

**CAPSTONE I (CS 392)**  
**Lab Report**  
**B.Tech. (CSE/ DS/CYS/iMSC)**  
**Semester II**  
**L-T-P-C: 2-0-4-4**

**Academic Year: 2024-25**



**DEPARTMENT OF COMPUTER SCIENCE AND  
ENGINEERING**

Student ID	BT22GDS056
Name	Agnishwar Raychaudhuri

**NIIT UNIVERSITY, NEEMRANA**

NH-8, Delhi-Jaipur Highway,  
Neemrana, District Alwar (Rajasthan),  
Pin-301705

Website: [www.niituniversity.in](http://www.niituniversity.in)

## **List of Experiments**

<b>Practical No.</b>	<b>Aim of Practical</b>	<b>COs</b>	<b>Page No.</b>
1	Collaboration Tool-Git	CO2	1
2	HTML and CSS	CO1	2
3	Javascript	CO1	4
4	BootStrap	CO1	5
5	NodeJS	CO3	6
6	ExpressJS	CO3	6
7	MongoDB	CO3	7
8	Database Connectivity	CO3	25
9	Session and Cookies Demo	CO3	28
10	Login Exercise	CO3	30
11	Setting up DevOPS for Project	CO4 CO5	31
12	APIs	CO4 CO5	34

## Practical 1: Collaboration Tool-Git:

### Steps of Execution:

1. Team lead will make Github repository and will send invite to other team members.
2. Team lead will create local repo and then push to remote repo
3. Team members will accept the invite and clone repo on their local machine.
4. All the team members will share file using git / github
5. Team members will update their repository and check the contents shared by others.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*      Repository name \*

 agnishwarr / LabManual LabManual is available.

Great repository names are short and memorable. Need inspiration? How about [crispy-barnacle](#) ?

Description (optional)

 Public      Anyone on the internet can see this repository. You choose who can commit.  
  Private      You choose who can see and commit to this repository.

Initialize this repository with:

Add people to LabManual

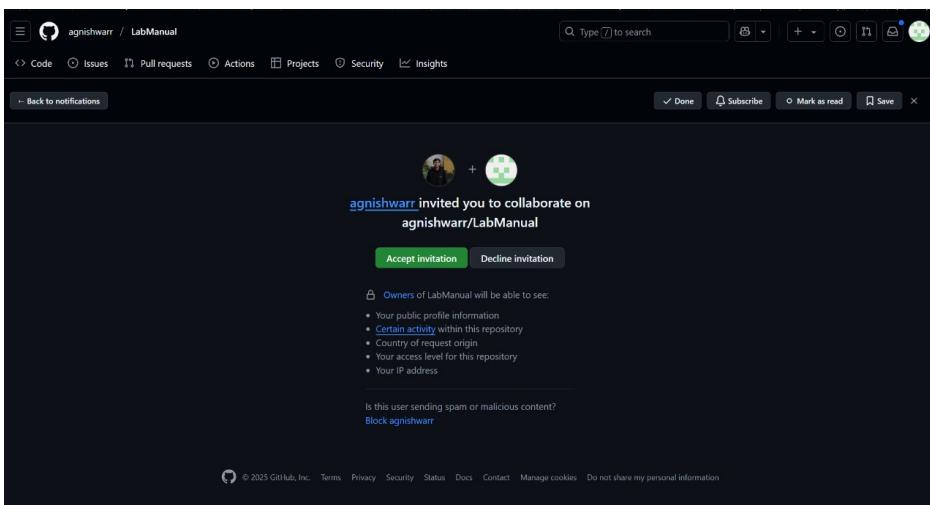
 Khushi Singh  
Khushi07S X

Cancel Add Khushi07S

Add people

```
echo "# LabManual" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/agnishwarr/LabManual.git
git push -u origin main
```

```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> echo "# LabManual" >> README.md
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git init
Initialized empty Git repository in C:/Users/Augni/Desktop/NIIT/Academics/SEM VI/Capstone/Lab Manual/.git/
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git add README.md
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git commit -m "first commit"
[master (root-commit) 5c27bc3] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git branch -M main
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git remote add origin https://github.com/agnishwarr/LabManual.git
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 251 bytes | 125.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/agnishwarr/LabManual.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual>
```



agnishwarr / LabManual

Type / to search

Code Issues Pull requests Actions Projects Security Insights

Back to notifications Done Subscribe Mark as read Save

agnishwarr invited you to collaborate on agnishwarr/LabManual

Accept invitation Decline invitation

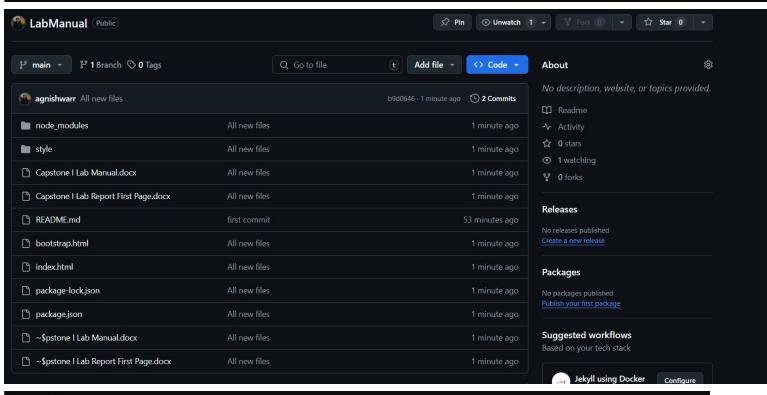
Owners of LabManual will be able to see:

- Your public profile information
- Certain activity within this repository
- Country of request origin
- Your access level for this repository
- Your IP address

Is this user sending spam or malicious content? Block agnishwarr

© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> git add .
warning: in the working copy of 'node_modules/.bin/node', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'node_modules/.bin/node.ps1', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'node_modules/.package-lock.json', LF will be replaced by CRLF the next time Git touches it
```



LabManual Public

main Branch 0 Tags

Add file Code

agnishwarr All files 5d0d546 · 1 minute ago 2 Commits

node\_modules All new files 1 minute ago

style All new files 1 minute ago

Capstone I Lab Manual.docx All new files 1 minute ago

Capstone I Lab Report First Page.docx All new files 1 minute ago

README.md first commit 53 minutes ago

bootstrap.html All new files 1 minute ago

index.html All new files 1 minute ago

package-lock.json All new files 1 minute ago

package.json All new files 1 minute ago

S-Postone I Lab Manual.docx All new files 1 minute ago

- S-Postone I Lab Report First Page.docx All new files 1 minute ago

About

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks

Releases

No releases published Create a new release

Packages

No packages published Publish your first package

Jekyll using Docker Configure

```
F:\>cd University/Capstone
F:\University\Capstone>git clone https://github.com/agnishwarr/LabManual.git
Cloning into 'LabManual'...
remote: Enumerating objects: 396, done.
remote: Counting objects: 100% (396/396), done.
remote: Compressing objects: 100% (271/271), done.
remote: Total 396 (delta 71), reused 395 (delta 70), pack-reused 0 (from 0)
Receiving objects: 100% (396/396), 35.51 MiB | 1.51 MiB/s, done.
Resolving deltas: 100% (71/71), done.
Updating files: 100% (358/358), done.
```

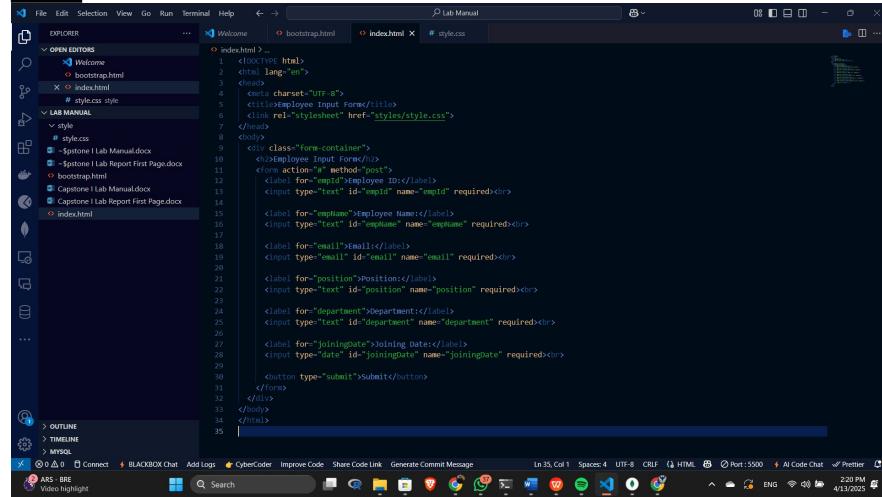
## Practical 2: HTML and CSS:

### Steps of Execution:

1. Make an input form for an employee.

## 2. Apply the relevant CSS using external CSS file

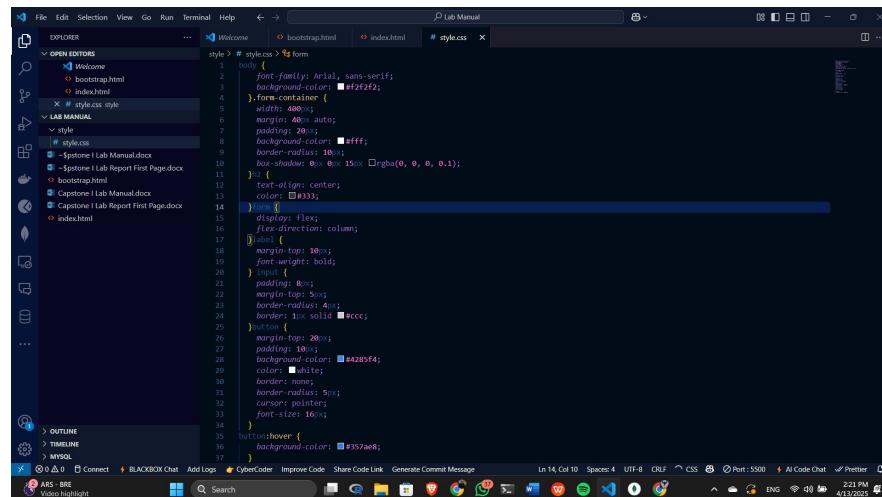
### Code



