1. Create a student's academic database with student name, year/semester, course name, course grade, CGPA. The collection should contain 5 documents foratleast3 semesters.

```
[  {    "name": "Alice Jones",
"year": 1,
"semester": 1,
"courses": [
{
"name": "Introduction to Computer Science",
 "grade": 90,
"credits": 3      },
{
"name": "Calculus I",
"grade": 85,
"credits": 4      },

{        "name": "English Composition",
"grade": 87,
"credits": 3      }    ],
    "CGPA": 3.53
  },
  {
    "name": "Bob Smith",
    "year": 1,
    "semester": 2,
    "courses": [
      {
        "name": "Data Structures and Algorithms",
        "grade": 92,
        "credits": 4
      },
      {
        "name": "Calculus II",
        "grade": 87,
        "credits": 4
    },
    {
      "name": "Introductory Physics",
      "grade": 84,
      "credits": 4
    }
  ],
  "CGPA": 3.68
},
{
  "name": "Charlie Brown",
  "year": 2,
  "semester": 1,
  "courses": [
    {
      "name": "Computer Networks",
      "grade": 89,
      "credits": 4
    },
    {
      "name": "Differential Equations",
      "grade": 88,
      "credits": 4
    },
    {
      "name": "Philosophy of Mind",
      "grade": 93,
      "credits": 3
    }
  ],
  "CGPA": 3.85
},
{
  "name": "David Lee",
      "year": 2,
      "semester": 2,
      "courses": [
        {
          "name": "Operating Systems",
          "grade": 94,
          "credits": 4
        },
        {
          "name": "Linear Algebra",
          "grade": 90,
          "credits": 4
        },
        {
          "name": "Environmental Science",
          "grade": 83,
          "credits": 3
        }
      ],
      "CGPA": 3.89
    },
    {
      "name": "Emma Davis",
      "year": 3,
      "semester": 1,
      "courses": [
        {
          "name": "Database Systems",
          "grade": 91,
          "credits": 4
        },
        {
          "name": "Algorithms and Complexity",
          "grade": 85,
          "credits": 4
        },
        {
          "name": "Artificial Intelligence",
          "grade": 92,
          "credits": 3
        }
      ],
      "CGPA": 3.91
    }
]
```

1.1 Write a MongoDB query to display the fieldsstudent name, year/semester, course name, course gradefor all the documents in the collection.

1.2 Write a MongoDB query to display the fieldsstudent name,course name where obtained course gradeis ‘A’ from the collection.

1.3 Write a MongoDB query to display the count in each course where obtained course gradeas ‘RA’ from the collection.

2. Create a restaurant database with name, complete address with pin code, item/menu list with price. The collection should contain 20 documents with data of minimum 3 repeated restaurant name in different location.

```
[
  {
    "name": "The Burger Place",
    "address": {
      "street": "123 Main St",
      "city": "New York",
      "state": "NY",
      "zip": "10001"
    },
    "menu": [
      {
        "item": "Hamburger",
        "price": 9.99
      },
      {
        "item": "Cheeseburger",
        "price": 10.99
      },
        {
          "item": "French Fries",
          "price": 3.99
        },
        {
          "item": "Onion Rings",
          "price": 4.99
        }
      ]
    },
    {
      "name": "Pizza Palace",
      "address": {
        "street": "456 Main St",
        "city": "Los Angeles",
        "state": "CA",
        "zip": "90001"
      },
      "menu": [
        {
          "item": "Margherita Pizza",
          "price": 12.99
        },
        {
          "item": "Pepperoni Pizza",
          "price": 14.99
        },
        {
          "item": "Garlic Bread",
          "price": 4.99
        },
        {
          "item": "Caesar Salad",
          "price": 8.99
        }
      ]
    },
    {
      "name": "Sushi House",
      "address": {
        "street": "789 Main St",
        "city": "San Francisco",
        "state": "CA",
        "zip": "94101"
      },
      "menu": [
        {
          "item": "California Roll",
          "price": 9.99
        },
    {
      "item": "Spicy Tuna Roll",
      "price": 11.99
    },
    {
      "item": "Salmon Nigiri",
      "price": 3.99
    },
    {
      "item": "Miso Soup",
      "price": 2.99
    }
  ]
},
{
  "name": "The Burger Place",
  "address": {
    "street": "101 Park Ave",
    "city": "New York",
    "state": "NY",
    "zip": "10017"
  },
  "menu": [
    {
      "item": "Hamburger",
      "price": 9.99
    },
    {
      "item": "Cheeseburger",
      "price": 10.99
    },
    {
      "item": "French Fries",
      "price": 3.99
    },
    {
      "item": "Onion Rings",
      "price": 4.99
    }
  ]
},
{
  "name": "Taco Time",
  "address": {
    "street": "1111 3rd Ave",
    "city": "Seattle",
    "state": "WA",
    "zip": "98101"
  },
  "menu": [
      {
        "item": "Beef Taco",
        "price": 2.99
      },
      {
        "item": "Chicken Taco",
        "price": 3.99
      },
      {
        "item": "Fish Taco",
        "price": 4.99
      },
      {
        "item": "Guacamole",
        "price": 1.99
      }
    ]
  },
  {
    "name": "The Burger Place",
    "
```

