18CS701

Hall Ticket Number:

IV/IV B.Tech (Regular\Supplementary) DEGREE EXAMINATION

November, 2022 — **Computer Science and Engineering**

Seventh Semester — **Full Stack Development**

Time: Three Hours — **Maximum:** 50 Marks

Answer question 1 compulsory. **(10X1 = 10Marks)**

Answer one question from each unit. **(4X10=40 Marks)**

			CO	BL	1 M
1	a)	List the benefits of using Node.js?	CO1	L1	
	b)	Write the uses of listen ( ) method in Node.js with syntax.	CO1	L4	
	c)	Define document in mongoDB.	CO1	L2	
	d)	What is mongo shell?	CO2	L4	
	e)	Define collection in MongoDB. Draw a collection hierarchy.	CO2	L2	
	f)	What are NoSQL databases? What are the different types of NoSQL databases?	CO2	L1	
	g)	What is MVC?	CO3	L3	
	h)	Write the purpose of Angular CLI.	CO3	L4	
	i)	List out the types of directives in Angular.	CO4	L3	
	j)	Write the built-in services in Angular.	CO4	L4	
		Unit-I			
2	a)	State the differences between Node.js and Express.js	CO1	L2	5M
	b)	Explain the differences between readFile and createReadStream in Node.js?	CO1	L3	5M
		(OR)			
3	a)	Demonstrate the use of EventEmitter in Node.js?	CO1	L1	5M
	b)	Write about differences between setImmediate( ) and setTimeout( )?	CO1	L2	5M
		Unit-II			
4	a)	Explain the procedure for creating cookie in Node.js with code.	CO2	L4	5M
	b)	Implement different collection operations in MongoDB. Give them example code.	CO2	L1	5M
		(OR)			
5	a)	Write differences between RDBMD and MongoDB.	CO2	L2	5M
	b)	Explain the steps involved in Express application that connect to MongoDB database with code.	CO2	L4	5M
		Unit-III			
6	a)	List and explain the built-in types in Typescript	CO3	L2	5M
	b)	What is TypeScript and why would I use it in place of JavaScript?	CO3	L3	5M
		(OR)			
7	a)	Illustrate the differences between AngularJS and Angular?	CO3	L2	5M
	b)	What is a template? How to use external templates in Angular? Explain with an example.	CO3	L2	5M
		Unit-IV			
8	a)	What is a data binding? Implement event-data binding with code in Angular.	CO4	L1	5M
	b)	Write the procedure for implement custom component in Angular with code.	CO4	L4	5M
		(OR)			
9	a)	Explain how custom events are implemented in Angular with code.	CO4	L4	5M
	b)	How do you implement Angular Service application? Explain the procedure with code.	CO4	L1	5M

IV/IV B.Tech (Regular) DEGREE EXAMINATION

November, 2022 **Computer Science & Engineering**
Seventh Semester **Full Stack Development (18CS701)**
Time: Three Hours **Maximum : 50 Marks**

SCHEME OF EVALUATION

Prepared By : **J.KUMARARAJA**
Assistant Professor,
Dept. of CSE,
BEC, BAPATLA.

(Signature)

Signature of the Faculty:

1. **Dr. N. Sudhakar**

2. **J.Kumararaja**

Signature of the External Faculty:

1.

2.

SCHEME OF EVALUATION

Allocation of Marks

UNIT – I

2. a) State the differences between Node.js and Express.js

[Consider Any Five Differences] – 5 Marks

2. b) Explain the differences between readFile and createReadStream in Node.js?

Explanation – 3 Marks
Program(s) – 2 Marks

(or)

3. a) Demonstrate the use of EventEmitter in Node.js?

Explanation – 3 Marks
Program – 2 Marks

3. b) Write about differences between setImmediate( ) and setTimeout( )?

Explanation – 3 Marks
Program(s) – 2 Marks

UNIT – II

4. a) Explain the procedure for creating cookie in Node.js with code.

Explanation – 3 Marks
Program(s) – 2 Marks

4. b) Implement different collection operations in MongoDB. Give them example code.

Explanation – 3 Marks
Program – 2 Marks

(or)

5. a) Write differences between RDBMD and MongoDB.

[Consider Any Five Differences] – 5 Marks

5. b) Explain the steps involved in Express application that connect to MongoDB database with code.

Explanation – 3 Marks
Diagram – 2 Marks

UNIT – III

6. a) List and explain the built-in types in Typescript

List Datatypes Names with explanation	**–**	**3 Marks**
Program	**–**	**2 Marks**

6. b) What is TypeScript and why would I use it in place of JavaScript?

Diagram	**–**	**2 Marks**
Explanation	**–**	**3 Marks**

(or)

7. a) Illustrate the differences between AngularJS and Angular?

[Consider Any Five Differences]	**–**	**5 Marks**

7. b) What is a template? How to use external templates in Angular? Explain with an example.

Definition & Explanations	**–**	**1+3 Marks**
Program	**–**	**2 Marks**

UNIT – IV

8. a) What is a data binding? Implement event-data binding with code in Angular.

Definition	**–**	**1 Mark**
Explanation	**–**	**2 Marks**
Program	**–**	**2 Marks**

8. b) Write the procedure for implement custom component in Angular with code.

Listing Names	**–**	**1 Marks**
Explanation	**–**	**2 Marks**
Program	**–**	**2 Marks**

(or)

9. a) Explain how custom events are implemented in Angular with code.

Explanation	**–**	**3 Marks**
Program	**–**	**2 Marks**

9. b) How do you implement Angular Service application? Explain the procedure with code.

Explanation	**–**	**3 Marks**
Program	**–**	**2 Marks**

- - - - -

IV/IV B.Tech (Regular) DEGREE EXAMINATION

November,2022 | **Computer Science & Engineering**
Seventh Semester | **Full Stack Development (18CS701)**
Time: Three Hours | **Maximum : 50 Marks**

SCHEME OF EVALUATION

Answer Questions No. 1 compulsorily ***(1 X 10 = 10 Marks)***
Answer ONE question from each unit. ***(4 X 10 = 40 Marks)***

1. Answer all questions **(1 X 10 = 10 Marks)**

a) List the benefits of using Node.js?

Node.js can be used for a wide variety of purposes. Because it is based on V8 and has highly optimized code to handle HTTP traffic, the most common use is as a webserver.

However, Node.js can also be used for a variety of other web services such as:

- Web services APIs such as REST (Representational State Transfer)
- Real-time multiplayer games
- Backend web services such as cross-domain, server-side requests
- Web-based applications
- Multiclient communication such as IM (Instant Messaging)

b) Write the purpose of listen( ) method in Node.Js application.

listen( ) function is used to bind and listen the connections on the specified host and port.
listen( ) method of server object which was returned from createServer( ) method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Sample Code:

```
// App listening on the below port
app.listen(PORT, function(err){
   if (err) console.log(err);
   console.log("Server listening on PORT", PORT);
});
```

c) Define document in mongoDB.

Documents

MongoDB stores data records as BSON documents, which are simply called documents.

A document is a set of field-and-value pairs with the following structure:

```
{
   field_name1: value1,
   field_name2: value2,
   field_name3: value3,
   ...
```

d) What is mongo shell?