```

<!DOCTYPE html> ...
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Employee Input Form</title>
    <link rel="stylesheet" href="styles/style.css">
</head>
<body>
    <div class="form-container">
        <form action="#" method="post">
            <label for="employeeId">Employee ID:</label>
            <input type="text" id="employeeId" name="employeeId" required=<br>>
            <label for="employeeName">Employee Name:</label>
            <input type="text" id="employeeName" name="employeeName" required=<br>>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required=<br>>
            <label for="position">Position:</label>
            <input type="text" id="position" name="position" required=<br>>
            <label for="department">Department:</label>
            <input type="text" id="department" name="department" required=<br>>
            <label for="joiningDate">Joining Date:</label>
            <input type="date" id="joiningDate" name="joiningDate" required=<br>>
            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>

```



```

body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
}
.form-container {
    width: 400px;
    margin: auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0px 0px 15px #rgba(0, 0, 0, 0.1);
}
input {
    text-align: center;
    color: #333;
}
input {
    display: flex;
    flex-direction: column;
}
label {
    margin-top: 10px;
    font-weight: bold;
}
input {
    padding: 8px;
    margin-top: 5px;
    border-radius: 4px;
    border: 1px solid #ccc;
}
button {
    margin-top: 20px;
    padding: 10px;
    background-color: #4a285f4;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
}
button:hover {
    background-color: #357ae8;
}

```

### Output

#### Employee Input Form

Employee ID:

Employee Name:

Email:

Position:

Department:

Joining Date:

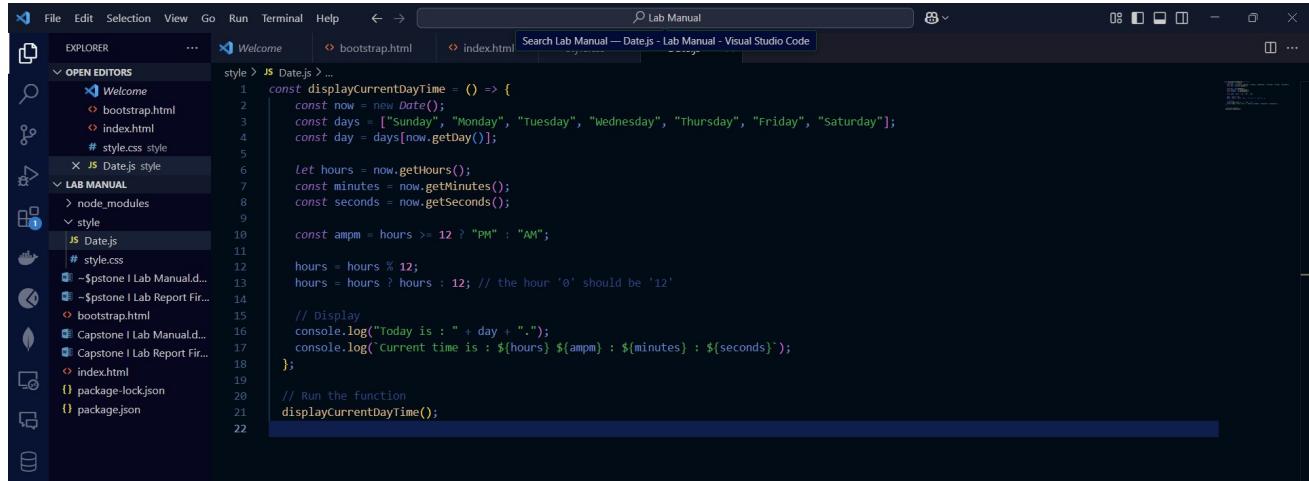
### Practical 3: Javascript:

Write a JavaScript program to display the current day and time in the following format.

Today is : Tuesday.

Current time is : 10 PM : 30 : 38

#### Code:



```

File Edit Selection View Go Run Terminal Help < - > Lab Manual
EXPLORER bootstrap.html index.html Search Lab Manual — Date.js - Lab Manual - Visual Studio Code
OPEN EDITORS style > JS Date.js > ...
Welcome
bootstrap.html
index.html
style.css
LAB MANUAL
node_modules
style
Date.js
style.css
Capstone I Lab Manual.d...
Capstone I Lab Report Fir...
bootstrap.html
Capstone I Lab Manual.d...
Capstone I Lab Report Fir...
index.html
package-lock.json
package.json

```

```

const displayCurrentDayTime = () => {
  const now = new Date();
  const days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
  const day = days[now.getDay()];

  let hours = now.getHours();
  const minutes = now.getMinutes();
  const seconds = now.getSeconds();

  const ampm = hours >= 12 ? "PM" : "AM";

  hours = hours % 12;
  hours = hours + hours : 12; // the hour '0' should be '12'

  // Display
  console.log("Today is : " + day + ".");
  console.log(`Current time is : ${hours} ${ampm} : ${minutes} : ${seconds}`);
};

// Run the function
displayCurrentDayTime();

```

Install the dependencies

```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> npm init -y
Wrote to C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual\package.json:
```

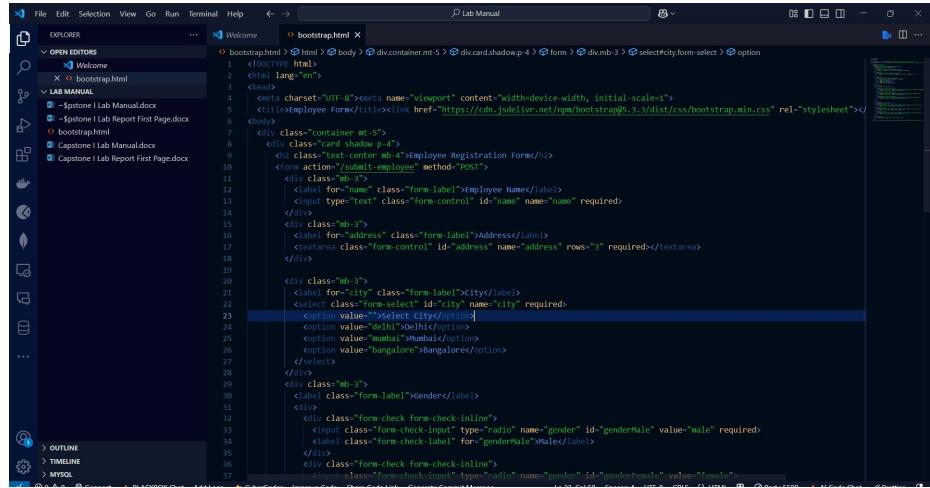
```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual>
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> npm i node
```

#### Output:

```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> cd style
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual\style> node Date.js
Today is : Monday.
Current time is : 10 AM : 58 : 51
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual\style> []
```

## Practical 4: BootStrap :

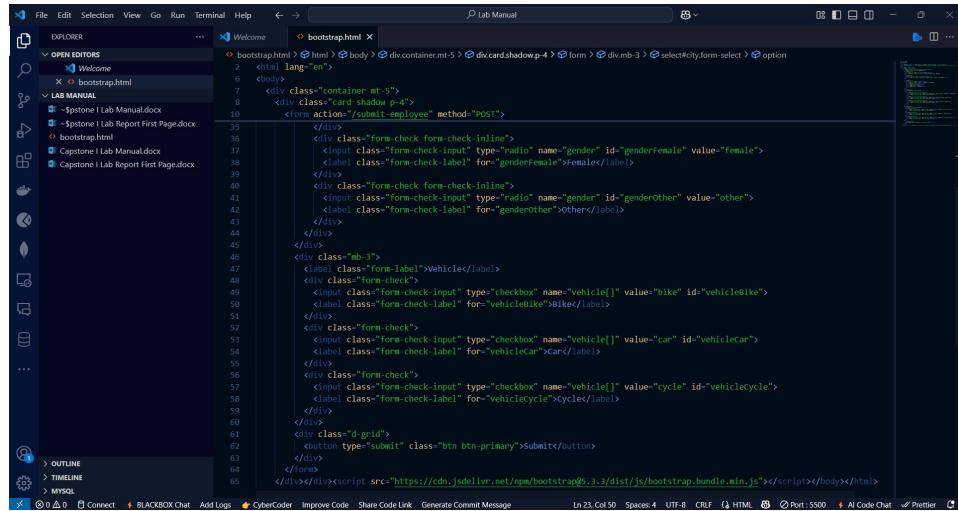
### Code