2.1 Write a MongoDB query to display the next 5 restaurants after skipping first 3 which have same location.

```
db.restaurants.find({}).sort({"address.city": 1, "name": 1}).skip(3).limit(5)
```

- skip() method to skip the first 3 restaurants that have the same location, and the limit() method to limit the results to the next 5 restaurants.

2.2 Write a MongoDB query to find the restaurant Id, name for those restaurants which have menu name 'dosa' in it.

```
db.restaurants.find({"menu.item": "dosa"}, {"_id": 1, "name": 1})
```

3. MongoDB Project on Building an Online Radio Station App with MongoDB, Express, and Node.js.

Create a new Node.js project and install the necessary dependencies:

shell

```
$ mkdir online-radio

$ cd online-radio

$ npm init -y

$ npm install express mongoose body-parser cors
```

Next, let's create a file called app.js and set up our Express app:

Javascript

Copy code

```
const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();

app.use(bodyParser.json());

app.use(cors());

mongoose.connect('mongodb://localhost/online-radio', { useNewUrlParser: true,
useUnifiedTopology: true });

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function() {

  console.log('Connected to MongoDB');

});

app.listen(3000, () => {

  console.log('Server started on port 3000');

});
```

We're using the body-parser middleware to parse incoming requests with JSON payloads, and the cors middleware to enable cross-origin resource sharing. We're also connecting to a local MongoDB database called online-radio.

Now, let's create a model for our radio stations. Create a new file called models/station.js:

javascript

```
const mongoose = require('mongoose');

const stationSchema = new mongoose.Schema({

  name: String,

  description: String,

  imageUrl: String,

  streamUrl: String,

  genre: String,
  location: String,
  website: String
});
module.exports = mongoose.model('Station', stationSchema);
```

We're defining a Station schema with fields for the station's name, description, image URL, stream URL, genre, location, and website. We're exporting a Station model based on this schema.

Next, let's create our API routes for creating, reading, updating, and deleting stations. Create a new file called routes/stations.js:

javascript

```javascript
const express = require('express');
const router = express.Router();
const Station = require('../models/station');

// GET all stations
router.get('/', (req, res) => {
  Station.find({}, (err, stations) => {
    if (err) return res.status(500).json({ error: err });
    res.json(stations);
  });
});

// GET a specific station
router.get('/:id', (req, res) => {
  Station.findById(req.params.id, (err, station) => {
    if (err) return res.status(500).json({ error: err });
    if (!station) return res.status(404).json({ error: 'Station not found' });
    res.json(station);
  });
});
```

```
// POST a new station
router.post('/', (req, res) => {
  const station = new Station(req.body);
  station.save((err, savedStation) => {
    if (err) return res.status(500).json({ error: err });
    res.status(201).json(savedStation);
  });
});


// PUT/update an existing station
router.put('/:id', (req, res) => {
  Station.findByIdAndUpdate(req.params.id, req.body, { new: true }, (err, updatedStation) =>
{
    if (err) return res.status(500).json({ error: err });
    if (!updatedStation) return res.status(404).json({ error: 'Station not found' });
    res.json(updatedStation);
  }); });
```