MongoDB Shell is the quickest way to connect, configure, query, and work with your MongoDB database. It acts as a command-line client of the MongoDB server.

The MongoDB Shell is a standalone, open-source product and developed separately from the MongoDB Server under the Apache 2 license. It is a fully functional JavaScript and Node.js 14.x REPL for interacting with MongoDB servers.

e) Define collection in MongoDB. Draw a collection hierarchy.

Collection: Its group of MongoDB documents. This can be thought like a table in RDBMS like Oracle, MySQL. This collection doesn't enforce any structure. Hence schema-less MongoDB is so popular.

- **Document: Document is referred to as a record in MongoDB collection.**

Collection hierarchy

f) What are NoSQL databases? What are the different types of NoSQL databases?

NoSQL is a non-relational database that is used to store the data in the nontabular form. NoSQL stands for Not only SQL. The main types are documents, key-value, wide-column, and graphs.

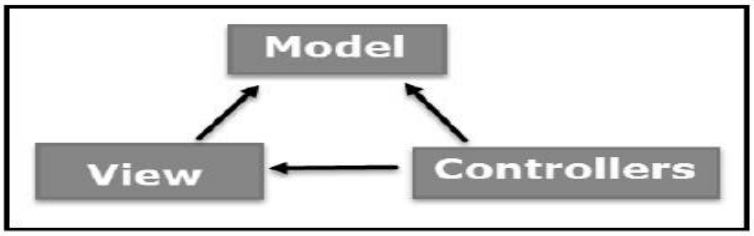
Types of NoSQL Database:

- Document-based databases.
- Key-value stores.
- Column-oriented databases.
- Graph-based databases

g) What is MVC?

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

Following are the components of MVC −



h) Write the purpose of Angular CLI.

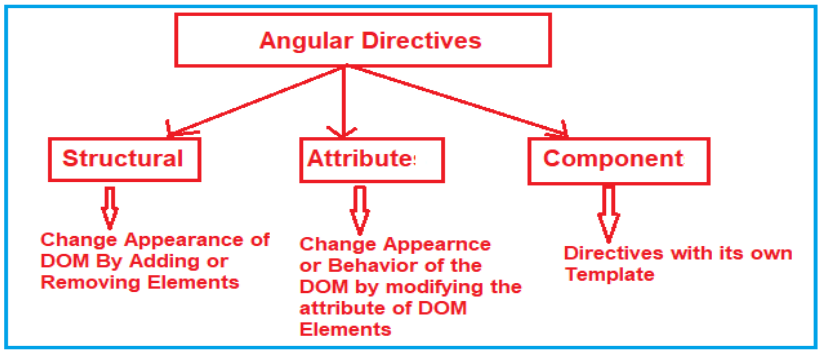
Angular CLI is basically a Command Line Interface that provides the capability to utilize the Command Window to execute and build the basic Angular application structure just by writing down the CLI Commands which Angular understands and interprets the action to be taken on each command. Angular CLI is compatible with all higher and major versions of Angular (NOT AngularJS).

i) List out the types of directives in Angular.

The Angular Directives are the elements which are basically used to change the behaviour or appearance or layout of the DOM (Document Object Model) element.

Types of Directives in Angular:

The Directives are classified into three types based on their behaviour. Please have a look at the following image for a better understanding of the directive's classification.



j) Write the built-in services in Angular.

Angular services are injectable object injected using dependency injection (DI) mechanism of Angular. The AngularJS services can be used to organize and share code across your app.

Here is a list of Angular built in Services with description.

$cookies , $parse, $timeout, $interval

UNIT – I

2. a) State the differences between Node.js and Express.js

S.No	Feature	Node.js	Express.js
1.	Usage	It is used to build server-side, input-output, event-driven apps.	It is used to build web-apps using approaches and principles of Node.js.
2.	Framework/Platform	Run-time platform or environment designed for server-side execution of JavaScript.	Framework based on Node.js.
3.	Controllers	Controllers are not provided.	Controllers are provided.
4.	Routing	Routing is not provided	Routing is provided.
5.	Middleware	Doesn't use such a provision.	Uses middleware for the arrangement of functions systematically server-side.
6.	Coding time	It requires more coding time.	It requires less coding time.
7.	Level of features	Fewer features.	More features than Node.js.
8.	Written in	Written in JavaScript C	JavaScript

2. b) Explain the differences between readFile and createReadStream in Node.js?

readFile: fs module contains the readFile method. It is used to read a file by bringing it into the buffer. It is an asynchronous method and due to this, it reads the file without blocking the execution of the code.

First, we bring the fs module into our app then use the readFile method inside it.

Syntax:

fs.readFile(filename, encoding, callback_function)

Example: In this example, we are reading the file using the readFile method, File to be read is output.txt.

output.txt file: This is an output file read from readFile method.

index.js

```
const fs = require('fs');
fs.readFile('output.txt', 'utf8', (err, data) => {
  console.log(`Data present in the file is::    ${data}`);
});
console.log('Outside readFile method');
```

Output:

```
Outside readFile method
Data present in the file is::
This is an output file read from readFile method.
```

createReadStream: fs module contains the inbuilt API(Application programming interface) createReadStream. It allows us to open a file/stream and reads the data present inside it.

Syntax:

fs.createReadStream(path, options)

Example: In this example, We are reading file name output.txt by createReadStream.on method.

Output.txt file: This is an output file read from createReadStream method.

index.js

```
const fs = require('fs');
const createReader = fs.createReadStream('output.txt');

createReader.on('data', (data) => {
  console.log(data.toString());
});

console.log('Outside createReader.on method');
```

Output:

```
Outside createReader.on method
This is an output file read from createReadStream method.
```

3. a) Demonstrate the use of EventEmitter in Node.js?

Events in Node.js

Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

Every action on a computer is an event. Like when a connection is made or a file is opened.

Node.js allows us to create and handle custom events easily by using events module.

Node.js events module has only one class to handle events which is EventEmitter class. It contains all required functions to take care of generating events.

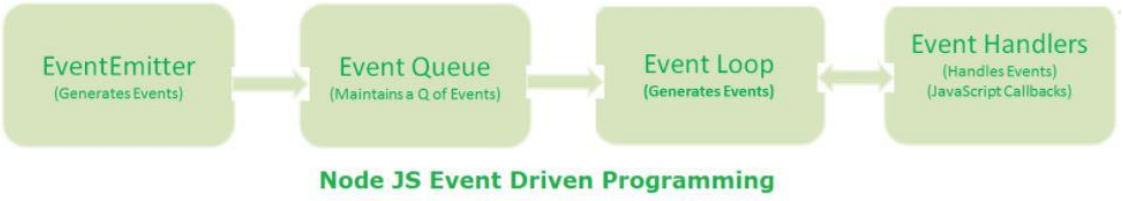
Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

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

var rs = fs.createReadStream('TestFile.txt');

rs.on('open', function () {

  console.log('The file is open');

});
```

EventEmitter Responsibility

EventEmitter class is responsible to generate events. Generating events is also known as Emitting. That's why this class is named as **EventEmitter**.

We can understand this Process through the given Diagram



Node JS Event Driven Programming

To include the built-in Events module use the require( ) method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

// Import events module

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

// Create an eventEmitter object

```
var eventEmitter = new events.EventEmitter();
```

Program:

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above
reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

//Raising
FirstEvent
```

In the above example, we first import the 'events' module and then create an object of EventEmitter class. We then specify event handler function using on( ) function. The on( ) method requires name of the event to handle and callback function which is called when an event is raised.