```

<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Employee Form</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <div class="card shadow p-4">
            <form action="/submit-employee" method="POST">
                <div class="mb-3">
                    <label for="name" class="form-label">Employee Name</label>
                    <input type="text" class="form-control" id="name" name="name" required>
                </div>
                <div class="mb-3">
                    <label for="address" class="form-label">Address</label>
                    <textarea class="form-control" id="address" name="address" rows="3" required></textarea>
                </div>
                <div class="mb-3">
                    <label for="city" class="form-label">City</label>
                    <select class="form-select" id="city" name="city" required>
                        <option value="">Select City</option>
                        <option value="Delhi">Delhi</option>
                        <option value="Mumbai">Mumbai</option>
                        <option value="Bangalore">Bangalore</option>
                    </select>
                </div>
                <div class="mb-3">
                    <label class="form-label">Gender</label>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio" name="gender" id="genderMale" value="male" required>
                        <label class="form-check-label" for="genderMale">Male</label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input type="radio" name="gender" id="genderFemale" value="female" checked="checked">
                        <label class="form-check-label" for="genderFemale">Female</label>
                    </div>
                </div>
            </form>
        </div>
    </div>
</body>

```

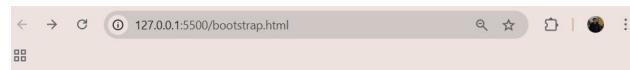


```

<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Employee Form</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <div class="card shadow p-4">
            <form action="/submit-employee" method="POST">
                <div class="mb-3">
                    <label class="form-label">Vehicle</label>
                    <div class="form-check">
                        <input class="form-check-input" type="checkbox" name="vehicle[1]" value="bike" id="vehicleBike">
                        <label class="form-check-label" for="vehicleBike">Bike</label>
                    </div>
                    <div class="form-check">
                        <input class="form-check-input" type="checkbox" name="vehicle[1]" value="car" id="vehicleCar">
                        <label class="form-check-label" for="vehicleCar">Car</label>
                    </div>
                    <div class="form-check">
                        <input class="form-check-input" type="checkbox" name="vehicle[1]" value="cycle" id="vehicleCycle">
                        <label class="form-check-label" for="vehicleCycle">Cycle</label>
                    </div>
                </div>
                <div class="d-grid">
                    <button type="submit" class="btn btn-primary">Submit</button>
                </div>
            </form>
        </div>
    </div>
</body>

```

### Output



**Employee Registration Form**

Employee Name

Address

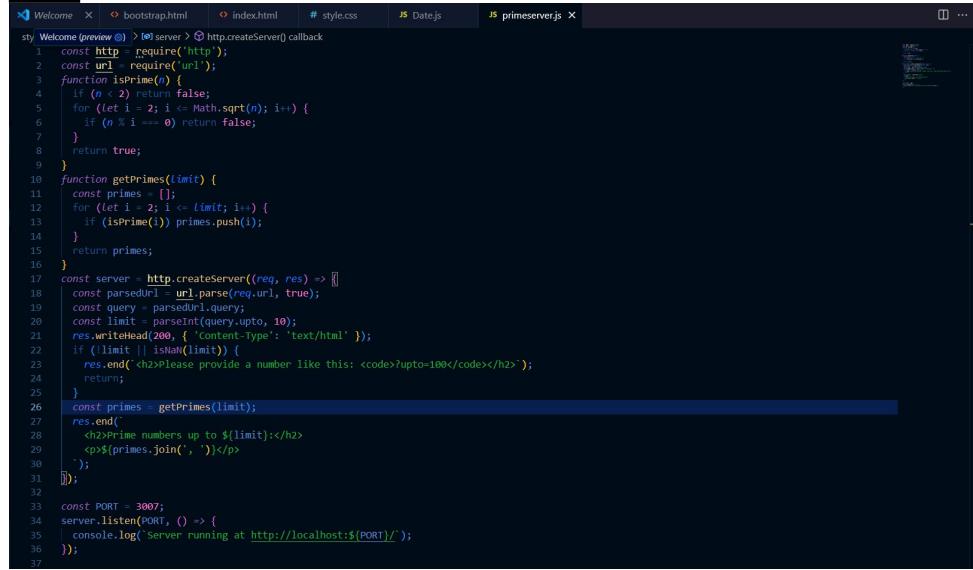
City

Gender  
 Male  Female  Other

Vehicle  
 Bike  Car  Cycle

## Practical 5: NodeJS :

Write NodeJS code using HTTP module  
Code

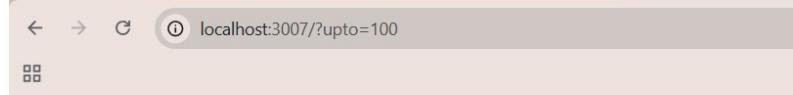


```

1 const http = require('http');
2 const url = require('url');
3 function isPrime(n) {
4     if (n < 2) return false;
5     for (let i = 2; i <= Math.sqrt(n); i++) {
6         if (n % i === 0) return false;
7     }
8     return true;
9 }
10 function getPrimes(upto) {
11     const primes = [];
12     for (let i = 2; i <= upto; i++) {
13         if (isPrime(i)) primes.push(i);
14     }
15     return primes;
16 }
17 const server = http.createServer((req, res) => {
18     const parsedUrl = url.parse(req.url, true);
19     const query = parsedUrl.query;
20     const limit = parseInt(query.upto, 10);
21     res.writeHead(200, { 'Content-Type': 'text/html' });
22     if (!limit || isNaN(limit)) {
23         res.end(`<h2>Please provide a number like this: <code>?upto=100</code></h2>`);
24         return;
25     }
26     const primes = getPrimes(limit);
27     res.end(
28         `<h2>Prime numbers up to ${limit}</h2>
29         <p>${primes.join(', ')}</p>
30     `);
31 });
32
33 const PORT = 3007;
34 server.listen(PORT, () => {
35     console.log(`Server running at http://localhost:\${PORT}/`);
36 });
37

```

## Output



localhost:3007/?upto=100

### Prime numbers up to 100:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

## Practical 6: ExpressJS :

Use middleware in ExpressJS  
Code

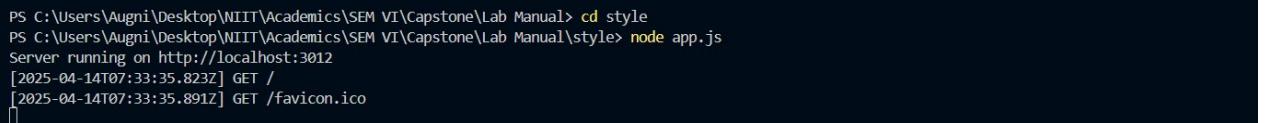


```

1 // app.js > [PORT]
2 const express = require('express');
3 const app = express();
4
5 // Custom Middleware
6 const loggerMiddleware = (req, res, next) => {
7     console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
8     next(); // Call next() to pass control to the next middleware or route handler
9 }
10
11 // Use the middleware
12 app.use(loggerMiddleware);
13
14 // Route
15 app.get('/', (req, res) => {
16     res.send('Hello, middleware!');
17 });
18
19 // Start the server
20 const PORT = 3012;
21 app.listen(PORT, () => {
22     console.log(`Server running on http://localhost:\${PORT}/`);
23 });
24

```

## Terminal



```

PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> cd style
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> node app.js
Server running on http://localhost:3012
[2025-04-14T07:33:35.823Z] GET /
[2025-04-14T07:33:35.891Z] GET /favicon.ico

```

## Output



Hello, Middleware!

## Practical 7: MongoDB :

### Structure of 'restaurants' collection:

```
{
  "address": {
    "building": "1007",
    "coord": [ -73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

1. Write a MongoDB query to display all the documents in the collection restaurants.

```
> db.restaurants.find();
<
```

2. Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.

```
> db.restaurants.find({}, {"restaurant_id" : 1, "name":1, "borough":1, "cuisine" :1});
<
```

3. Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine, but exclude the field \_id for all the documents in the collection restaurant.

```
> db.restaurant.find({}, {"restaurant_id" : 1, "name":1, "borough":1, "cuisine" :1, "_id":0});
<
```

4. Write a MongoDB query to display the fields restaurant\_id, name, borough and zip code, but exclude the field \_id for all the documents in the collection restaurant.

```
> db.restaurants.find({}, {"restaurant_id": 1, "name": 1, "borough": 1, "address zipcode": 1, "_id": 0});
```

5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.

```
> db.restaurants.find({"borough": "Bronx"});
```

6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

```
> db.restaurants.find({"borough": "Bronx"}).limit(5);
```

7. Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.

```
> db.restaurants.find({"borough": "Bronx"}).skip(5).limit(5);
```

8. Write a MongoDB query to find the restaurants who achieved a score more than 90.

```
> db.restaurants.find({ "grades.score": { $gt: 90 } })
```

9. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

```
> db.restaurants.find({ "grades.score": { $gt: 80, $lt: 100 } })
```

10. Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.

```
> db.restaurants.find({ "address.coord.1": { $lt: -95.754168 } })
```

11. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

```
> db.restaurants.find({
  cuisine: { $ne: "American" },
  "grades.score": { $gt: 70 },
  "address.coord.1": { $lt: -65.754168 }
})
```

12. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.  
 Note : Do this query without using \$and operator.

```
> db.restaurants.find({
  cuisine: { $ne: "American" },
  "grades.score": { $gt: 70 },
  "address.coord.0": { $lt: -65.754168 }
})
```

13. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.

```
> db.restaurants.find({
  cuisine: { $ne: "American" },
  "grades.grade": "A",
  borough: { $ne: "Brooklyn" }
}).sort({ cuisine: -1 })
```

14. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.

```
> db.restaurants.find(
  { name: { $regex: /^Wil/ } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

15. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as last three letters for its name.

```
> db.restaurants.find(
  { name: { $regex: /ces$/ } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

16. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name.

```
> db.restaurants.find(
  { name: { $regex: /Reg/ } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

17. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.