4. Demonstrate single field indexing (create, find & drop) inMongoDBfor countries population database.

Let's assume we have a collection named "countries" with the following schema

```
{
  "name": "string",
  "population": "number"
}
```

To **create an index** on the "population" field, we can use the following command in the MongoDB shell:

```
db.countries.createIndex({"population": 1})
```

To **find** documents using the index, we can use the following query:

```
db.countries.find({"population": {$gt: 100000000}})
```

To **drop** the index, we can use the following command:

db.countries.dropIndex({"population": 1})

5. Build MongoDB cluster using MongoAtlas.

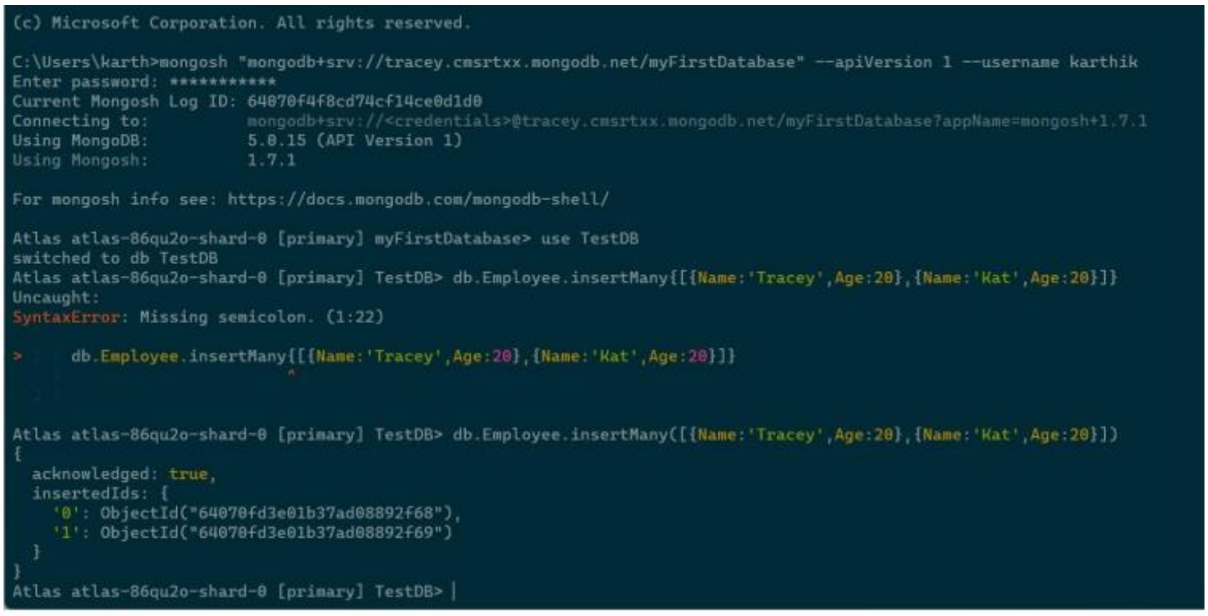
Step-1: Create a cluster in MongDO Atlas by signing in to it.

Step-2: Connect the mongoDB Atlas with mongDB shell using the provided connection string.

