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

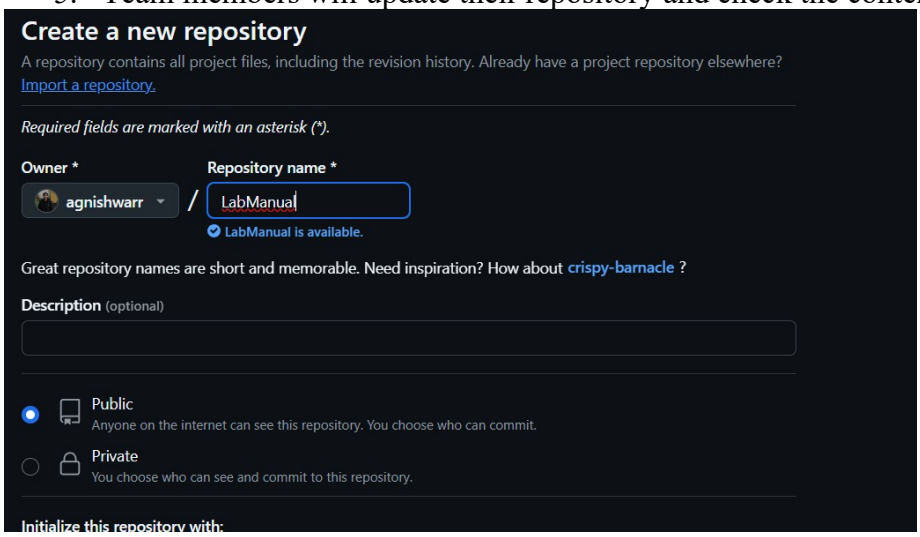
List of Experiments

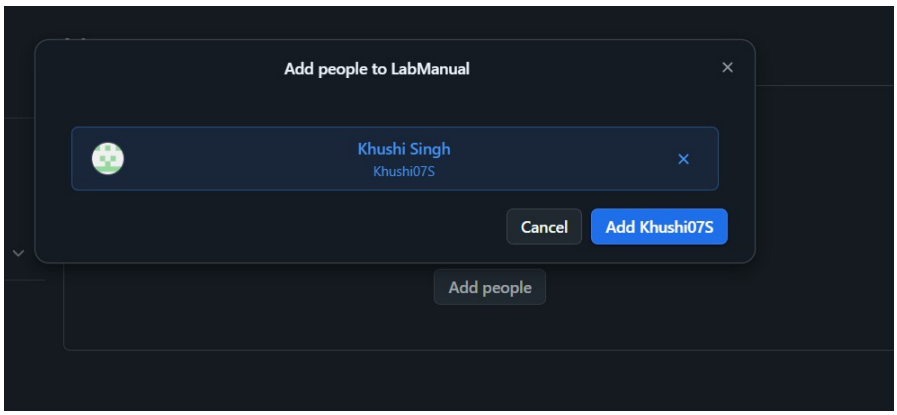
Practical No.	Aim of Practical	COs	Page No.
1	Collaboration Tool-Git	CO2	1
2	HTML and CSS	CO1	2
3	Javascript	CO1	4
4	BootStrap	CO1	5
5	NodeJS	CO3	6
6	ExpressJS	CO3	6
7	MongoDB	CO3	7
8	Database Connectivity	CO3	25
9	Session and Cookies Demo	CO3	28
10	Login Exercise	CO3	30
11	Setting up DevOPS for Project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

Practical 1: Collaboration Tool-Git:

Steps of Execution:

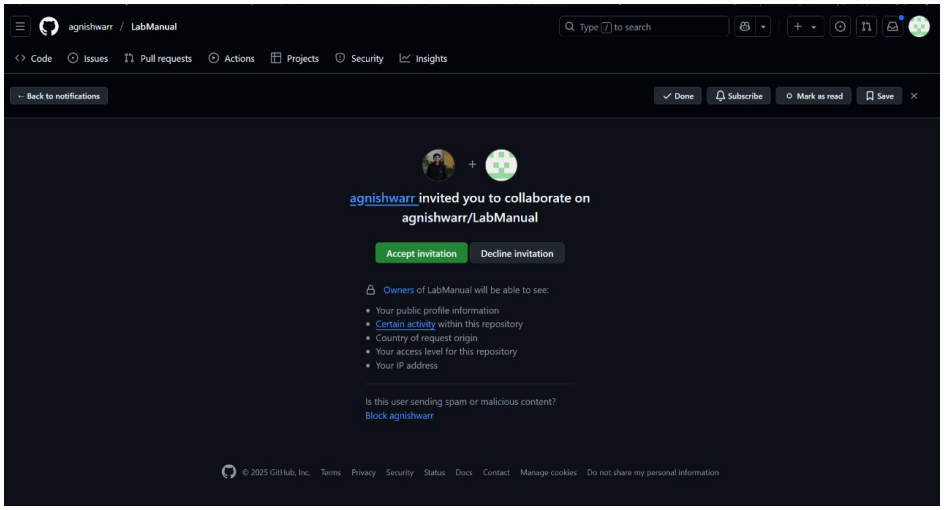
1. Team lead will make Github repository and will send invite to other team members.
2. Team lead will create local repo and then push to remote repo
3. Team members will accept the invite and clone repo on their local machine.
4. All the team members will share file using git / github
5. Team members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



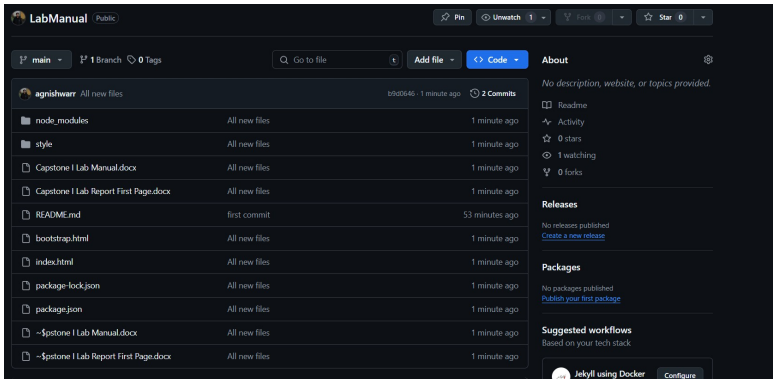



```
echo "# LabManual" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/agnishwarr/LabManual.git
git push -u origin main
```

```
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>
```




```
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
```




```
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% (350/350), done.
```

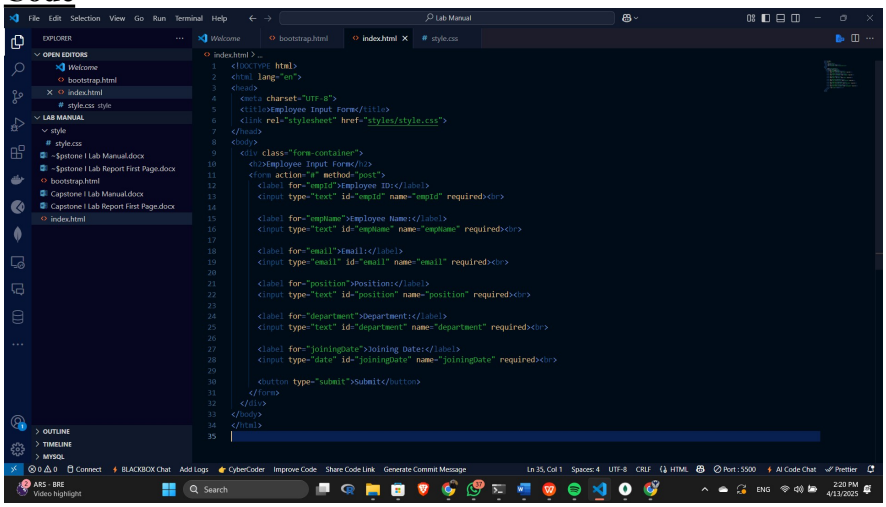
Practical 2: HTML and CSS:

Steps of Execution:

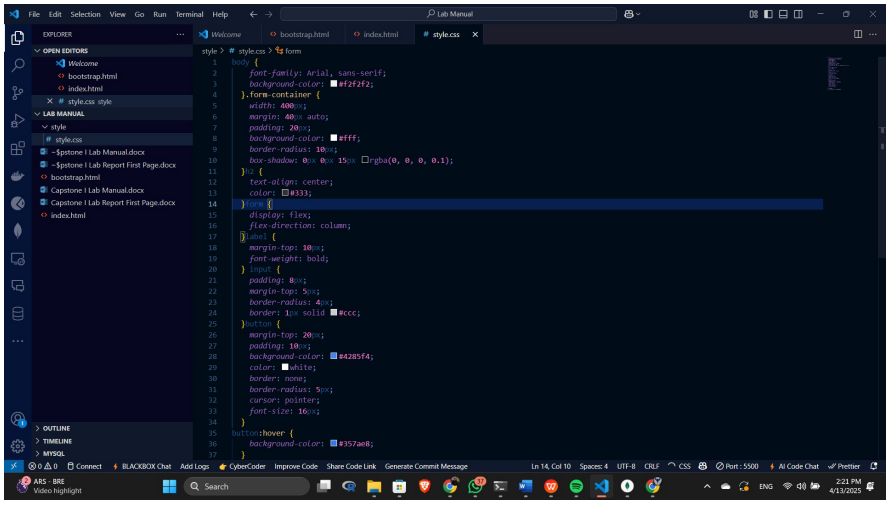
1. Make an input form for an employee.

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">
    <h2>Employee Input Form</h2>
    <form action="#" method="post">
      <label for="empId">Employee ID:</label>
      <input type="text" id="empId" name="empId" required><br>

      <label for="empName">Employee Name:</label>
      <input type="text" id="empName" name="empName" 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: 40px auto;
  padding: 20px;
  background-color: #fff;
  border-radius: 10px;
  box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
}h2 {
  text-align: center;
  color: #333;
}form {
  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: #4285f4;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
}
button:hover {
  background-color: #357ae8;
}
```

Output

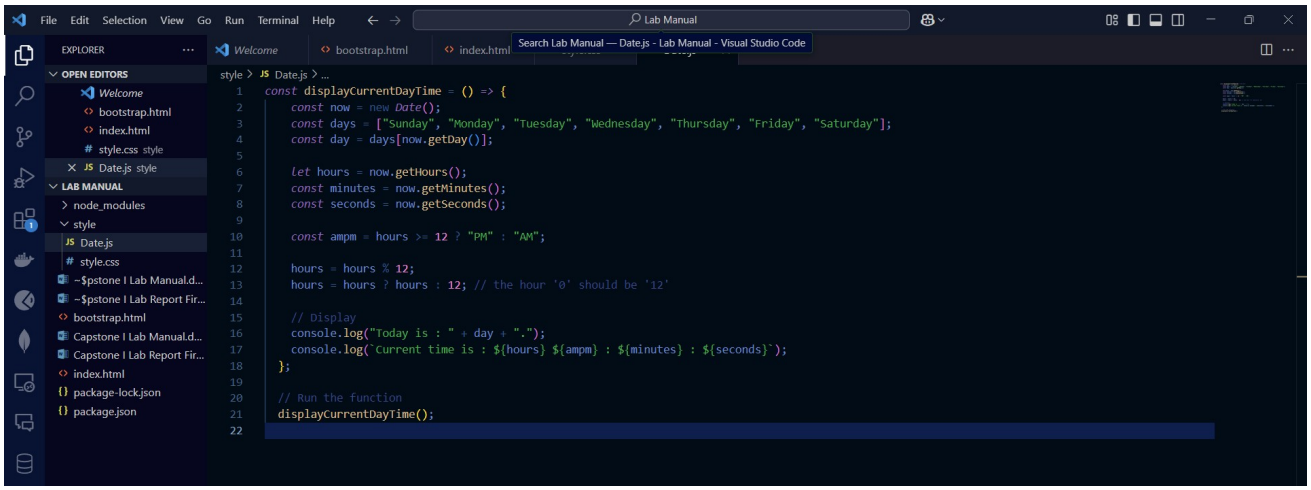
Employee Input Form

Employee ID: 1
Employee Name: A
Email: agnishwar73@gmail.com
Position: Student
Department: DS
Joining Date: 04/08/2025
Submit

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:



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:
```