```
> db.restaurants.find({
  borough: "Bronx",
  cuisine: { $in: ["American", "Chinese"] }
})
```

18. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronx or Brooklyn.

```
> db.restaurants.find(
  { borough: { $in: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

19. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn.

```
> db.restaurants.find(
  { borough: { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

20. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.

```
> db.restaurants.find(
  { "grades.score": { $lte: 10 } },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

21. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinese' or restaurant's name begins with letter 'Wil'.

```
> db.restaurants.find(
  {
    $or: [
      { cuisine: { $nin: ["American", "Chinese"] } },
      { name: { $regex: /^Wil/ } }
    ]
  },
  { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
)
```

22. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among

many of survey dates..

```
> db.restaurants.find(
  {
    grades: {
      $elemMatch: {
        grade: "A",
        score: 11,
        date: ISODate("2014-08-11T00:00:00Z")
      }
    }
  },
  { restaurant_id: 1, name: 1, grades: 1 }
)
<
```

23. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

```
> db.restaurants.find(
  {
    "grades.1.grade": "A",
    "grades.1.score": 9,
    "grades.1.date": ISODate("2014-08-11T00:00:00Z")
  },
  { restaurant_id: 1, name: 1, grades: 1 }
)
<
```

24. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..

```
> db.restaurants.find(
  {
    "address.coord.1": { $gt: 42, $lte: 52 }
  },
  { restaurant_id: 1, name: 1, address: 1 }
)
<
```

25. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

```
> db.restaurants.find().sort({ name: 1 })
<
```

26. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

```
> db.restaurants.find().sort({ name: -1 })
<
```

27. Write a MongoDB query to arrange the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
> db.restaurants.find().sort({ cuisine: 1, borough: -1 })  
<
```

28. Write a MongoDB query to know whether all the addresses contains the street or not.

```
> db.restaurants.find({ "address.street": { $exists: false } })  
<
```

29. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

```
> db.restaurants.find({  
    "address.coord": {  
        $type: "double"  
    }  
})  
<
```

30. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

```
> db.restaurants.find(  
    { "grades.score": { $mod: [7, 0] } },  
    { restaurant_id: 1, name: 1, grades: 1 }  
)  
<
```

31. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.

```
> db.restaurants.find(  
    { name: { $regex: /mon/, $options: "i" } },  
    {  
        name  
        ::contentReference[oaicite:0]{index=0}  
    }  
<
```

32. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.

```
> db.restaurants.find(  
    { name: { $regex: /^Mad/ } },  
    {  
        name: 1,  
        borough: 1,  
        "address.coord.0": 1, // Longitude  
        "address.coord.1": 1, // Latitude  
        cuisine: 1  
    }  
)  
<
```

33. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5.

```
> db.restaurants.find({
  "grades.score": { $lt: 5 }
})
```

34. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan.

```
> db.restaurants.find({
  "grades.score": { $lt: 5 },
  borough: "Manhattan"
})
```

35. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn.

```
> db.restaurants.find({
  "grades.score": { $lt: 5 },
  borough: { $in: ["Manhattan", "Brooklyn"] }
})
```

36. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
> db.restaurants.find({
  "grades.score": { $lt: 5 },
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $ne: "American" }
})
```

37. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
> db.restaurants.find({
  "grades.score": { $lt: 5 },
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $nin: ["American", "Chinese"] }
})
```

38. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6.

```
> db.restaurants.find({
  $and: [
    { "grades.score": 2 },
    { "grades.score": 6 }
  ]
})<
```

39. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan.

```
> db.restaurants.find({
  borough: "Manhattan",
  $and: [
    { "grades.score": 2 },
    { "grades.score": 6 }
  ]
})<
```

40. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  "grades.score": 2,
  "grades.score": 6
})<
```

41. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $ne: "American" },
  "grades.score": 2,
  "grades.score": 6
})<
```

42. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $nin: ["American", "Chinese"] },
  $and: [
    { grades: { $elemMatch: { score: 2 } } },
    { grades: { $elemMatch: { score: 6 } } }
  ]
})<
```

43. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6.

```
> db.restaurants.find({
  grades: { $elemMatch: { score: { $in: [2, 6] } } }
})
```

44. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6 and are located in the borough of Manhattan.

```
> db.restaurants.find({
  borough: "Manhattan",
  grades: { $elemMatch: { score: { $in: [2, 6] } } }
})
```

45. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  grades: { $elemMatch: { score: { $in: [2, 6] } } }
})
```

46. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $ne: "American" },
  grades: { $elemMatch: { score: { $in: [2, 6] } } }
})
```

47. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $nin: ["American", "Chinese"] },
  grades: { $elemMatch: { score: { $in: [2, 6] } } }
})
```

48. Write a MongoDB query to find the restaurants that have all grades with a score greater than 5.

```
> db.restaurants.find({
  grades: { $not: { $elemMatch: { score: { $lte: 5 } } } }
})<
```

49. Write a MongoDB query to find the restaurants that have all grades with a score greater than 5 and are located in the borough of Manhattan.

```
> db.restaurants.find({
  borough: "Manhattan",
  grades: { $not: { $elemMatch: { score: { $lte: 5 } } } }
})<
```

50. Write a MongoDB query to find the restaurants that have all grades with a score greater than 5 and are located in the borough of Manhattan or Brooklyn.

```
> db.restaurants.find({
  borough: { $in: ["Manhattan", "Brooklyn"] },
  grades: { $not: { $elemMatch: { score: { $lte: 5 } } } }
})<
```

51. Write a MongoDB query to find the average score for each restaurant.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$name",
      avgScore: { $avg: "$grades.score" }
    }
  }
])<
```

52. Write a MongoDB query to find the highest score for each restaurant.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$name",
      maxScore: { $max: "$grades.score" }
    }
  }
])<
```

53. Write a MongoDB query to find the lowest score for each restaurant.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$name",
      minScore: { $min: "$grades.score" }
    }
  }
])<
```

54. Write a MongoDB query to find the count of restaurants in each borough.

```
> db.restaurants.aggregate([
  {
    $group: {
      _id: "$borough",
      count: { $sum: 1 }
    }
  }
])<
```

55. Write a MongoDB query to find the count of restaurants for each cuisine.

```
> db.restaurants.aggregate([
  {
    $group: {
      _id: "$cuisine",
      count: { $sum: 1 }
    }
  }
])<
```

56. Write a MongoDB query to find the count of restaurants for each cuisine and borough.

```
> db.restaurants.aggregate([
  {
    $group: {
      _id: { cuisine: "$cuisine", borough: "$borough" },
      count: { $sum: 1 }
    }
  }
])<
```

57. Write a MongoDB query to find the count of restaurants that received a grade of 'A' for each cuisine.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "A" } },
  {
    $group: {
      _id: "$cuisine",
      count: { $sum: 1 }
    }
  }
])<
```

58. Write a MongoDB query to find the count of restaurants that received a grade of 'A' for each borough.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "A" } },
  {
    $group: {
      _id: "$borough",
      count: { $sum: 1 }
    }
  }
])<
```

59. Write a MongoDB query to find the count of restaurants that received a grade of 'A' for each cuisine and borough.

```
> db.restaurants.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "A" } },
  {
    $group: {
      _id: { cuisine: "$cuisine", borough: "$borough" },
      count: { $sum: 1 }
    }
  }
])<
```

60. Write a MongoDB query to find the number of restaurants that have been graded in each month of the year.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: { $month: "$grades.date" },
      count: { $sum: 1 }
    }
  },
  { $sort: { "_id": 1 } }
])<
```

61. Write a MongoDB query to find the average score for each cuisine.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$cuisine",
      avgScore: { $avg: "$grades.score" }
    }
  }
])
```

62. Write a MongoDB query to find the highest score for each cuisine.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$cuisine",
      maxScore: { $max: "$grades.score" }
    }
  }
])
```

63. Write a MongoDB query to find the lowest score for each cuisine.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$cuisine",
      minScore: { $min: "$grades.score" }
    }
  }
])
```

64. Write a MongoDB query to find the average score for each borough.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$borough",
      avgScore: { $avg: "$grades.score" }
    }
  }
])
```

65. Write a MongoDB query to find the highest score for each borough.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$borough",
      maxScore: { $max: "$grades.score" }
    }
  }
])
```

66. Write a MongoDB query to find the lowest score for each borough.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$borough",
      minScore: { $min: "$grades.score" }
    }
  }
])
```

67. Write a MongoDB query to find the name and address of the restaurants that received a grade of 'A' on a specific date.

```
> db.Restaurant.find({
  grades: {
    $elemMatch: {
      grade: "A",
      date: ISODate("2014-03-03T00:00:00Z") // example date
    }
  }
}, { name: 1, address: 1 })
```

68. Write a MongoDB query to find the name and address of the restaurants that received a grade of 'B' or 'C' on a specific date.

```
> db.Restaurant.find({
  grades: {
    $elemMatch: {
      grade: { $in: ["B", "C"] },
      date: ISODate("2014-03-03T00:00:00Z") // example date
    }
  }
}, { name: 1, address: 1 })
```

69. Write a MongoDB query to find the name and address of the restaurants that have at least one 'A' grade and one 'B' grade.

```
> db.Restaurant.find({
  grades: {
    $all: [
      { $elemMatch: { grade: "A" } },
      { $elemMatch: { grade: "B" } }
    ]
  }
}, { name: 1, address: 1 })
```

70. Write a MongoDB query to find the name and address of the restaurants that have at least one 'A' grade and no 'B' grades.

