assertion libraries of your choices. Chai is an assertion library widely used with mocha.



Q90. What are the most popular Node.js ORM?

Ans. The most popular Node.js ORM are as:

- Sequelize
- Node-ORM2
- Bookshelf
- Waterline
- JugglingDB
- Mongoose for mongoDB

Q91. What are JavaScript task runners?

Ans. JavaScript task runners are tools to make build operation simple and clean. They save time and automate build operations. JavaScript task runners also help us to manage web development workflow.



Grunt and **Gulp** are the most popular JavaScript task runners for managing web development workflow.

Q92. What are Grunt and Gulp?

Ans. Grunt and Gulp are task based command line build tools for JavaScript. They are built on the top of Node.js. They are used for automate build process, automate testing, deployment and release process. They support various plugins for front-end and node development.

Setup of Html minify task using Grunt (gruntfile.js)

```
grunt.loadNpmTasks('grunt-contrib-htmlmin');
grunt.initConfig({
  htmlmin: { // Task
    dist: { // Target
      options: { // Target options
        removeComments: true,
        collapseWhitespace: true
      },
      files: {
        'dist/index.html': 'src/index.html', // 'destination': 'source'
        'dist/contact.html': 'src/contact.html'
      }
    }
  }
});

grunt.registerTask('default', ['htmlmin']);
```

Setup of Html minify task using Gulp (gulpfile.js)

```
var gulp = require('gulp');
var minifyHtml = require("gulp-minify-html");

// task
gulp.task('minify-html', function () {
  gulp.src('./views/*.html') // path to html files
    .pipe(minifyHtml())
    .pipe(gulp.dest('./dist/views'));
});
```


Q93. What is the difference between Grunt and Gulp?

Ans. The differences between Grunt and Gulp are given below:

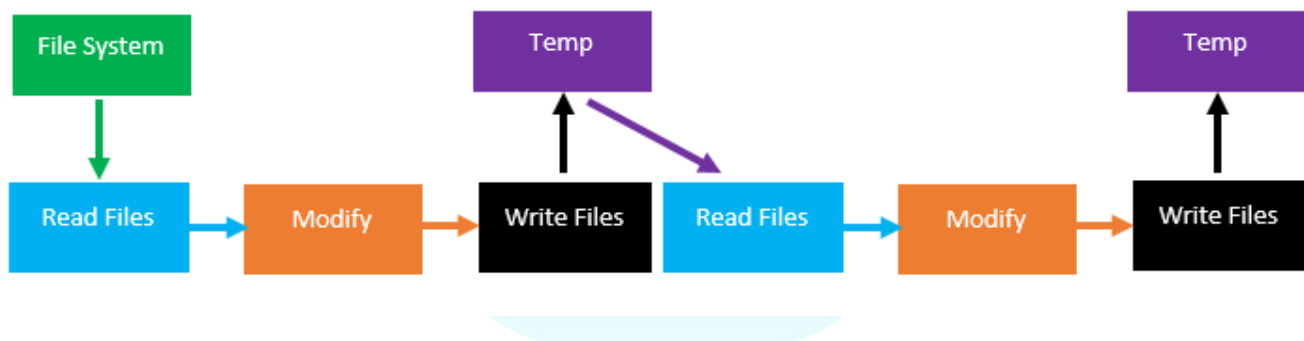
Grunt	Gulp
Configuration over code	Code over Configuration
File based	Stream based
5600+ Packages	2300+ Package
Widely used	Rising rapidly

Q94. How Grunt and Gulp deal with Task Automation?

Ans. The Grunt and Gulp deal with task automation differently. Grunt uses temp files while Gulp uses Node streams to perform task automation. The disk writes are slow as compared to in-memory operations; it means that Gulp is faster than Grunt.

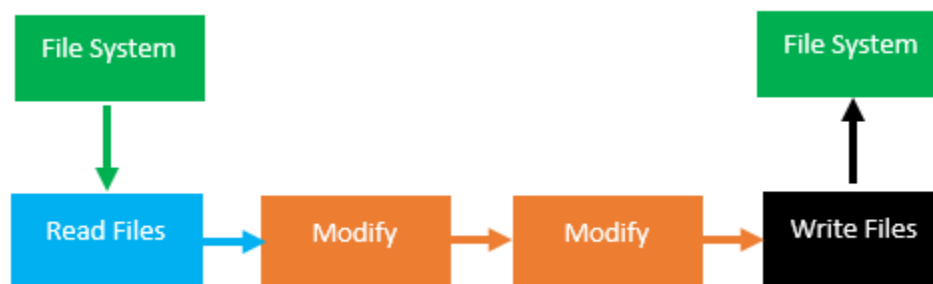
Grunt Task Automation Workflow

Grunt perform the task automation by creating temporary files which are disk I/O operations. In Grunt, to run two tasks on a group of files you need to configure an input and output for each task which result into two separate write to the disk as shown in the given fig.