3. b) Write about differences between setImmediate( ) and setTimeout( )?

The setImmediate() function is used to execute a function right after the current event loop finishes. In simple terms, the function functionToExecute( ) is called after all the statements in the script are executed. It is the same as calling the setTimeout() function with zero delays. The setImmediate() function can be found in the Timers module of Node.js.

```
setImmediate(functionToExecute, [, ...args] )
```

It is followed by an optional list of parameters passed onto functionToExecute( ) as input parameters.

If the first parameter, i.e., functionToExecute() is not a function, then a TypeError will be thrown

Immediate timers are created using the setImmediate(callback,[args]) method built into Node.js. When you call setImmediate( ), the callback function is placed on the event queue and popped off once for each iteration through the event queue loop after I/O events have a chance to be called. For example, the following schedules myFunc() to execute on the next cycle through the event queue:

setImmediate(myFunc(), 1000);

The setImmediate() function returns a timer object ID. You can pass this ID to clearImmediate(immediateId) at any time before it is picked up off the event queue. For example:

var myImmediate = setImmediate(myFunc);
… clearImmediate(myImmediate);

We can see that the function inside the setImmediate( ) function is executed after all the statements in script have finished executing.

Program:

```
function myFunction(platform){
    console.log("Hi, Welcome to " + platform);
}

console.log("Before the setImmediate call")
let timerID = setImmediate(myFunction, "BEC");
console.log("After the setImmediate call")
for(let i=0; i<10; i++){
    console.log("Iteration of loop: "+i);
}
```

In the above example, an argument is passed to myFunction through the setImmediate() function.

The function passed to setImmediate() is called after all the statements in the script are executed.

Delaying Work with Timeouts

Timeout timers are used to delay work for a specific amount of time. When that time expires, the callback function is executed and the timer goes away. Use timeouts for work that only needs to be performed once.

Timeout timers are created using the setTimeout(callback, delayMilliSeconds, [args]) method built into Node.js. When you call setTimeout(), the callback function is executed after delayMilliSecondsexpires.

For example, the following executes myFunc()after 1 second:

setTimeout(myFunc, 1000);

The setTimeout() function returns a timer object ID. You can pass this ID to clearTimeout(timeoutId)at any time before the delayMilliSecondsexpires to cancel the timeout function. For example:

myTimeout = setTimeout(myFunc,100000);
… clearTimeout(myTimeout);

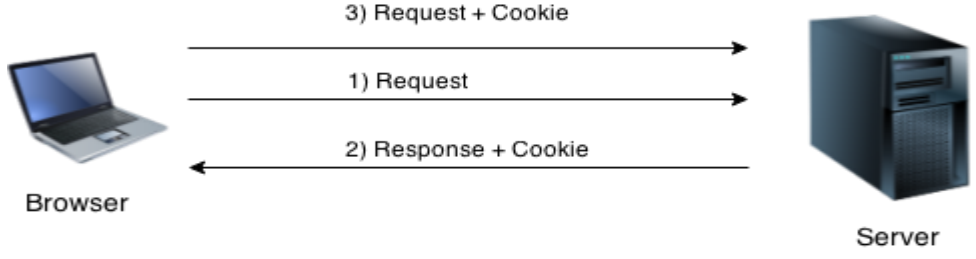
Below program implements a series of simple timeouts that call the simpleTimeout() function, which outputs the number of milliseconds since the timeout was scheduled. Notice that it doesn't matter which order setTimeout() is called; the results, shown in as shown in Output below, are in the order that the delay expires.

Implementing a series of timeouts at various intervals

simple_timer.js:

```
function simpleTimeout(consoleTimer){
     console.timeEnd(consoleTimer);
}
     console.time("twoSecond");
     setTimeout(simpleTimeout, 2000, "twoSecond");
     console.time("oneSecond");
     setTimeout(simpleTimeout, 1000, "oneSecond");
     console.time("fiveSecond");
     setTimeout(simpleTimeout, 5000, "fiveSecond");
     console.time("50MilliSecond");
     setTimeout(simpleTimeout, 50, "50MilliSecond");
```

, to recognize user.



Install cookie

You have to acquire cookie abilities in Express.js. So, install cookie-parser middleware through npm by using the following command:

npm install --save cookie-parser

Import cookie-parser into your app.

```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());
```

Define a route:

Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names.

Let's define a new route in your express app like set a new cookie:

```
app.get('/cookie',function(req, res){
res.cookie('cookie_name' , 'cookie_value').send('Cookie is set');
 });
app.get('/', function(req, res) {
console.log("Cookies :  ", req.cookies);
});
```

Browser sends back that cookie to the server, every time when it requests that website.

Express.js Cookies Example File: cookies_example.js

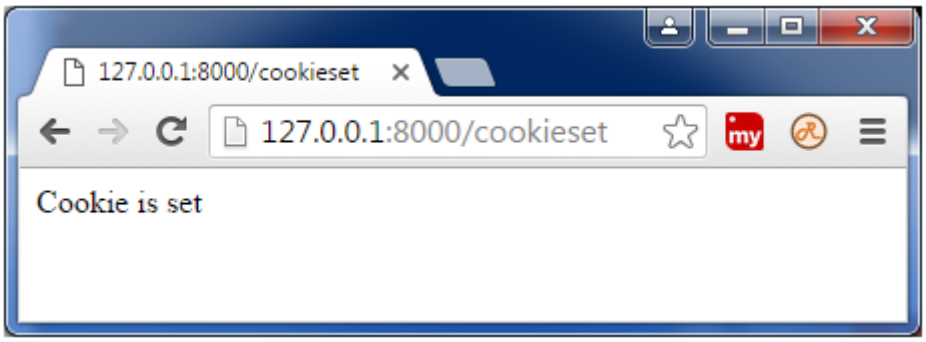
```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());
app.get('/cookieset',function(req, res){
res.cookie('cookie_name', 'cookie_value');
res.cookie('college', 'BEC');
res.cookie('name', 'KUMAR');
res.status(200).send('Cookie is set');
});
app.get('/cookieget', function(req, res) {
      res.status(200).send(req.cookies);
});
```

```
      app.get('/', function (req, res) {
            res.status(200).send('Welcome to BEC,BAPATLA');
      });
      var server = app.listen(8000, function () {
            var host = server.address().address;
            var port = server.address().port;
            console.log('Example app listening at http://%s:%s', host,
      port);
});
```

Open the page **http://127.0.0.1:8000/** on your browser to see output.

Set cookie:

Now open **http://127.0.0.1:8000/cookieset** to set the cookie:



Get cookie:

Now open **http://127.0.0.1:8000/cookieget** to get the cookie.

4. b) Implement different collection operations in MongoDB. Give them example code.

Basic CRUD operations

The following section shows you how to create (C), read (R), update (U), and delete (D) a document. These operations are often referred to as CRUD operations.

Create

To add a document to a collection, you use the insertOne() method of the collection.