```
> db.Restaurant.find({
  "grades.grade": "A",
  "grades.grade": { $ne: "B" }
}, { name: 1, address: 1 })
```

71. Write a MongoDB query to find the name ,address and grades of the restaurants that have at least one 'A' grade and no 'C' grades.

```
> db.Restaurant.find({
  "grades.grade": "A",
  "grades.grade": { $ne: "C" }
}, { name: 1, address: 1, grades: 1 })
```

72. Write a MongoDB query to find the name, address, and grades of the restaurants that have at least one 'A' grade, no 'B' grades, and no 'C' grades.

```
> db.Restaurant.find({
  "grades.grade": "A",
  "grades.grade": { $nin: ["B", "C"] }
}, { name: 1, address: 1, grades: 1 })
```

73. Write a MongoDB query to find the name and address of the restaurants that have the word 'coffee' in their name.

```
> db.Restaurant.find({
  name: /coffee/i
}, { name: 1, address: 1 })
```

74. Write a MongoDB query to find the name and address of the restaurants that have a zipcode that starts with '10'.

```
> db.Restaurant.find({
  "address.zipcode": /^10/
}, { name: 1, address: 1 })
```

75. Write a MongoDB query to find the name and address of the restaurants that have a cuisine that starts with the letter 'B'.

```
> db.Restaurant.find({
  cuisine: /^B/
}, { name: 1, address: 1 })
```

76. Write a MongoDB query to find the name, address, and cuisine of the restaurants that have a cuisine that ends with the letter 'y'.

```
> db.Restaurant.find({
  cuisine: /y$/,
}, { name: 1, address: 1, cuisine: 1 })
```

77. Write a MongoDB query to find the name, address, and cuisine of the restaurants that have a cuisine that contains the word 'Pizza'.

```
> db.Restaurant.find({
  cuisine: /Pizza/i
}, { name: 1, address: 1, cuisine: 1 })
<
```

78. Write a MongoDB query to find the restaurants achieved highest average score.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$_id",
      name: { $first: "$name" },
      avgScore: { $avg: "$grades.score" }
    }
  },
  { $sort: { avgScore: -1 } },
  { $limit: 1 }
])
```

79. Write a MongoDB query to find all the restaurants with the highest number of "A" grades.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "A" } },
  {
    $group: {
      _id: "$name",
      countA: { $sum: 1 }
    }
  },
  { $sort: { countA: -1 } },
  { $limit: 1 }
])
```

80. Write a MongoDB query to find the cuisine type that is most likely to receive a "C" grade.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "C" } },
  {
    $group: {
      _id: "$cuisine",
      count: { $sum: 1 }
    }
  },
  { $sort: { count: -1 } },
  { $limit: 1 }
])
<
```

81. Write a MongoDB query to find the restaurant that has the highest average score for the cuisine "Turkish".

```
> db.Restaurant.aggregate([
  { $match: { cuisine: "Turkish" } },
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$name",
      avgScore: { $avg: "$grades.score" }
    }
  },
  { $sort: { avgScore: -1 } },
  { $limit: 1 }
])
<
```

82. Write a MongoDB query to find the restaurants that achieved the highest total score.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$name",
      totalScore: { $sum: "$grades.score" }
    }
  },
  { $sort: { totalScore: -1 } },
  { $limit: 1 }
])
< {
  _id: 'Morris Park Bake Shop',
  totalScore: 41
}
```

83. Write a MongoDB query to find all the Chinese restaurants in Brooklyn.

```
> db.Restaurant.find({
  borough: "Brooklyn",
  cuisine: "Chinese"
})
<
```

84. Write a MongoDB query to find the restaurant with the most recent grade date.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  { $sort: { "grades.date": -1 } },
  { $limit: 1 },
  {
    $project: {
      name: 1,
      "grades.date": 1
    }
  }
])
```

85. Write a MongoDB query to find the top 5 restaurants with the highest average score for each cuisine type, along with their average scores.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $group: {
      _id: "$_id",
      name: { $first: "$name" },
      cuisine: { $first: "$cuisine" },
      avgScore: { $avg: "$grades.score" }
    }
  },
  {
    $sort: { cuisine: 1, avgScore: -1 }
  },
  {
    $group: {
      _id: "$cuisine",
      topRestaurants: { $push: { name: "$name", avgScore: "$avgScore" } }
    }
  },
  {
    $project: {
      top5: { $slice: ["$topRestaurants", 5] }
    }
  }
])
```

86. Write a MongoDB query to find the top 5 restaurants in each borough with the highest number of "A" grades.

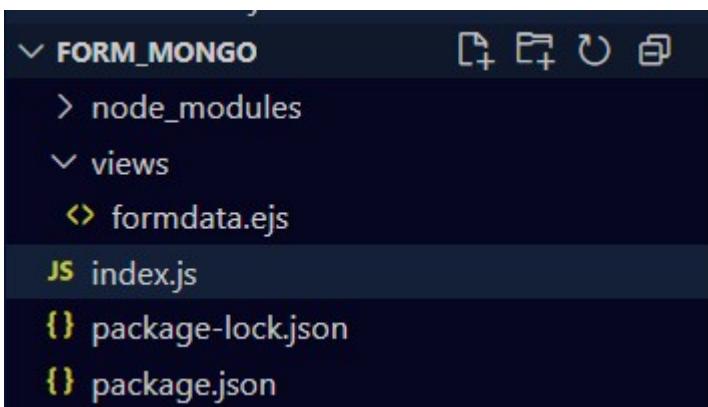
```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  { $match: { "grades.grade": "A" } },
  {
    $group: {
      _id: "$_id",
      name: { $first: "$name" },
      borough: { $first: "$borough" },
      aCount: { $sum: 1 }
    }
  },
  { $sort: { borough: 1, aCount: -1 } },
  {
    $group: {
      _id: "$borough",
      top: { $push: { name: "$name", aCount: "$aCount" } }
    }
  },
  {
    $project: {
      top5: { $slice: ["$top", 5] }
    }
  }
])
```

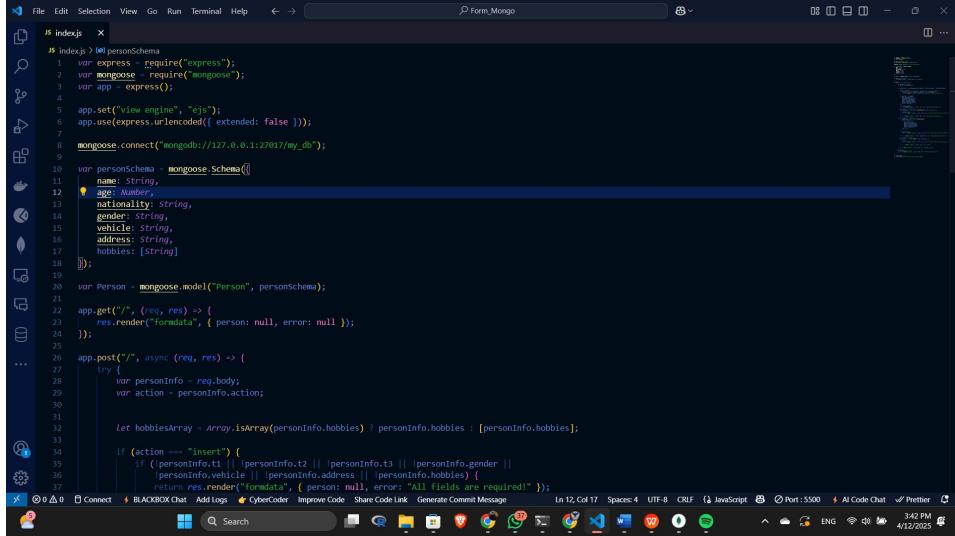
87. Write a MongoDB query to find the borough with the highest number of restaurants that have a grade of "A" and a score greater than or equal to 90.

```
> db.Restaurant.aggregate([
  { $unwind: "$grades" },
  {
    $match: {
      "grades.grade": "A",
      "grades.score": { $gte: 90 }
    }
  },
  {
    $group: {
      _id: "$borough",
      count: { $sum: 1 }
    }
  },
  { $sort: { count: -1 } },
  { $limit: 1 }
])
```

### Practical 8: Database Connectivity:

#### Code

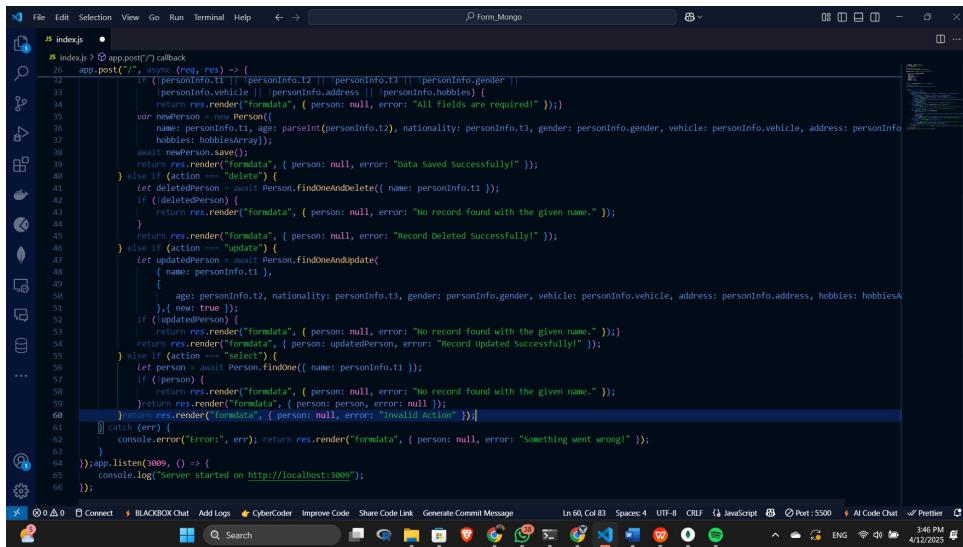