Gulp Task Automation Workflow

Gulp uses Node streams to group tasks together and process them sequentially in memory. In Gulp, to run two tasks on a group of files, Gulp requires only one write to the disk as shown in the fig.



Q95. What is MEAN stack?

Ans. MEAN stands for MongoDB, Express, AngularJS and Node.js. MEAN is a most popular full-stack JavaScript solution for developing fast, robust and scalable web applications. It uses MongoDB to store data in JSON/BSON formats which makes it well suited to use with JavaScript. It uses Express as Node.js web application framework.

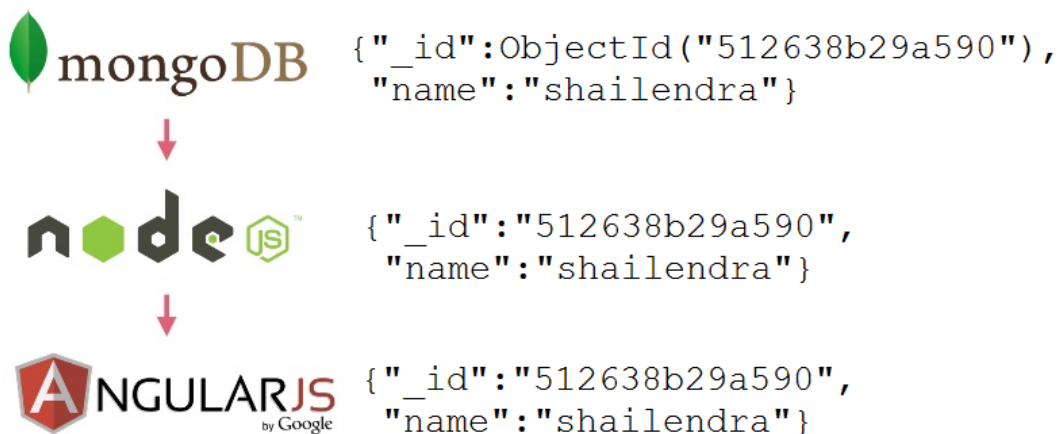


It uses AngularJS as front-end SPA development. It uses Node.js as Server Side JavaScript environment.

Q96. Why to use MEAN stack?

Ans. There are following reasons to use MEAN stack for web development.

- **One Language** - MEAN stack uses JavaScript as a programming language for both client-side and server-side. Language unification improve the developer productivity.
- **One Object** - Programmers write code in object oriented fashion and MEAN stack allows you to play with same object on client-side, server-side and also in database-side. So, there is no need to use libraries for converting data into objects during client-side and server-side interaction.



- **NoSQL database** – MEAN stack uses most popular NoSQL database (MongoDB) that is extremely flexible as compared to RDBMS. You don't need to worry about the schema changes and database migrations.
- **Cross Platform** - All the technologies used by the MEAN stack are cross platform compatible. They can run on windows, linux, mac etc.
- **Free and Open Source** - All the technologies used by the MEAN stack are free and open-source. Hence, you need to worry about the cost.

- **Community Support** - All the technologies used by the MEAN stack have very good community support. You will find a lot of samples for Node.js, Express, AngularJS and MongoDB.

Q97. What is MongoDB?

Ans. MongoDB is open-source, NoSQL database written in C++. It was introduced in 2009 by 10gen, now known as MongoDB Inc. It is document-oriented database which stores data in JSON/BSON format. It supports dynamic schema i.e. No DDL, No T-SQL.



It supports multiple platform like Windows, Linux, Mac etc.

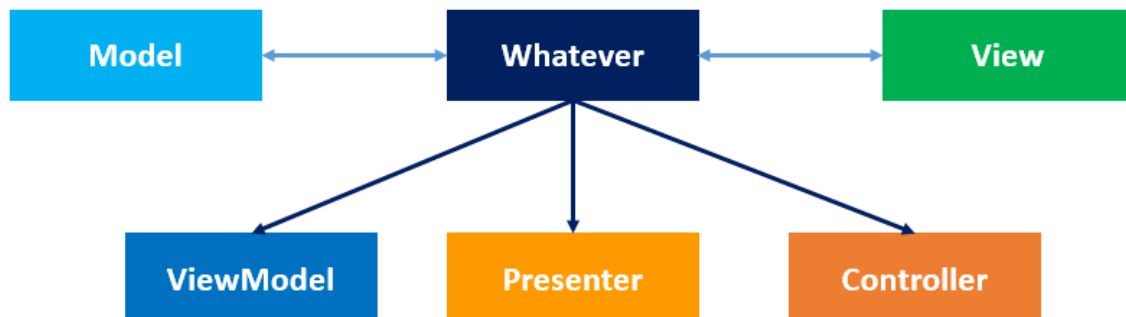
Q98. Why to use MongoDB?