The following command adds a new document (or a new book) to the books collection:

```
db.books.insertOne({
  title: "MongoDB Tutorial",
  published_year: 2020
})
```

Output:

```
{
 "acknowledged" : true,
 "insertedId" : ObjectId("5f2f39fb82f5c7bd6c9375a8")
}
```

Once you press enter, the mongo shell sends the command to the MongoDB server.

If the command is valid, MongoDB inserts the document and returns the result. In this case, it returns an object that has two keys acknowledged and insertedId.

The value of the insertedId is the value of the _id field in the document.

When you add a document to a collection without specifying the _id field, MongoDB automatically assigns a unique ObjectId value to the _id field and add it to the document. MongoDB uses the _id field to uniquely identify the document in the collection.

Read

To select documents in a collection, you use the findOne() method:

```
db.books.findOne()
```

Output:

```
{
  _id: ObjectId("62143f34cca1c7af0ad1d126"),
  title: 'MongoDB Tutorial',
  published_year: 2020
}
```

Code language: CSS (css)

To format the output, you use the pretty() method like this:

```
db.b().pretty()ooks.find
```

Code language: CSS (css)

Output:

```
{
        "_id" : ObjectId("5f2f3d8882f5c7bd6c9375ab"),
        "title" : "MongoDB Tutorial",
        "published_year" : 2020
}
```

As you can see clearly from the output, MongoDB added the _id field together with other field-and-value pairs to the document.

Update

To update a single document, you use the updateOne() method.

The updateOne() method takes at least two arguments:

- The first argument identifies the document to update.
- The second argument represents the updates that you want to make.

The following shows how to update the published_year of the document whose title is "MongoDB Tutorial":

```
db.books.updateOne(
            { title: "MongoDB Tutorial"},
            { $set: { published_year: 2019 }}
)
```

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

How it works.

The first argument identifies which document to update. In this case, it will update the first document that has the title "MongoDB tutorial":

```
{title: "MongoDB Tutorial"}
```

The second argument is an object that specifies which fields in the document to update:

```
{
   $set: {
      published_year: 2019
   }
}
```

The $set is an operator that replaces a field value with a specified value. In this example, it updates the published_year of the document to 2019.

Delete

To delete a document from a collection, you use the deleteOne() method. The deleteOne() method takes one argument that identifies the document that you want to delete.

The following example uses the deleteOne() method to delete a document in the books collection:

```
db.books.deleteOne({title: "MongoDB Tutorial"});
```

Output:

```
{
   "acknowledged": true,
   "deletedCount": 1
}
```

The output shows that one document has been deleted successfully via the deletedCount field.

To show all collections of the current database, you use the show collections command:

```
show collections
```

5. a) Write differences between RDBMD and MongoDB.

RDBMS	MongoDB
It is a relational database.	It is a non-relational and document-oriented database.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It is vertically scalable i.e increasing RAM.	It is horizontally scalable i.e we can add more servers.
It has a predefined schema.	It has a dynamic schema.
It is quite vulnerable to SQL injection.	It is not affected by SQL injection.
It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability).	It centers around the CAP theorem (Consistency, Availability, and Partition tolerance).
It is row-based.	It is document-based.
It is slower in comparison with MongoDB.	It is almost 100 times faster than RDBMS.
Supports complex joins.	No support for complex joins.
It is column-based.	It is field based.
It does not provide JavaScript client for querying.	It provides a JavaScript client for querying.
It supports SQL query language only.	It supports JSON query language along with SQL.

5. b) Explain the steps involved in Express application that connect to MongoDB database with code.

Step-1: create a directory and navigate to it.

```
$ mkdir express-mongodb
$ cd express-mongodb
```

Step-2 : Initialise npm on the directory and install the necessary modules. Also, create the index file.

```
$ npm init
$ npm install express
$ npm install ejs
$ touch index.js
$ npm install mongodb
```

Step 3: Initialize the express app and make it listen to a port on localhost and create a connection to mongodb.

Index.js

```
const express = require('express')
const {MongoClient, ObjectId} = require("mongodb")
const userPosts = require('./posts')
var app = express();
var col
app.set('viewengine', 'ejs')
app.set('views', 'my_views')
app.use(express.static('public'))
app.get('/', (req, res) => {
     var URI = "mongodb://ACS482:ACS482@10.2.2.22:27017/ACS482"
     var client = new MongoClient(URI)
     await client.connect()
     var db = client.db()
     col = db.collection("posts")
     var doc = col.find()
     var docarray = await.doc.toArray()
     res.render('displayposts',{userposts:docArray})
})
```

```
app.listen(3000, () => {
    console.log('server started on port 3000')
})
```

Step-4 : create a displayform to display the list of documents

Displayposts.ejs

```
<!DOCTYPE html>
<html>
<head>
   <title>Posts</title>
   <link rel='stylesheet' href="style.css">
</head>
<body>
   <table border="2px">
      <a href="/create">create
         <input type="button" align="center" value="createpost"></a>
      <h2>User posts</h2>
      <thead>
         <th>UserId</th>
         <th>Id</th>
         <th>title</th>
         <th>body</th>
      </thead>
      <% for (var i=0 ; i<userposts.length;i++ ){ % >
         <tr>
            <td> <% = userposts[i].userId %></td>
      <td> <% = userposts[i].id %></td>
      <td> <% = userposts[i].title %></td>
      <td> <% = userposts[i].body %></td>
```

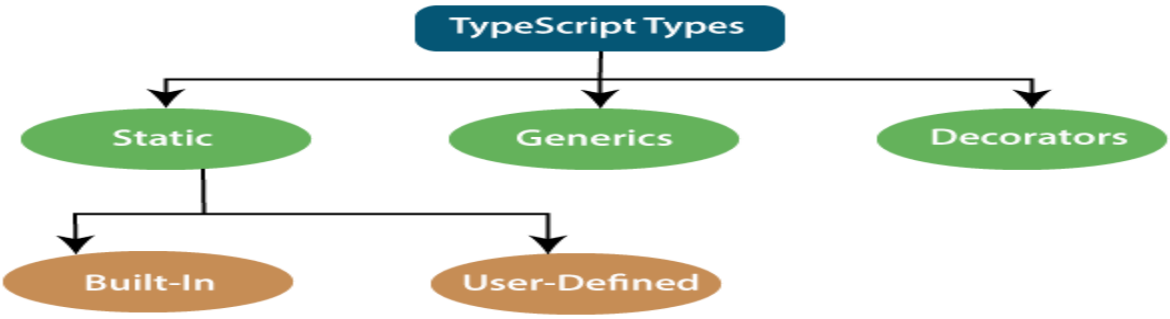
```
        </tr>
      <% } %>
      </table>
    </body>
</html>
```

6. a) List and explain the built-in types in Typescript

TypeScript Type

The TypeScript language supports different types of values. It provides data types for the JavaScript to transform it into a strongly typed programing language. JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript. TypeScript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.

TypeScript provides data types as an optional Type System. We can classify the TypeScript data type as following.