```

1  // index.js
2  var express = require('express');
3  var mongoose = require('mongoose');
4  var app = express();
5
6  app.set('view engine', 'ejs');
7  app.use(express.urlencoded({ extended: false }));
8
9  mongoose.connect('mongodb://127.0.0.1:27017/my_db');
10
11 var personSchema = mongoose.Schema({
12   name: String,
13   age: Number,
14   nationality: string,
15   gender: string,
16   vehicle: string,
17   address: string,
18   hobbies: [String]
19 });
20
21 var Person = mongoose.model("Person", personSchema);
22
23 app.get('/', (req, res) => {
24   res.render("formdata", { person: null, error: null });
25 }
26
27 app.post("/", async (req, res) => {
28   try {
29     var personInfo = req.body;
30     var action = personInfo.action;
31
32     let hobbiesArray = Array.isArray(personInfo.hobbies) ? personInfo.hobbies : [personInfo.hobbies];
33
34     if (action === "insert") {
35       if (!personInfo.t1 || !personInfo.t2 || !personInfo.t3 || !personInfo.gender || !personInfo.vehicle || !personInfo.address || !personInfo.hobbies) {
36         return res.render("formdata", { person: null, error: "All fields are required!" });
37     }
38     await newPerson.save();
39     return res.render("formdata", { person: null, error: "Data Saved Successfully!" });
40   } else if (action === "delete") {
41     let deletedPerson = await Person.findByIdAndDelete({ name: personInfo.t1 });
42     if (!deletedPerson) {
43       return res.render("formdata", { person: null, error: "No record found with the given name." });
44     }
45     return res.render("formdata", { person: null, error: "Record Deleted Successfully!" });
46   } else if (action === "update") {
47     let updatedPerson = await Person.findByIdAndUpdate(
48       { name: personInfo.t1 },
49       {
50         $set: { personInfo.t2, nationality: personInfo.t3, gender: personInfo.gender, vehicle: personInfo.vehicle, address: personInfo.address, hobbies: hobbiesArray },
51       },
52       { new: true });
53     if (!updatedPerson) {
54       return res.render("formdata", { person: null, error: "No record found with the given name." });
55     }
56     return res.render("formdata", { person: updatedPerson, error: "Record Updated Successfully!" });
57   } else if (action === "select") {
58     let person = await Person.findOne({ name: personInfo.t1 });
59     if (!person) {
60       return res.render("formdata", { person: null, error: "No record found with the given name." });
61     }
62     return res.render("formdata", { person: person, error: null });
63   }
64 }
65 ]
66 catch (err) {
67   console.error(`Error: ${err}`);
68   return res.render("formdata", { person: null, error: "Something went wrong!" });
69 }
70 );
71 app.listen(3009, () => {
72   console.log("Server started on http://localhost:3009");
73 });

```



```

1  // index.js
2  app.post("/", async (req, res) => {
3
4    if (!personInfo.t1 || !personInfo.t2 || !personInfo.t3 || !personInfo.gender || !personInfo.vehicle || !personInfo.address || !personInfo.hobbies) {
5      return res.render("formdata", { person: null, error: "All fields are required!" });
6    }
7
8    var newPerson = new Person();
9    newPerson.name = personInfo.t1;
10   newPerson.age = parseInt(personInfo.t2);
11   newPerson.nationality = personInfo.t3;
12   newPerson.gender = personInfo.gender;
13   newPerson.vehicle = personInfo.vehicle;
14   newPerson.address = personInfo.address;
15   newPerson.hobbies = hobbiesArray;
16
17   await newPerson.save();
18
19   return res.render("formdata", { person: null, error: "Data Saved Successfully!" });
20
21 } else if (action === "delete") {
22   let deletedPerson = await Person.findByIdAndDelete({ name: personInfo.t1 });
23   if (!deletedPerson) {
24     return res.render("formdata", { person: null, error: "No record found with the given name." });
25   }
26   return res.render("formdata", { person: null, error: "Record Deleted Successfully!" });
27
28 } else if (action === "update") {
29   let updatedPerson = await Person.findByIdAndUpdate(
30     { name: personInfo.t1 },
31     {
32       $set: { personInfo.t2, nationality: personInfo.t3, gender: personInfo.gender, vehicle: personInfo.vehicle, address: personInfo.address, hobbies: hobbiesArray },
33     },
34     { new: true });
35
36   if (!updatedPerson) {
37     return res.render("formdata", { person: null, error: "No record found with the given name." });
38   }
39   return res.render("formdata", { person: updatedPerson, error: "Record Updated Successfully!" });
40
41 } else if (action === "select") {
42   let person = await Person.findOne({ name: personInfo.t1 });
43   if (!person) {
44     return res.render("formdata", { person: null, error: "No record found with the given name." });
45   }
46   return res.render("formdata", { person: person, error: null });
47
48 }
49 ]
50 catch (err) {
51   console.error(`Error: ${err}`);
52   return res.render("formdata", { person: null, error: "Something went wrong!" });
53 }
54 );
55 app.listen(3009, () => {
56   console.log("Server started on http://localhost:3009");
57 });

```

## Run the App

```

PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Form_Mongo> node index.js
Server started on http://localhost:3009

```

## CRUD Operations and Output

**Person Form**

Name:

Age:

Nationality:

Gender:

Male  Female  Other

Vehicle:

Address:

Hobbies:

Reading  Music  Sports  Traveling

## Insert Operation

### Person Form

Data Saved Successfully!

Name:

Age:

Nationality:

Gender:

Male  Female  Other

Vehicle:

Address:

Hobbies:

Reading  Music  Sports  Traveling

```
_id: ObjectId('67fa3eb147a43de822d7cf37')
name : "Sample"
age : 11
nationality : "Irish"
gender : "Male"
vehicle : "Car"
address : "Dublin
Ireland"
▶ hobbies : Array (2)
__v : 0
```

## Search Operation

### Person Form

Name:

Age:

Nationality:

Gender:

Male  Female  Other

Vehicle:

Address:

Hobbies:

Reading  Music  Sports  Traveling

## Update Operation

### Person Form

Record Updated Successfully!

Name:

Age:

Nationality:

Gender:

Male  Female  Other

Vehicle:

Address:

Hobbies:

Reading  Music  Sports  Traveling

```
_id: ObjectId('67ac12cb8bae716ddbd8197d')
name : "AGNISHWAR"
age : 21
nationality : "Indian"
gender : "Male"
vehicle : "Train"
address : "aadsaffdagf
dasfad"
▶ hobbies : Array (1)
__v : 0
```

## Delete Operation

## Person Form

Record Deleted Successfully!

Name:   
 Age:   
 Nationality:

Gender:  
 Male  Female  Other

Vehicle:

Address:

Hobbies:

Reading  Music  Sports  Traveling



25 ▾ 1 - 2 of 2 ⏪

## Practical 9: Session and Cookies Demo :

### Session management

#### Code

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();
app.use(cookieParser());
app.use(express.urlencoded({ extended: true }));
app.get('/', (req, res) => {
  const session = req.cookies.session;
  if (session) {
    res.send(`
      <h2>Welcome back, ${session}</h2>
      <a href="/logout">Logout</a>
    `);
  } else {
    res.send(`
      <form method="POST" action="/login">
        <label>Username: <input type="text" name="username" /></label>
        <button type="submit">Login</button>
      </form>
    `);
  }
}); app.post('/login', (req, res) => {
  const username = req.body.username;
  if (!username) return res.redirect('/');
  res.cookie('session', username, { maxAge: 5000, httpOnly: true });
  res.redirect('/');
}); app.get('/logout', (req, res) => {
  res.clearCookie('session');
  res.redirect('/');
});
const PORT = 3017;
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

#### Output

Welcome back, customer!

[Logout](#)

Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite	Partition Key	Cross Site	Priority
api_anonymous_id	8ae796ff-817f-4320-a5fc-326d8fc07d1	localhost	/	2026-04-01...	52	✓					Medium
iconsize	10x10	localhost	/	2025-07-01...	13	✓					Medium
jwt-auth-stamp-offset	10000000	localhost	/	2026-04-01...	13						Medium
session	customer	localhost	/	2025-04-01...	15	✓					Medium

After 5 seconds

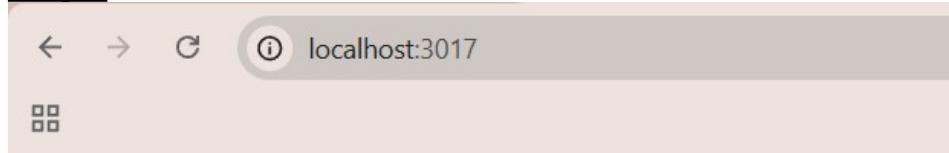
Username:  Login

Make use of cookies for storing client data

## Code