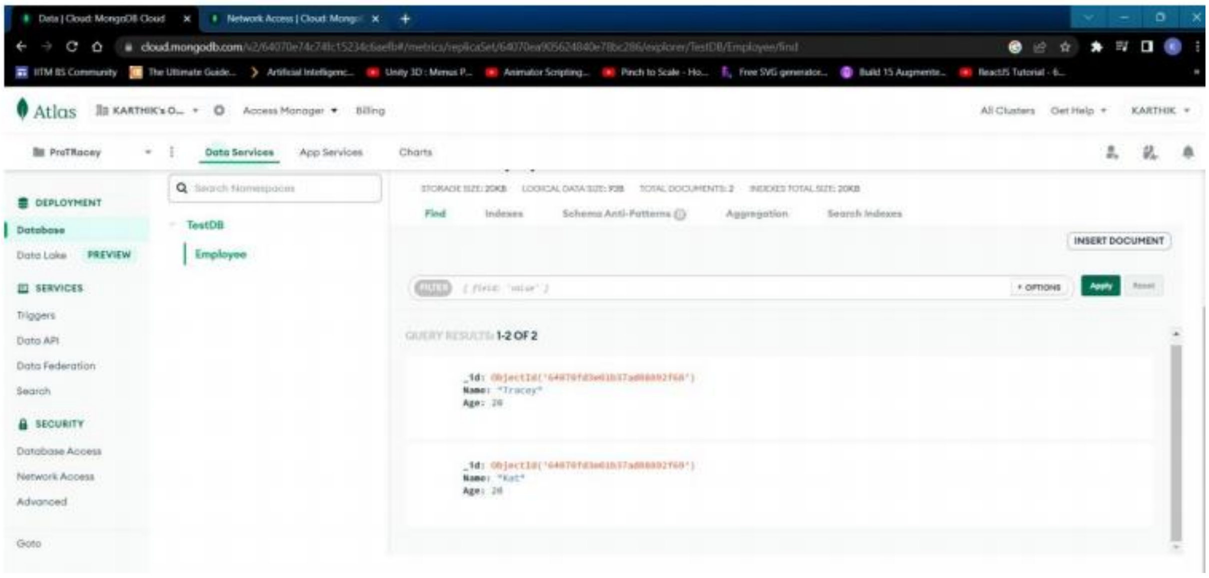
Step-3: After making the connection in the shell create a database in the cluster.

Step-4: Create collections in the database.

Step-5: Insert and retrieve document from the collections.



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\karth>mongosh "mongodb+srv://tracey.cmsrtxx.mongodb.net/myFirstDatabase" --apiVersion 1 --username karthik
Enter password: ***********
Current Mongosh Log ID: 64070f4f8cd74cf14ce0d1d0
Connecting to:          mongodb+srv://<credentials>@tracey.cmsrtxx.mongodb.net/myFirstDatabase?appName=mongosh+1.7.1
Using MongoDB:          5.0.15 (API Version 1)
Using Mongosh:          1.7.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-86qu2o-shard-0 [primary] myFirstDatabase> use TestDB
switched to db TestDB
Atlas atlas-86qu2o-shard-0 [primary] TestDB> db.Employee.insertMany{[{Name:'Tracey',Age:20},{Name:'Kat',Age:20}]}
Uncaught:
SyntaxError: Missing semicolon. (1:22)

>     db.Employee.insertMany{[{Name:'Tracey',Age:20},{Name:'Kat',Age:20}]}
                            ^


Atlas atlas-86qu2o-shard-0 [primary] TestDB> db.Employee.insertMany([{Name:'Tracey',Age:20},{Name:'Kat',Age:20}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64070fd3e01b37ad08892f68"),
    '1': ObjectId("64070fd3e01b37ad08892f69")
  }
}
Atlas atlas-86qu2o-shard-0 [primary] TestDB> |
```



STORAGE SIZE: 20KB   LOGICAL DATA SIZE: 93B   TOTAL DOCUMENTS: 2   INDEXES TOTAL SIZE: 20KB

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('64070fd3e01b37ad08892f68')
Name: "Tracey"
Age: 20

_id: ObjectId('64070fd3e01b37ad08892f69')
Name: "Kat"
Age: 20
```

6. WriteAngularJs code to display book detail with the availability and issued status using appropriate AngularJs expression and directives.