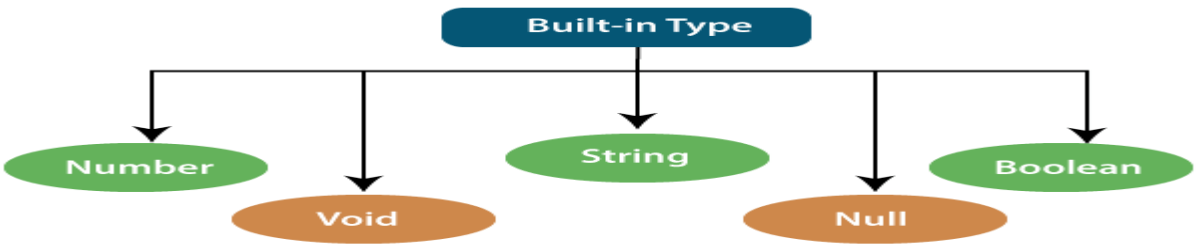
1. Static Types

In the context of type systems, static types mean "at compile time" or "without running a program." In a statically typed language, variables, parameters, and objects have types that the compiler knows at compile time. The compiler used this information to perform the type checking.

Static types can be further divided into two sub-categories:

Built-in or Primitive Type

The TypeScript has five built-in data types, which are given below.



Number

Like JavaScript, all the numbers in TypeScript are stored as floating-point values. These numeric values are treated like a number data type. The numeric data type can be used to represents both integers and fractions. TypeScript also supports Binary(Base 2), Octal(Base 8), Decimal(Base 10), and Hexadecimal(Base 16) literals.

Syntax:

1. let identifier: number = value;

Examples:-

```
1. let first: number = 12.0;              // number
2. let second: number = 0x37CF;           // hexadecimal
3. let third: number = 0o377 ;            // octal
4. let fourth: number = 0b111001;         // binary
5.
6. console.log(first);            // 123
7. console.log(second);           // 14287
8. console.log(third);            // 255
9. console.log(fourth);           // 57
```

String

We will use the string data type to represents the text in TypeScript. String type work with textual data. We include string literals in our scripts by enclosing them in single or double quotation marks. It also represents a sequence of Unicode characters. It embedded the expressions in the form of **$ {expr}**.

Syntax

let identifier: string = " ";

Or

let identifier: string = ' ';

Examples

1. let empName: string = "Rohan";
2. let empDept: string = "IT";
3.
4. // Before-ES6
5. let output1: string = empName + " works in the " + empDept + " department.";
6. // After-ES6
7. let output2: string = `${empName} works in the ${empDept} department.`;
8. console.log(output1);//Rohan works in the IT department.
9. console.log(output2);//Rohan works in the IT department.

Boolean

The string and numeric data types can have an unlimited number of different values, whereas the Boolean data type can have only two values. They are "true" and "false." A Boolean value is a truth value which specifies whether the condition is true or not.

Syntax

1. let identifier: boolean = Boolean value;

Examples

1. let isDone: boolean = false;

void

A void is a return type of the functions which do not return any type of value. It is used where no data type is available. A variable of type void is not useful because we can only assign undefined or null to them. An undefined data type denotes uninitialized variable, whereas null represents a variable whose value is undefined.

Syntax

1. let unusable: void = undefined;

Examples

```
function helloUser( ): void {
        alert("This is a welcome message");
}
let tempNum: void = undefined;
 tempNum = null;
 tempNum = 123;      //Error
```

Null

Null represents a variable whose value is undefined. Much like the void, it is not extremely useful on its own. The Null accepts the only one value, which is null. The Null keyword is used to define the Null type in TypeScript, but it is not useful because we can only assign a null value to it.

Examples

```
let num: number = null;
let bool: boolean = null;
let str: string = null;
```

Undefined

The Undefined primitive type denotes all uninitialized variables in TypeScript and JavaScript. It has only one value, which is undefined. The undefined keyword defines the undefined type in TypeScript, but it is not useful because we can only assign an undefined value to it.

Example

```
let num: number = undefined;
let bool: boolean = undefined;
let str: string = undefined;
```

Any Type

It is the "super type" of all data type in TypeScript. It is used to represents any JavaScript value. It allows us to opt-in and opt-out of type-checking during compilation. If a variable cannot be represented in any of the basic data types, then it can be declared using "**Any**" data type. Any type is useful when we do not know about the type of value (which might come from an API or 3rd party library), and we want to skip the type-checking on compile time.

Syntax

```
let identifier: any = value;
```

Examples

```
let val: any = 'Hi';
      val = 555;   // OK
      val = true;   // OK

          function ProcessData(x: any, y: any) {
                    return x + y;
            }
      let result: any;
      result = ProcessData("Hello ", "Any!"); //Hello Any!
      result = ProcessData(2, 3); //5
```

6. b) What is TypeScript and why would I use it in place of JavaScript?

The TypeScript language supports different types of values. It provides data types for the JavaScript to transform it into a strongly typed programing language. JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript. TypeScript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.

Advantage of TypeScript over JavaScript

- TypeScript always highlights errors at compilation time during the time of development, whereas JavaScript points out errors at the runtime.
- TypeScript supports strongly typed or static typing, whereas this is not in JavaScript.
- TypeScript runs on any browser or JavaScript engine.
- Great tooling supports with IntelliSense which provides active hints as the code is added.
- It has a namespace concept by defining a module.

```
//Defining a Student class.
class Student {
     //defining fields
          id: number;
         name:string;
          constructor(id: number, name: string) {
                 this.id = id;
                    this.name = name;
         }
//creating method or function
  display():void {
     console.log("Student ID is: "+this.id)
     console.log("Student ID is: "+this.name)
     }
 }
//Creating an object or instance
let obj = new Student(101,"Virat Kohli");
     //obj.id = 101;
     //obj.name = "Virat Kohli";
     obj.display();
```

7. a) Illustrate the differences between AngularJS and Angular?

Category	Angular JS	Angular
Architecture	It supports the Model-View-Controller design. The view processes the information available in the model to generate the output.	It uses components and directives. Components are the directives with a template.
Written Language	Written in JavaScript.	Written in Microsoft's TypeScript language, which is a superset of ECMAScript 6 (ES6).
Mobile support	It does not support mobile browsers.	Angular is supported by all the popular mobile browsers.
Expression Syntax	**ng-bind** is used to bind data from view to model and vice versa.	Properties enclosed in "()" and "[]" are used to bind data between view and model.
Dependency Injection	It does not use Dependency Injection.	Angular is supported by all the popular mobile browsers.
Supported Languages	It only supports JavaScript.	It provides support for TypeScript and JavaScript.
Routing	AngularJS uses $routeprovider.when() for routing configuration.	Angular uses @Route Config{(…)} for routing configuration.
Structure	It is less manageable in comparison to Angular.	It has a better structure compared to AngularJS, easier to create and maintain for large applications but behind AngularJS in the case of small applications.
CLI	It does not come with a CLI tool.	It comes with the Angular CLI tool.
Examples Application	iStock, Netflix, and Angular JS official website.	Upwork, Gmail, and Wikiwand.

7. b) What is a template? How to use external templates in Angular? Explain with an example.

Templates in Angular represents a view whose role is to display data and change the data whenever an event occurs. It's default language for templates is HTML.

Templates separate view layer from the rest of the framework so we can change the view layer without breaking the application.