```
js index.js > ...
1 const express = require('express');
2 const cookieParser = require('cookie-parser');
3 const app = express();
4 const urlencodedParser = () => {
5   const app = Express()
6     .coded({ extended: true })
7     .get('/', (req, res) => {
8       const name = req.cookies.name;
9       if (name) {
10         res.send(`
11           <h2>Hello, ${name}! </h2>
12           <a href="/clear">Clear Name</a>
13         `);
14       } else {
15         res.send(`
16           <form method="POST" action="/setname">
17             <label>Enter your name: <input type="text" name="name" required></label>
18             <button type="submit">Submit</button>
19           </form>
20         `);
21     });
22     app.post('/setname', (req, res) => {
23       const { name } = req.body;
24       res.cookie('name', name, { maxAge: 24 * 60 * 60 * 1000, httpOnly: true });
25       res.redirect('/');
26     });
27     app.get('/clear', (req, res) => {
28       res.clearCookie('name');
29       res.redirect('/');
30     });
31     const PORT = 3017;
32     app.listen(PORT, () => {
33       console.log(`Server running at http://localhost:${PORT}`);
34     });
35   }
}
```

## Output



Hello, Agnishwar Raychaudhuri! 🙌

## Clear Name

## Practical 10: Login Exercise :

### Install dependencies

```
PS C:\Users\Augni\Downloads\login-auth-app> npm init -y
```

```
PS C:\Users\Augni\Downloads\login-auth-app> npm install express ejs express-session body-parser dotenv
```

### Code

File Explorer View

```
└─ LOGIN-AUTH-APP
    └─ login-auth-app
        ├─ public
        │   # style.css
        ├─ routes
        │   JS auth.js
        └─ views
            └─ dashboard.ejs
            └─ login.ejs
            └─ register.ejs
        └─ .env
    └─ JS app.js
    > node_modules
    {─ package-lock.json
    {─ package.json
```

Code Editor View

```
JS app.js x
login-auth-app > JS app.js > ...
1 const express = require('express');
2 const session = require('express-session');
3 const bodyParser = require('body-parser');
4 const path = require('path');
5 const fs = require('fs');
6 const dotenv = require('dotenv');
7 const config = require('./config');
8 const app = express();
9 const PORT = process.env.PORT || 4000;
10 app.use(bodyParser.urlencoded({ extended: true }));
11 app.use(express.static('public'));
12 app.use(session({
13     secret: process.env.SESSION_SECRET,
14     resave: false,
15     saveUninitialized: true
16 }));
17 app.set('view engine', 'ejs');
18 app.set('views', path.join(__dirname, 'views'));
19 const authRoutes = require('./routes/auth');
20 app.use('/', authRoutes);
21 app.listen(PORT, () => {
22     console.log(`Server running on http://localhost:${PORT}`);
23 });
```

Code Editor View

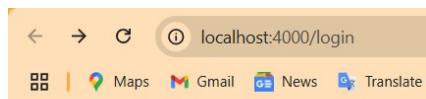
```
JS app.js x JS auth.js x
C:\Users\Augni\Downloads\login-auth-app\login-auth-app\app.js | callback
1 const express = require('express');
2 const router = express.Router();
3 const users = [];
4 router.get('/login', (req, res) => {
5     res.redirect('/login');
6 });
router.get('/login', (req, res) => {
7     res.render('login', { message: null });
8 });
router.post('/login', (req, res) => {
9     const { username, password } = req.body;
10    const user = users.find(u => u.username === username && u.password === password);
11    if (user) {
12        req.session.user = user;
13        res.redirect('/dashboard');
14    } else {
15        res.render('login', { message: 'Invalid username or password' });
16    }
17 });
router.get('/register', (req, res) => {
18     res.render('register', { message: null });
19 });
router.post('/register', (req, res) => [
20     const { username, password } = req.body;
21     if (users.find(u => u.username === username)) {
22         return res.render('register', { message: 'Username already exists' });
23     }
24     users.push({username, password});
25     res.redirect('/login');
26 ]);
router.get('/dashboard', (req, res) => {
27     if (!req.session.user) return res.redirect('/login');
28     res.render('dashboard', { user: req.session.user });
29 });
router.get('/logout', (req, res) => {
30     req.session.destroy();
31     res.redirect('/login');
32 });
module.exports = router;
```

Run the app

```
PS C:\Users\Augni\Downloads\login-auth-app\login-auth-app> node app.js
Server running on http://localhost:4000

```

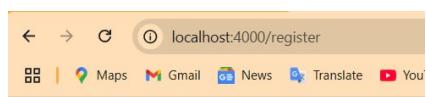
## Output



### Login

Username
Password
<input type="button" value="Login"/>

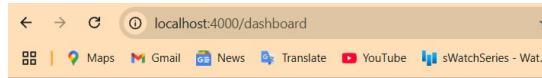
Don't have an account? [Register](#)



### Register

Username
Password
<input type="button" value="Register"/>

Already have an account? [Login](#)

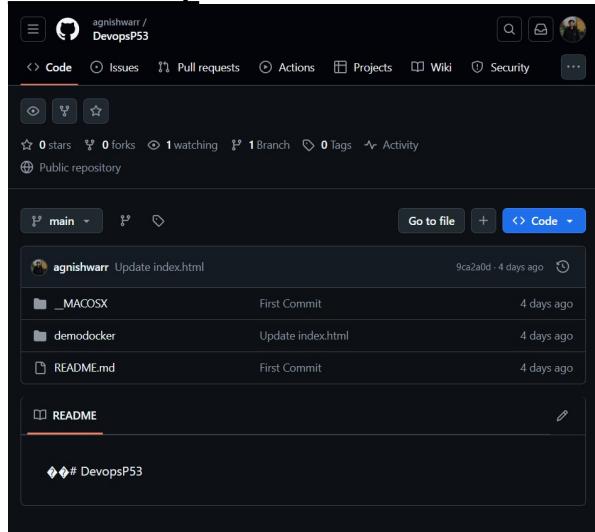


Welcome, Agni!

[Logout](#)

## Practical 11: Setting up DevOPS for Project :

### GitHub Setup



Code Issues Pull requests Actions Projects Wiki Security

0 stars 0 forks 1 watching 1 Branch 0 Tags Activity

Public repository

main Go to file + <> Code

agnishwarr Update index.html 9ca2a0d · 4 days ago

MACOSX First Commit 4 days ago

demodocker Update index.html 4 days ago

README.md First Commit 4 days ago

README

DevopsP53

## Jenkins Setup

Discard old builds ?

GitHub project  
Project url ?  
`https://github.com/agnishwarr/DevopsP53.git/`

Advanced ▾

This project is parameterised ?

Throttle builds ?

Execute concurrent builds if necessary ?

Advanced ▾

**Source Code Management**  
Connect and manage your code repository to automatically pull the latest code for your builds.

None

Git ?  
Repositories ?

Repository URL ? `https://github.com/agnishwarr/DevopsP53.git`

Credentials ?  
- none -

+ Add

Advanced ▾

Add Repository

Branch Specifier (blank for 'any') ? `*/main`

Add Branch

**Build Steps**  
Automate your build process with ordered tasks like code compilation, testing, and deployment.

Execute Windows batch command ?

Command  
See the list of available environment variables

```
cd demodocker
dir
set "PATH=%PATH%;C:\Program Files\Docker\ Docker\resources\bin"
docker build -t my_app:1.0 .
docker compose up -d
```

Advanced ▾

Add build step ▾

## Docker and Dockerfile

```
README.md
```

## demo app - developing with Docker

This demo app shows a simple user profile app set up using

- index.html with pure js and css styles
- nodejs backend with express module
- mongodb for data storage

All components are docker-based

### With Docker

To start the application

Step 1: Create docker network

```
docker network create mongo-network
```

Step 2: start mongodb

```
docker run -d -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=123456
```

### With Docker Compose

To start the application

Step 1: start mongodb and mongo-express

```
docker-compose -f docker-compose.yaml up
```

You can access the mongo-express under localhost:8080 from your browser

Step 2: in mongo-express UI – create a new database "my-db"

Step 3: in mongo-express UI – create a new collection "users" in the database "my-db"

Step 4: start node server

```
npm install  
node server.js
```

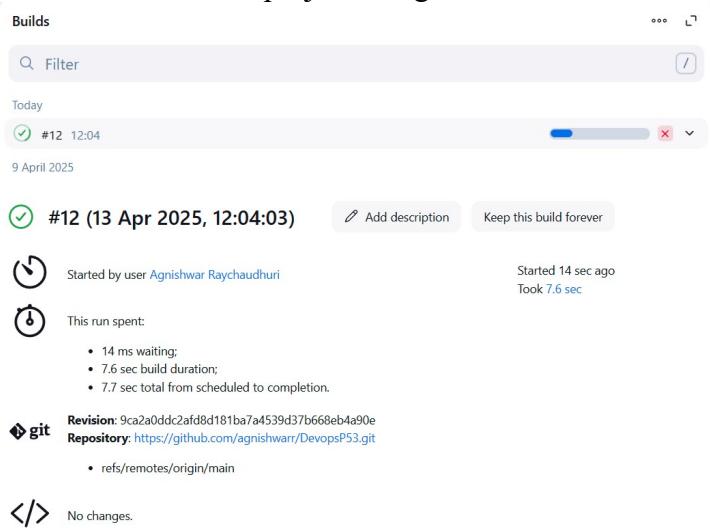
Step 5: access the nodejs application from browser