HTML code

```
<!-- <!DOCTYPE html>
<html ng-app="myApp">
<head>
	<title>Book Details</title>
	<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
	<script src="app.js"></script>
</head>
<body ng-controller="bookCtrl">

	<ul>
		<li ng-repeat="b in books">
			{{b.title}} - {{b.available ? 'Available' : 'Not Available'}} (Issued:
{{b.issued ? 'Yes' : 'No'}})
		</li>
	</ul>
</body>
</html> -->
```

ANGULAR JS code:

```
// var app = angular.module("myApp", []);
// app.controller("bookCtrl", function($scope) {
// 	$scope.books = [
// 		{
// 			title: "To Kill a Mockingbird",
// 			available: true,
// 			issued: false
// 		},
// 		{
// 			title: "1984",
// 			available: false,
// 			issued: true
// 		},
// 		{
// 			title: "The Catcher in the Rye",
// 			available: true,
// 			issued: false
// 		},
// 		{
// 			title: "Pride and Prejudice",
// 			available: false,
// 			issued: true
// 		},
//          {
//                  title: "The Lord of the Rings",
//                  available: true,
//                  issued: false
//          }
//      ];
// });
```

7. Create TODO list page for current date using AngularJs

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

app.controller("todoCtrl", function($scope) {

        $scope.today = new Date();

        $scope.todos = [

                {text: "Finish project"},

                {text: "Go for a walk"},

                {text: "Buy groceries"},

                {text: "Call mom"},

                {text: "Read a book"}

        ];

        $scope.addTodo = function() {

                $scope.todos.push({text:$scope.newTodo});

                $scope.newTodo = "";

        };

        $scope.remove = function(index) {

                $scope.todos.splice(index, 1);

        };

});
```

8. Build SPA of cricket score board using AngularJs.

To begin, you'll need to create a new Angular project. You can do this using the Angular CLI by running the following command in your terminal:

```
ng new cricket-scoreboard
```

Next, you'll need to create a new component for the scoreboard. You can do this using the Angular CLI by running the following command in your terminal:

```
ng generate component scoreboard
```

This will create a new component called scoreboard and generate some files for you. You can find these files in the src/app/scoreboard directory.

In the scoreboard.component.ts file, you can define the logic for your scoreboard. You can define variables for the current score, wickets, overs, and other relevant information.

typescript

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

@Component({
  selector: 'app-scoreboard',
  templateUrl: './scoreboard.component.html',
  styleUrls: ['./scoreboard.component.css']
})

export class ScoreboardComponent {
  currentScore: number = 0;
  wickets: number = 0;
  overs: number = 0;
}
```

In the scoreboard.component.html file, you can define the markup for your scoreboard. You can use Angular's data binding syntax to display the current score, wickets, overs, and other relevant information.

css

```
<div>
  <h2>Cricket Scoreboard</h2>
  <p>Current Score: {{ currentScore }}</p>
  <p>Wickets: {{ wickets }}</p>
  <p>Overs: {{ overs }}</p>
</div>
```

Once you've defined your component logic and markup, you can use it in your main app.component.html file. You can add the scoreboard component to the markup by using its selector.

php

Copy code

```
<div>
  <app-scoreboard></app-scoreboard>
</div>
```

Finally, you can run your application using the Angular CLI by running the following command in your terminal:

```
ng serve
```

This will start a development server and open your application in your default web browser. You should see the cricket scoreboard that you defined in your scoreboard.component.html file.

9. Demonstrate Http sevice using AngularJs.

PROCEDURE:

Step-1: Create a html page

Step-2: Include the angular js cdn

Step-3: create a app variable using module().

Step-4: Use $http.get() to receive the data from the server. On successful retrieval display

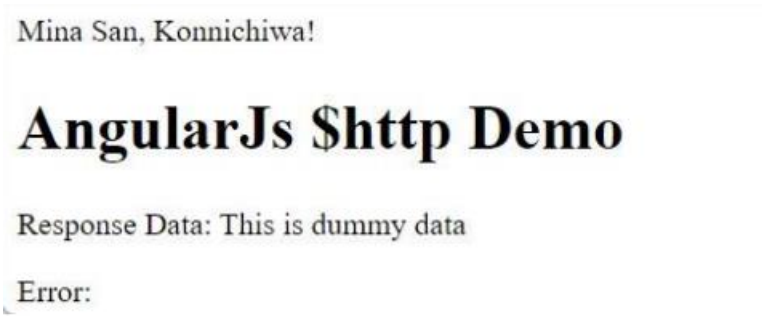
the data. Else display the error.

PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app ="myApp">
<h1>AngularJS $http Demo: </h1>
<div>
<div ng-controller="myController">
Response Data: {{data}} <br />
Error: {{error}}
</div>
</div>
<script>
var myApp = angular.module('myApp', []);
myApp.controller("myController", function ($scope, $http) {
var onSuccess = function (data, status, headers, config) {
```

```
$scope.data = data;
};
var onError = function (data, status, headers, config) {
$scope.error = status;
}
var promise = $http.get("/demo/getdata");
promise.success(onSuccess);
promise.error(onError);
});
</script>
</body>
</html>
```

OUTPUT:



```
Mina San, Konnichiwa!

AngularJs $http Demo

Response Data: This is dummy data

Error:
```

10. Create and import your own module in Nodejs.

PROCEDURE:

To include a module, use the require() function with the name of the module:

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

Now your application has access to the HTTP module, and is able to create a server:

```
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.end('Hello World!');
}).listen(8080);
```

Create Your Own Modules
You can create your own modules, and easily include them in your applications.
The following example creates a module that returns a date and time object:
Create a module that returns the current date and time:

```
exports.myDateTime = function () {
return Date();
};
```

Use the exports keyword to make properties and methods available outside the module file.

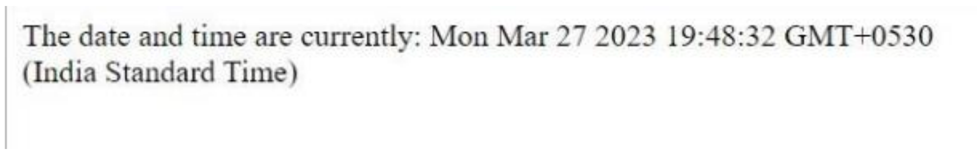
Save the code above in a file called "myfirstmodule.js"

Include Your Own Module
Now you can include and use the module in any of your Node.js files.
Use the module "myfirstmodule" in a Node.js file

PROGRAM:

```
myModule.js
exports.myDateTime = function () {
return Date();
};
myFirstModule.js
var http = require('http');
var dt = require('./myModule');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.write("The date and time are currently: " + dt.myDateTime());
res.end();
}).listen(8080);
```

OUTPUT:



```
The date and time are currently: Mon Mar 27 2023 19:48:32 GMT+0530
(India Standard Time)
```

11. Create a file with the following content

"India is my country and all Indians are my brothers and sisters. I love my country and I am proud of its rich and varied heritage. I shall always strive to be worthy of it. "

Using appropriate node file system method add the following line in the file content

"I shall give respect to my parents, teachers and elders and treat everyone with courtesy."

```
const fs = require('fs');
const filePath = 'content.txt';
fs.readFile(filePath, 'utf8', (err, data) => {
if (err) throw err;
const updatedContents = `${data}\nI shall give respect to my parents, teachers and
elders and treat everyone with courtesy.`;
fs.writeFile(filePath, updatedContents, 'utf8', (err) => {
if (err) throw err;
console.log('File updated successfully.');
console.log(data);
});
});
```

12. Create book details database in MongoDB with 5 documents and order the document based on name of the book.

create a new database called "book_details" using the following command in the MongoDB shell

```
use book_details
```

Next, let's create a new collection called "books" using the following command

```
db.createCollection("books")
```

Now, let's insert 5 documents into the "books" collection using the insertMany command:

```
db.books.insertMany([
  { name: "The Great Gatsby", author: "F. Scott Fitzgerald", year: 1925 },
  { name: "To Kill a Mockingbird", author: "Harper Lee", year: 1960 },
  { name: "Pride and Prejudice", author: "Jane Austen", year: 1813 },
  { name: "1984", author: "George Orwell", year: 1949 },
  { name: "The Catcher in the Rye", author: "J.D. Salinger", year: 1951 }
])
```

Finally, let's order the documents in the "books" collection based on the name of the book using the sort method:

```
db.books.find().sort({ name: 1 })
```

13. Create an employee personal detail file and an employee project detail file. Merge the content of those files and save it in new file using file system interactions function ofNodeJs.

create the employee personal detail file named "employee_personal.json" with the following content

```
{
  "name": "John Doe",
  "age": 30,
  "email": "johndoe@example.com",
  "address": "123 Main St, Anytown USA"
}
```

We can create the file using the `fs.writeFile` method:

```
const fs = require('fs')

const personalDetails = {
  name: 'John Doe',
  age: 30,
  email: 'johndoe@example.com',
  address: '123 Main St, Anytown USA'
}

fs.writeFile('employee_personal.json', JSON.stringify(personalDetails), err => {
  if (err) throw err
  console.log('Employee personal detail file created')
})
```

Next, let's create the employee project detail file named "employee_project.json" with the following content:

```
{
  "project_name": "Acme Corp Website",
  "project_description": "Develop a new website for Acme Corp",
  "start_date": "2022-01-01",
  "end_date": "2022-06-30",
  "status": "In progress"
}
```

We can create the file using the `fs.writeFile` method:

```
const projectDetails = {
  project_name: 'Acme Corp Website',
  project_description: 'Develop a new website for Acme Corp',
  start_date: '2022-01-01',
  end_date: '2022-06-30',
  status: 'In progress'
}

fs.writeFile('employee_project.json', JSON.stringify(projectDetails), err => {
  if (err) throw err
  console.log('Employee project detail file created')
})
```

Now, let's merge the content of those files and save it in a new file named "employee_details.json" using the `fs.readFile` and `fs.appendFile` methods:

```
fs.readFile('employee_personal.json', 'utf8', (err, personalData) => {
 if (err) throw err
 fs.readFile('employee_project.json', 'utf8', (err, projectData) => {
  if (err) throw err
  const employeeDetails = { ...JSON.parse(personalData), ...JSON.parse(projectData) }
  fs.appendFile('employee_details.json', JSON.stringify(employeeDetails), err => {
   if (err) throw err
   console.log('Employee details file created')
  })
 })
})
```

14. Demonstrate ExpressJs scaffolding.

PROCEDURE:

Step-1: Create a directory name myApp. Create a file name app.js in the myApp directory having the below mentioned program.

Step-2: Open Node.js command prompt, go to myApp and run init command

Step-3: Fill the and entries and press enter

Step-4: It will create a package.json file in myApp folder and the data is show in JSON format.

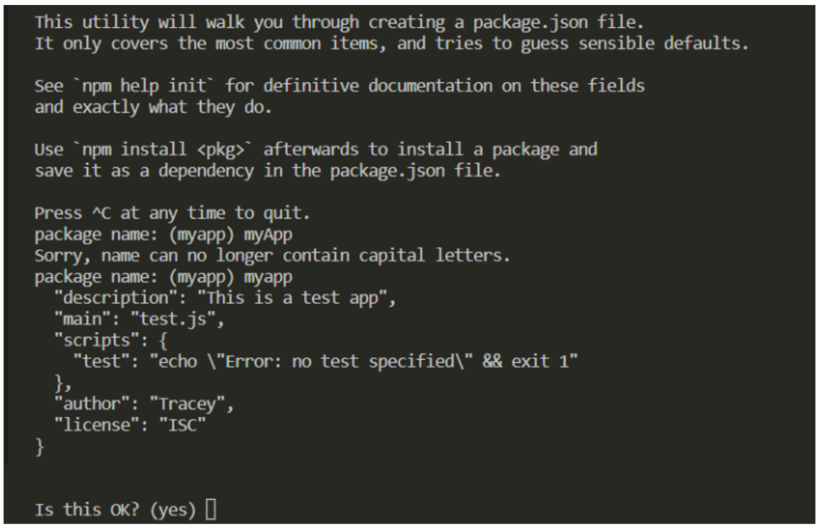
PROGRAM:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send('Welcome to JavaTpoint!');
});
app.listen(8000, function () {
```

console.log('Example app listening on port 8000!');