Ans. Now-a-days, programmers write code in object oriented fashion using languages like C#, Python, Php, Java etc. Hence, programmers need a database which can store the data in objects. Since, querying and manipulating data objects is easy and it reduces the time of database operations.

And MongoDB stores data in objects. Hence querying MongoDB is easy as compared to RDBMS.

Q99. What is AngularJS?

Ans. AngularJS is an open-source JavaScript MVW framework developed by Google. It helps you to create single-page application that only require HTML, CSS, and JavaScript on the client side.



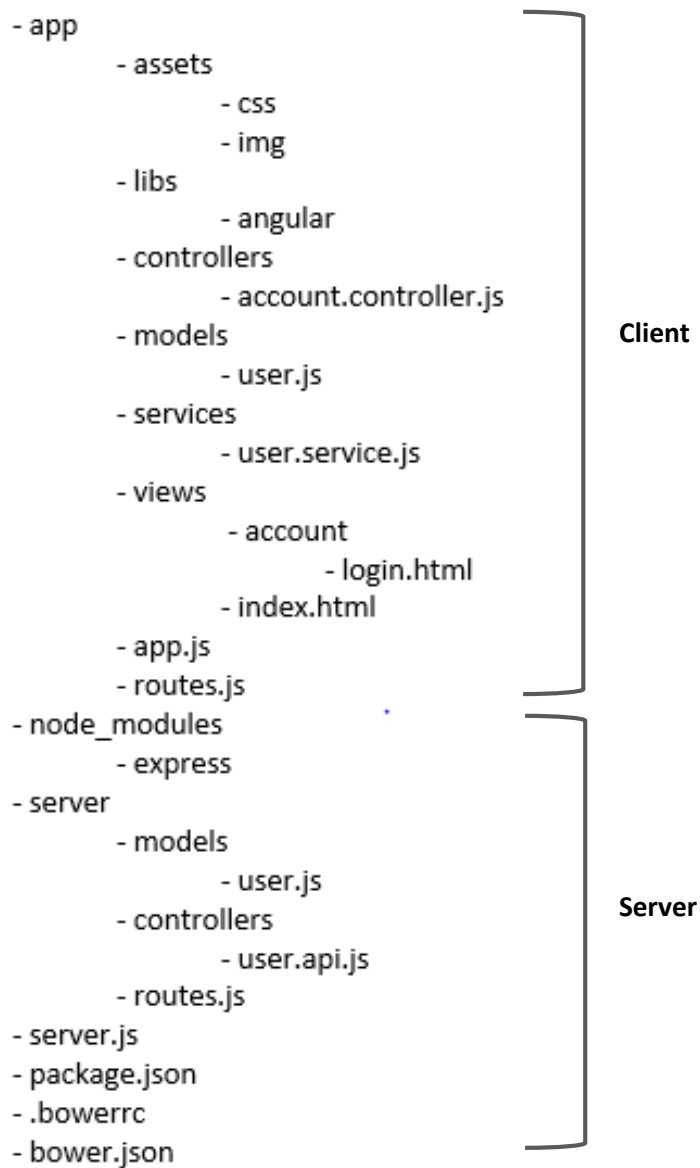
Angular acts as a compiler which compiles html and generate the output. It extends HTML by adding attr, tags & expressions. Also, it provides so many features as given below:

- Events Handling
- Powerful Data Binding
- Built-In Templating Engine
- Routing

- Form Validations
- REST API support using \$http, \$resource services
- Web Storage - Local Storage, Session Storage
- Cookies
- Animations
- Dependency Injection
- Testing

Q100. What is the recommended folder structure for MEAN app development?

Ans. A typical folder structure for MEAN application is given below:



Other Free E-Books

The author will truly appreciate it, if you will share his additional books among others because he wants it to reach as many techy people as possible so that they can also get benefited by his contribution.

You can download these books from www.dotnettricks.com in PDF format absolutely free.





SERVICES



Corporate Training



Campus Training



Industrial Training

Follow Us



/DotNetTricks



/DotNetTricksIn



+91-9871749695

Contact Us



+91-120 426 5198
+91-987 174 9695



info@dotnettricks.com
support@dotnettricks.com



B-64, Near SBI Bank
Sec-2 Noida-201301



www.dotnettricks.com
www.dotnet-tricks.com