```
http://localhost:3000
```

To build a docker image from the application

```
docker build -t my-app:1.0 .
```

Let us now build the project using Jenkins



The screenshot shows the Jenkins build history for a project named 'Builds'. It displays a single build entry for '#12 (13 Apr 2025, 12:04:03)'. The build was started by user Agnishwar Raychaudhuri and took 7.6 seconds. The build status is green, indicating success. The Jenkins interface includes a 'Filter' search bar, a date range selector for 'Today', and buttons for 'Add description' and 'Keep this build forever'.

**#12 (13 Apr 2025, 12:04:03)**

Started by user Agnishwar Raychaudhuri      Started 14 sec ago  
Took 7.6 sec

This run spent:

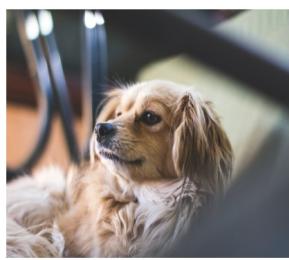
- 14 ms waiting;
- 7.6 sec build duration;
- 7.7 sec total from scheduled to completion.

**git** Revision: 9ca2a0ddc2af8d181ba7a4539d37b668eb4a90e  
Repository: <https://github.com/agnishwarr/DevopsP53.git>

</> No changes.

## Output

### User profile



Pet Name: **Cute Dog**

Email Id: **anna.smith@example.com**

Interests: **coding**

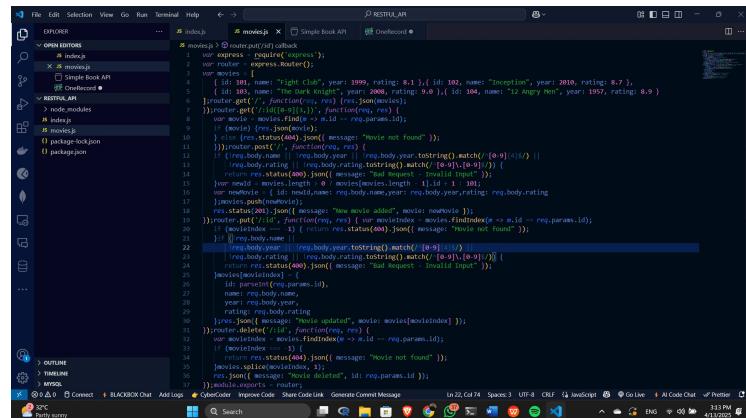
[Edit](#)

## Practical 12: APIs :

Host Movie API and check it.

### Code

```
JS index.js  JS movies.js  Simple Book API  RESTful API OneRecord
JS C:\Users\Agnih\Desktop\NIIT\Academics\SEM VI\Capstone\RESTFUL_API\index.js
1 var express = require('express');
2 var bodyParser = require('body-parser');
3 var cookieParser = require('cookie-parser');
4 var multer = require('multer');
5 var upload = multer();
6 var app = express();
7 app.use(cookieParser());
8 app.use(bodyParser.json());
9 app.use(bodyParser.urlencoded({ extended: true }));
10 app.use(upload.array());
11 var movies = require('./movies.js');
12 app.use('/movies', movies);
13 var PORT = process.env.PORT || 3004;
14 app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
15 });
16 
```

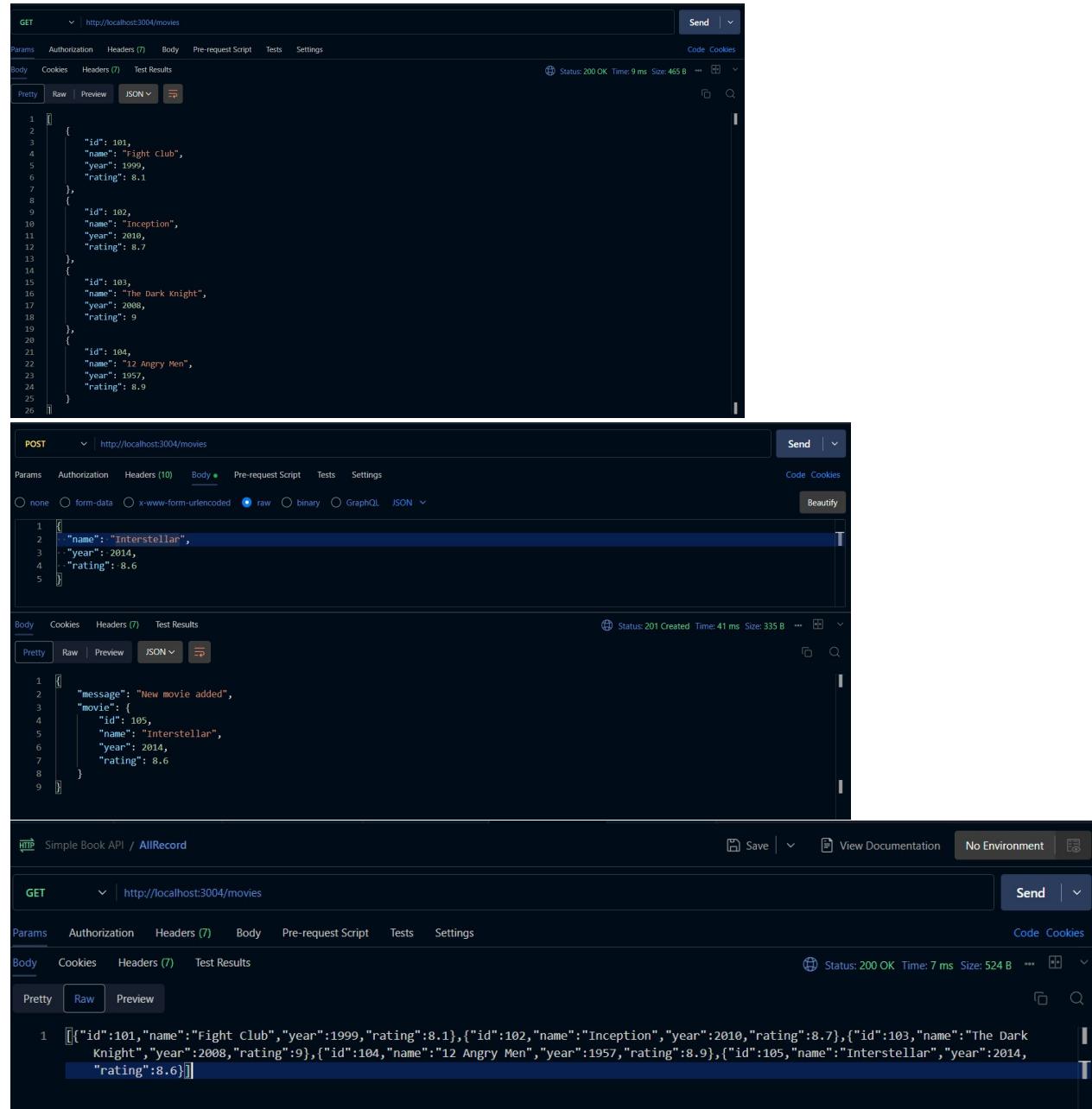


```
index.js movies.js Simple Book API RESTful API OneRecord
1 router.get('/movies', function(req, res) {
2   var movies = [
3     { id: 102, name: "Right Club", year: 1999, rating: 8.1 },
4     { id: 103, name: "The Dark Knight", year: 2008, rating: 9.0 },
5     { id: 104, name: "12 Angry Men", year: 1957, rating: 8.9 }
6   ];
7   res.json(movies);
8 });
9 router.post('/movies', function(req, res) {
10   if (!req.body.name || !req.body.year || !req.body.rating) {
11     return res.status(400).json({ message: 'Movie not found' });
12   }
13   if (!req.body.name || !req.body.year || !req.body.rating || !req.body.testing) {
14     return res.status(400).json({ message: 'Bad request - invalid input' });
15   }
16   var newMovie = {
17     id: req.body.id,
18     name: req.body.name,
19     year: req.body.year,
20     rating: req.body.rating,
21     testing: req.body.testing
22   };
23   var movieIndex = movies.findIndex(movie => movie.id === req.params.id);
24   if (movieIndex === -1) {
25     movies.push(newMovie);
26   } else {
27     movies[movieIndex] = newMovie;
28   }
29   res.json(movies);
30 });
31 router.put('/movies/:id', function(req, res) {
32   if (!req.body.name || !req.body.year || !req.body.rating) {
33     return res.status(400).json({ message: 'Movie not found' });
34   }
35   var movieIndex = movies.findIndex(movie => movie.id === req.params.id);
36   if (movieIndex === -1) {
37     return res.status(404).json({ message: 'Movie not found' });
38   }
39   var movie = movies[movieIndex];
40   movie.name = req.body.name;
41   movie.year = req.body.year;
42   movie.rating = req.body.rating;
43   if (req.body.testing) {
44     movie.testing = req.body.testing;
45   }
46   movies[movieIndex] = movie;
47   res.json(movies);
48 });
49 router.delete('/movies/:id', function(req, res) {
50   var movieIndex = movies.findIndex(movie => movie.id === req.params.id);
51   if (movieIndex === -1) {
52     return res.status(404).json({ message: 'Movie not found' });
53   }
54   movies.splice(movieIndex, 1);
55   res.json(movies);
56 });
57 module.exports = router;
58 
```

## Run the App

```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\RESTFUL_API> node index.js
Server is running on port 3004
```

## Output



The screenshot shows three API interactions using the Postman application:

- GET /movies**: A request to retrieve all movies. The response is a JSON array containing five movie objects.
- POST /movies**: A request to add a new movie. The request body is a JSON object with fields: name, year, and rating. The response is a JSON object indicating the new movie has been added with its details.
- GET /movies**: Another request to retrieve all movies, resulting in the same JSON array as the first GET request.

The JSON data for the movies is as follows:

```

[{"id": 101, "name": "Fight Club", "year": 1999, "rating": 8.1}, {"id": 102, "name": "Inception", "year": 2010, "rating": 8.7}, {"id": 103, "name": "The Dark Knight", "year": 2008, "rating": 9}, {"id": 104, "name": "12 Angry Men", "year": 1957, "rating": 8.9}
]

```