Elements of Templates

1. HTML
2. Interpolation
3. Template Expressions
4. Template Statements

Let's start with the explanation of each one of the template elements 👉

HTML

Angular uses HTML as a template language.

Interpolation

Interpolation is one of the forms of data binding where we can access a component's data in a template. For interpolation, we use double curly braces {{ }}.

Template Expressions

The text inside {{ }} is called as template expression.

Ex,

1. {{Expression}}

Angular first evaluates the expression and returns the result as a string. The scope of a template expression is a component instance. That means, if we write {{ Name }}, Name should be the property of the component to which this template is bound to.

Template Statements

Template Statements are the statements which respond to a user event.

1. (event) = {{Statement}}

Ex : lets create an click event - Add changename() method inside hi.component.ts file as below,

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.   selector: 'app-hi',
5.   templateUrl: `./hi.component.html`,
6.   styleUrls: ['./hi.component.css']
7. })
8. export class HiComponent implements OnInit {
9.   Name : string = "XYZ";
10.   changeName() {
```

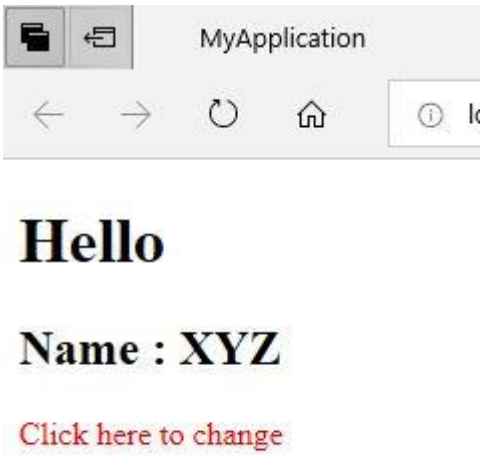
```
11.      this.Name = "ABC";
12. }
13.   constructor() { }
14.   ngOnInit(): void {
15.   }
16. }
```

Now open hi.component.html and add the below line of code inside it,

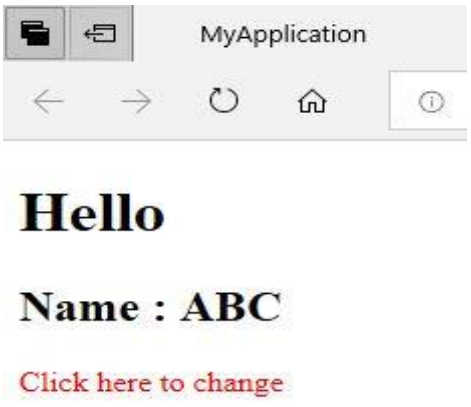
```
1. <h1>Hello</h1>
2. <h2>Name : {{Name}}</h2>
3. <p (click)="changeName()">Click here to change</p>
```

Here, changeName() method is bound to click event which will be invoked on click at run time. This is called event binding

Now observe the output screen in the browser,



Output after clicking. When user clicks on the paragraph, course name will be changed to 'ABC'



8. a) What is a data binding? Implement event-data binding with code in Angular.

In Angular, Data Binding means to bind the data (Component's filed) with the View (HTML Content). That is whenever you want to display dynamic data on a view (HTML) from the component then you need to use the concept Data binding.
Data Binding is a process that creates a connection to communicate and synchronize between the user interface and the data. In order words, we can say that Data Binding means to interact with the data and view. So, the interaction between the templates (View) and the business logic is called data binding.

Angular Event Binding with Examples

Event Binding

When a user interacts with an application in the form of a keyboard movement, button click, mouse over, selecting from a drop-down list, typing in a textbox, etc. it generates an event. These events need to be handled to perform some kind of action. This is where event binding comes into the picture and in Angular Application, we can use event binding to get notified when these events occur.

Event Binding work in Angular

The following image shows the syntax for binding to the click event of a button. Within parentheses on the left-hand side of the equal sign, we have the target event, (click in this case) and on the right-hand side, we have the template statement such as Component properties or methods. In this case, it is the component method i.e. **onClick**() method which is going to be called when the button click event occurs.

```
<button (click)="onClick()">Click Me </button>
```

With event binding, you can also use the on- prefix alternative as shown in the image below. This is known as the canonical form. It's up to you which approach you follow. Behind the scene, they are going to perform the same task.

```
<button on-click="onClick()">Click Me </button>
```

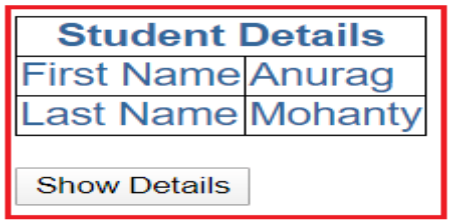
Angular Event Binding Example:

Let us understand Angular Event Binding with an example. Please modify the **app.component.ts** file as shown below.

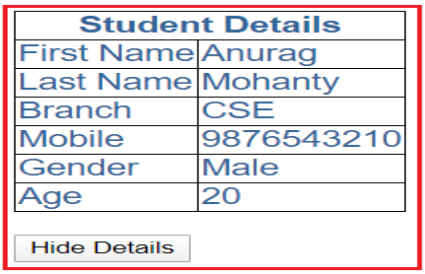
Now, run the application and launch the browser developer tools by pressing the F12 key. Once you open

Another Example:

When the page loads for the first time, we want to display only the First Name and Last Name of the student. We also display the "Show Details" button as shown in the below image.



When the user clicks on the **"Show Details"** button, we want to display the "**Gender**", "**Age**", "**Mobile**", and "**Branch**" as well. The text on the button should be changed to **"Hide Details"** as shown in the below image and when the user clicks on the **"Hide Details"** button, then the "**Gender**", "**Age**", "**Mobile**", and "**Branch**" should be hidden and the button text should be changed to **"Show Details"**.



We can achieve this very easily in angular with the help of event binding. Here we will make use of one of the angular directives i.e. "**ngIf**".

Modify app.component.ts file:

Notice we have introduced "ShowDetails" boolean property. The default value is false, so when the page loads for the first time, we will have "Gender", "Age", "Mobile", and "Branch" hidden. We also have a method, ToggleDetails(), which will toggle the value of ShowDetails. The ngIf directive conditionally adds or removes content from the DOM based on whether or not an expression is true or false. If "**ShowDetails**" is true, "Gender", "Branch", "Mobile" and "Age" <tr> elements are added to the DOM, else removed.

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-eventbind',
  templateUrl: './eventbind.component.html',
  styleUrls: ['./eventbind.component.css']
})
export class EventbindComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }
  ColumnSpan: number = 2;
  FirstName: string = 'Khumaini';
  LastName: string = "Shaik";
  Branch : String = "CSE";
  Mobile: number = 24589;
  Gender: string = "Male";
  Age: number = 21;

  ShowDetails: boolean = false;
  ToggleDetails(): void {
    this.ShowDetails = !this.ShowDetails;
  }

}
```

Modify app.component.html file:

Notice the click event of the button element is bounded to ToggleDetails() method. To dynamically change the text on the button, we are using a ternary operator:

{{ShowDetails ? 'Hide' : 'Show'}} Details

We used ngIf structural directive on "Gender", "Branch", "Mobile" and "Age" <tr> elements. The * prefix before a directive indicates, it is a structural directive.