});

OUTPUT:



```
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 init` for definitive documentation on these fields
and exactly what they do.

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

Press ^C at any time to quit.
package name: (myapp) myApp
Sorry, name can no longer contain capital letters.
package name: (myapp) myapp
  "description": "This is a test app",
  "main": "test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Tracey",
  "license": "ISC"
}


Is this OK? (yes) 
```

15. Build a Simple CRUD Application with Express and MongoDB with suitable Database.

1. Set up your environment: Install **Node.js** and **MongoDB** on your computer, and create a new directory for your project. Open the directory in your terminal and run **npm init** to create a new **package.json** file.
2. Install dependencies: Install the **express**, **mongoose**, and **body-parser** packages using the command **npm install express mongoose body-parser**.
3. Create a new file called **server.js**, and import the necessary modules:

```
const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');


const app = express();

const port = 3000;
```

4. Set up middleware: Use the **body-parser** middleware to parse incoming request bodies in JSON format:

```
app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());
```

5. Connect to MongoDB: Use **mongoose** to connect to your MongoDB database. Replace **<username>**, **<password>**, **<cluster-url>**, and **<database-name>** with your own values.

```
mongoose.connect(`mongodb+srv://<username>:<password>@<cluster-url>/<database-name>?retryWrites=true&w=majority`, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => {
  console.log('Connected to MongoDB!');
}).catch((err) => {
  console.error(err);
});
```

6. Define a Mongoose schema: Create a new file called **model.js**, and define a Mongoose schema for your data. Here's an example schema for a **Task**:

```
const mongoose = require('mongoose');

const TaskSchema = new mongoose.Schema({
  title: String,
  description: String,
  completed: Boolean
});

module.exports = mongoose.model('Task', TaskSchema);
```

16. Create application front page for Bi-cycle models using AngularJs.

```
ng new bicycle-models-app
```

Next, create a new component for the front page using the Angular CLI:

```
ng generate component front-page
```

In the `front-page.component.html` file, add HTML markup for the front page:

```
<div class="container">
```

```
 <h1>Bicycle Models</h1>
 <p>Welcome to our bicycle models application. Browse our selection of bicycles and find the
perfect one for you.</p>
 <div class="row">
  <div class="col-md-4">
   <div class="card mb-4">
    <img class="card-img-top" src="https://via.placeholder.com/350x200" alt="Card image cap">
    <div class="card-body">
     <h5 class="card-title">Mountain Bikes</h5>
     <p class="card-text">Explore our selection of mountain bikes, perfect for off-road
adventures.</p>
     <a href="#" class="btn btn-primary">View Models</a>
    </div>
   </div>
  </div>
  <div class="col-md-4">
   <div class="card mb-4">
    <img class="card-img-top" src="https://via.placeholder.com/350x200" alt="Card image cap">
    <div class="card-body">
     <h5 class="card-title">Road Bikes</h5>
     <p class="card-text">Check out our collection of road bikes, designed for speed and
performance.</p>
     <a href="#" class="btn btn-primary">View Models</a>
    </div>
   </div>
  </div>
  <div class="col-md-4">
   <div class="card mb-4">
    <img class="card-img-top" src="https://via.placeholder.com/350x200" alt="Card image cap">
    <div class="card-body">
     <h5 class="card-title">City Bikes</h5>
     <p class="card-text">Find your perfect city bike, ideal for commuting or leisurely rides.</p>
```

```
          <a href="#" class="btn btn-primary">View Models</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

This HTML code creates a simple front page with a title, introductory text, and three cards that showcase different types of bicycles available in the application.

In the `front-page.component.css` file, add styling for the front page:

```
.container {
  max-width: 960px;
  margin: 0 auto;
  padding: 40px 15px;
}
.card {
  height: 100%;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
  transition: box-shadow 0.3s ease;
}
.card:hover {
  box-shadow: 0 8px 16px 0 rgba(0, 0, 0, 0.2);
}
.card-img-top {
  height: 200px;
  object-fit: cover;
}
```