The ngIf directive conditionally adds or removes content from the DOM based on whether or not an expression is true or false. If "**ShowDetails**" is true, "Gender", "Branch", "Mobile" and "Age" <tr> elements are added to the DOM, else removed.

```
<table>
    <thead>
        <tr>
            <th attr.colspan="{{ColumnSpan}}">
                Student Details
            </th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>First Name</td>
            <td>{{FirstName}}</td>
        </tr>
        <tr>
            <td>Last Name</td>
            <td>{{LastName}}</td>
        </tr>
        <tr *ngIf='ShowDetails'>
            <td>Branch</td>
            <td>{{Branch}}</td>
        </tr>
        <tr *ngIf='ShowDetails'>
            <td>Mobile</td>
            <td>{{Mobile}}</td>
        </tr>
        <tr *ngIf='ShowDetails'>
            <td>Gender</td>
            <td>{{Gender}}</td>
        </tr>
        <tr *ngIf='ShowDetails'>
            <td>Age</td>
            <td>{{Age}}</td>
        </tr>
    </tbody>
</table>
<br/>
<button (click)='ToggleDetails()'>
    {{ShowDetails ? 'Hide' : 'Show'}} Details
</button>
```

Modify app.component.css file:

Modify the app.component.css file as shown below.
Now run the application and you will see everything is working as expected as per our requirement.

```
table {
    color: #369;
    font-family: Arial, Helvetica, sans-serif;
    font-size:large;
    border-collapse: collapse;
}
td {
```

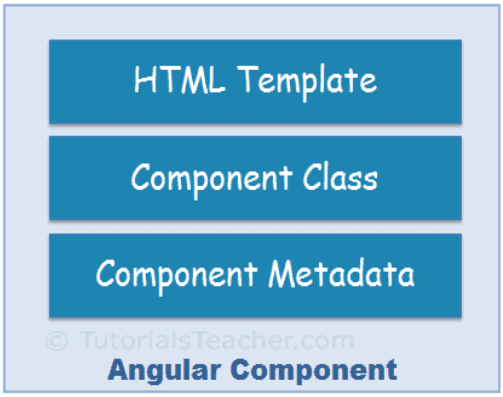
8. b) Write the procedure for implement custom component in Angular with code.

Angular Component

Angular is a SPA framework, and a view is made of one or more component. An Angular component represents a portion of a view.

Generally, an interactive web page is made of HTML, CSS, and JavaScript. Angular component is no different.

Angular Component = HTML Template + Component Class + Component Metadata



HTML Template

HTML template is nothing but a regular HTML code with additional Angular specific syntax to communicate with the component class.

Class

Essentially, a component class is a TypeScript class that includes properties and methods. Properties store data and methods include the logic for the component. Eventually, this class will be compiled into JavaScript.

Note:

TypeScript is an open-source, object-oriented language developed and maintained by Microsoft. It is a typed superset of JavaScript that compiles to plain JavaScript.

Metadata

Metadata is some extra data for a component used by Angular API to execute the component, such as the location of HTML and CSS files of the component, selector, providers, etc.

Generate Angular Component using Angular CLI

You can create files for a component manually or using the Angular CLI command. Angular CLI reduces the development time. So, let's use Angular CLI to create a new component.

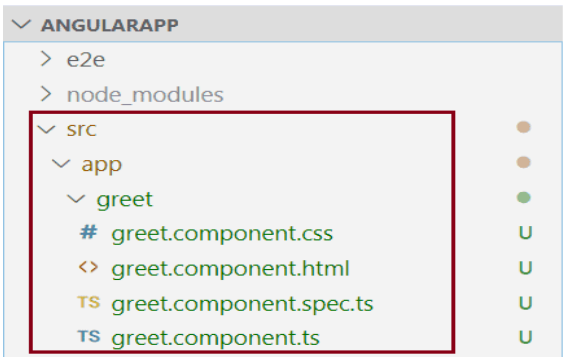
Use the following CLI command to generate a component.

```
ng generate component <component name>
```

All Angular CLI command starts with ng, generate or g is a command, component is an argument and then the name of the component.

The following executes the ng g command to generate the greet component in VS Code.

The above command will create a new "greet" folder and app folder and create four files, as shown below.



Component Files

Above, greet.component.css is a CSS file for the component, greet.component.html is an HTML file for the component where we will write HTML for a component, greet.component.spec.ts is a test file where we can write unit tests for a component, and greet.component.ts is the class file for a component.

Note:

A component can have a single file or multiple files. A single TypeScript file can include an HTML template, component class, and component metadata.

The following figure illustrates the important part of the component class.

The greet.component.ts includes the following parts:

Component Class: GreetComponent is the component class. It contains properties and methods to interact with the view through an Angular API. It implements the OnInit interface, which is a lifecycle hook.

Component Metadata: The @Component is a decorator used to specify the metadata for the component class defined immediately below it. It is a function and can include different configs for the component. It instructs Angular where to get required files for the component, create and render component. All Angular components must have @Component decorator above the component class.

The import statement gets the required feature from the Angular or other libraries. Import allows us to use exported members from external modules. For example, @Component decorator and OnInit interface are contained in @angular/core library. So, we can use them after importing it.

Now, let's add a property and method in the component class, as shown below.

Example: Add Properties and Methods in the Component Class

```
export class GreetComponent implements OnInit {

        constructor() { }

              ngOnInit(): void {
        }

        name: string = "Steve";

         greet(): void {
                alert("Hello " + this.name);
              };

       }
```

Above, we have added the name property and the greet method in the component class. Let's use these in the HTML template.

Open greet.component.html file, remove existing code and add the following code.

greet.component.ts

```
<div>
    Enter Your Name: <input type="text" value={{name}} />
<br />
    <button (click)="greet()">Greet Me!</button>
</div>
```

In the above HTML template, we used name property in the {{ }} interpolation to display its value and greet() function as click event. Lean more about it in event binding section.

Bootstrapping Component

Now, it's time to load our component, but before that, we need to host our application and load the root component. This process is called bootstrapping.

Angular is a single page application (SPA) framework. So, we need to host our application in index.html, and then we need to define a root module to bootstrap our root component. Index.html will be the only web page in an Angular application, and that's why it is called SPA.

When you generate Angular application using Angular CLI, it automatically creates index.html, root component app.component.ts, root module app.module.ts, and HTML template app.component.html for you. The AppComponent class in app.component.ts is a root component, and the AppModule class in app.module.ts is a root module.

Here, we will load our greet component into the root component in two steps.

1. Declare a component in the root module.

We want to load our new GreetComponent into the root component. So, the root module must know about it. We can do it by adding the GreetComponent in the declarations array in app.module.ts, as shown below.

Example: Add Component in the root module app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { GreetComponent } from './greet/greet.component';
//import GreetComponent
@NgModule({
  declarations: [
    AppComponent,
    GreetComponent  // <- include GreetComponent in
declarations
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Add component tag in the root HTML template.

After adding a component declaration, use component tag <app-greet></app-greet> in the HTML file of the root component, which is app.component.html, as shown below.

app.component.html

```
<div>
    <app-greet></app-greet>
 </div>
```

We can also create a single component file greet.component.ts if the HTML code of a component is less. Use the template parameter in the @Component decorator to include HTML of the component. The following greet component gives the same result as above.

Example: Component Class with HTML Template

```
import { Component } from '@angular/core';

@Component({
    selector: "app-greet",
    template: `<div>
    Enter Your Name: <input type="text" value={{name}} />
<br/>
    <button (click)="greet()">Greet Me!</button>
    </div>`
})

export class GreetComponent {

    name: string = "Steve";
    greet(): void {
        alert("Hello " + this.name);
    };
```

9. a) Explain how custom events are implemented in Angular with code.

Sometimes the DOM events are not enough and there is a need to raise custom events. For example, a child component may need to emit an event and tell the parent component that it has updated itself. The parent component then listens to this event and may need to update itself. The flow of data in this case is one way, that is from the child component to parent component.

In Angular we can raise custom events. Our custom events can pass data according to our requirement. Custom event is a way of communication between components.

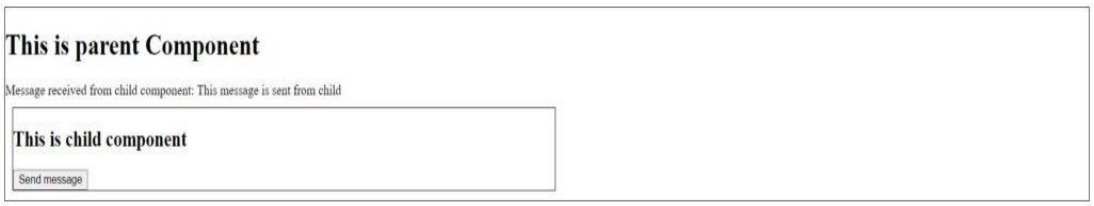
Usually the custom events are generated by directives and are used to communicate from child component to parent component. The steps to emit custom events and its use are:

- **Step 1**: Declare a property of type EventEmitter in child or the component who will emit the event.
  @Output() customMessageEvent = new EventEmitter();
  Decorate the property with @Output( ) decorator.
- **Step 2**: Emit the event with data using the emit method.
  this.customMessageEvent.emit('This message is sent from child');
- **Step 3**: Use the custom event property in the HTML using event binding syntax, i.e., enclosed in parentheses and define the template expression to be called when event is emitted.
  <app (customMessageEvent)="changeMessageInParent($event)"></app-child>
- When custom event is emitted, changeMessageInParent method of parent component will be called.

Example of custom events with EventEmitter

In this example, we'll do the following:

1. Create two components, parent and child. Child component is used in parent component and thus two components have parent-child relationship.
2. When we click a button in child component, the child component will emit an event to the parent component with a message.
3. The parent component will listen to that event and will show the message received from the child component.



parent.component.html

```
<div style="border: 1px solid black">
<h1>This is parent Component</h1>
```

```
<span>Message received from child component: {{ messageFromChild
}}</span>
<app-child (customMessageEvent)="changeMessageInParent($event)"></app-
child>
</div>
```

parent.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
selector: 'app-parent',
templateUrl: './parent.component.html',
styleUrls: ['./parent.component.css'],
})
export class ParentComponent implements OnInit {
messageFromChild: string = '';
constructor() {}
ngOnInit(): void {}
changeMessageInParent(event: string) {
this.messageFromChild = event;
}
}
```

child.component.html

```
<div style="border: 1px solid black; width: 50%; margin: 10px 10px 10px
10px">
<h2>This is child component</h2>
<button type="button" (click)="sendMessageToParent()">Send
message</button>
</div>
```

child.component.ts

```
import { Component, EventEmitter, OnInit, Output } from '@angular/core';
@Component({
selector: 'app-child',
templateUrl: './child.component.html',
styleUrls: ['./child.component.css'],
})
export class ChildComponent implements OnInit {
@Output() customMessageEvent = new EventEmitter<string>();
ngOnInit(): void {}
sendMessageToParent() {
this.customMessageEvent.emit('This message is sent from child');
}
```

```
}
```

Let us now understand the example.

1. We have exposed a property customeMessageEvent in child component with @Output decorator. This property is of type EventEmitter. Our custom event emitter emits data of type string.
2. When the button is clicked in child component, we are calling sendMessageToParent method.
3. sendMessageToParent method emits custom event with data as string message This message is sent from child.
4. In parent component, we are using child component with our custom event property. <app-child (customMessageEvent)="changeMessageInParent($event)"></app-child>
5. Whenever custom event is emitted, changeMessageInParent method of parent component is called with event object $event. This event object contains the message received from the child component.
6. changeMessageInParent receives the data from event and set the messageFromChild property of parent. This messageFromChild property is then use to show message from the child on screen.

9. b) How do you implement Angular Service application? Explain the procedure with code.

The Angular Services are the piece of code or logic that are used to perform some specific task. A service can contain a value or function or combination of both. The Services in angular are injected into the application using the dependency injection mechanism.

Let us Angular Service step by step with an example.

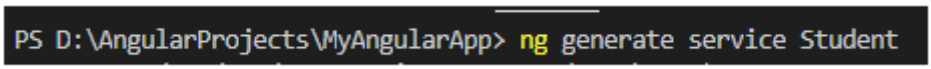
First we will understand how to create angular service and then we will discuss how to use angular service within a component.

Step1: Creating Angular Service

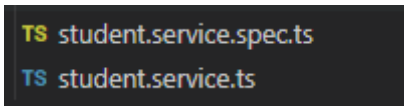
The angular service is composed of three things. You need to create an **export** class and you need to decorate that class with **@Injectable** decorator and you need to import the Injectable decorator from the **angular core** library. The syntax to create angular service is given below.

```
import { Injectable } from '@angular/core';
@Injectable()
export class ServiceName {
   //Method and Properties
}
```

Let say you want to create an angular service for fetching the student details and this student details is going to be used by many components. So, open terminal and type **ng generate service Student** and press enter as shown below.



Once you press the enter button it will create two files within the app folder as shown below.



Modifying student.service.ts file:

Open **student.service.ts** file and then copy and paste the following code in it. At the moment we have hard-coded the student data, later in this article series we will discuss how to get this data from restful APIs. Here, you need to focus on two things. The **@Injectable** decorator and the **getStudents** method which returns the student data. As with all other angular decorators, we preceded the name with an @ symbol, and we do not add a semicolon (;) at the end.

Code:

```
import { Injectable } from '@angular/core';
@Injectable()
export class StudentService {
getStudents(): any[] {
return [
   {
      ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',
      Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'
      },
   {
      ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',
      Branch: 'ETC', DOB: '23/05/1989', Gender: 'Male'
      },
  {
      ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',
      Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'
      },
  {
      ID: 'std104', FirstName: 'Hina', LastName: 'Sharma',
      Branch: 'ETC', DOB: '19/08/1990', Gender: 'Female'
      },
  {
      ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy',
      Branch: 'CSE', DOB: '12/94/1991', Gender: 'Male'
      }
  ];
 }
}
```

Note: The **@Injectable( )** decorator in angular is used to inject other dependencies into the service. At the moment our service does not have any other dependencies, so, you can remove the **@Injectable( )** decorator and the service should works. However, the Angular Team recommends to always use **@Injectable( )** decorator to ensures consistency.