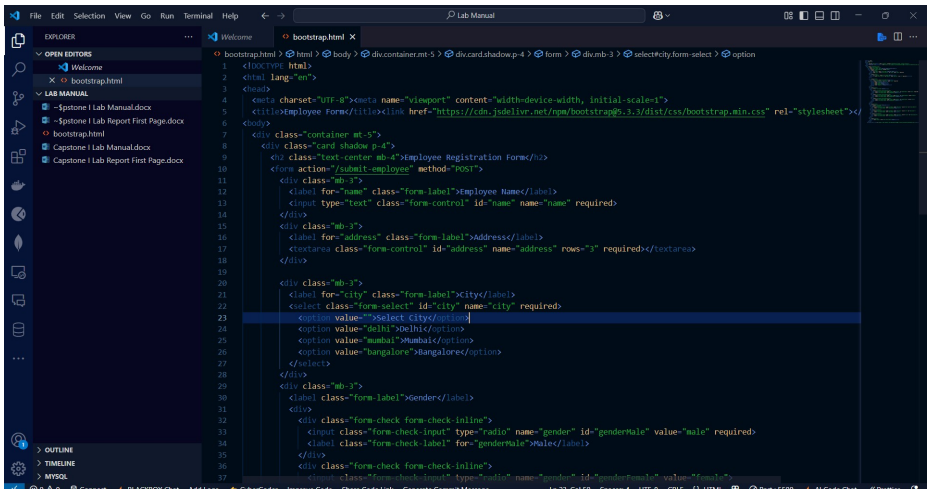
```
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual>
PS C:\Users\Augni\Desktop\NIIT\Academics\SEM VI\Capstone\Lab Manual> npm i node
```

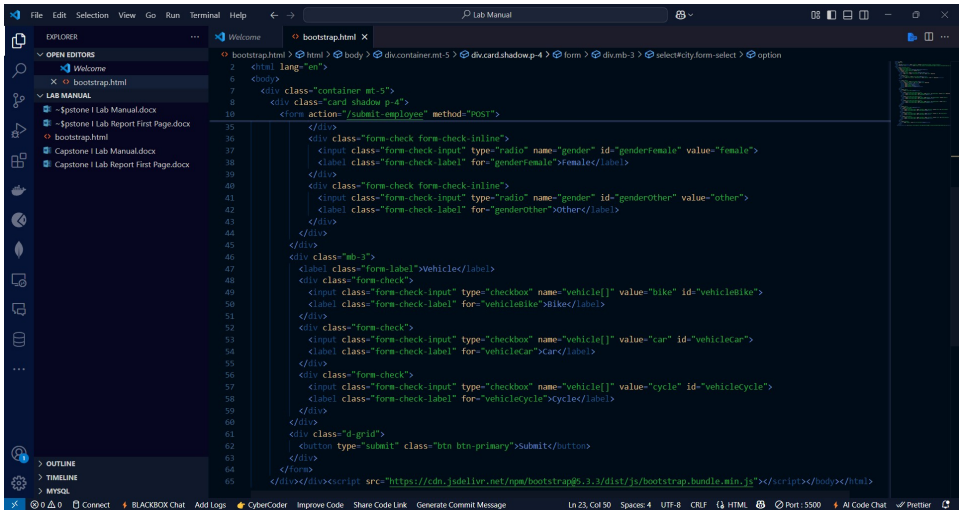
Output:

```
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>
```

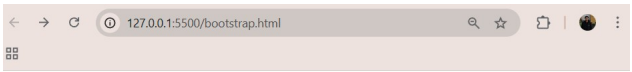
Practical 4: BootStrap :

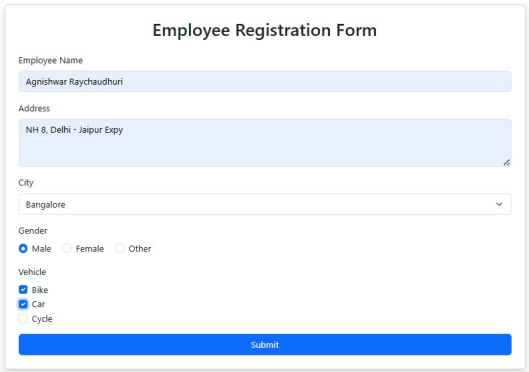
Code





Output

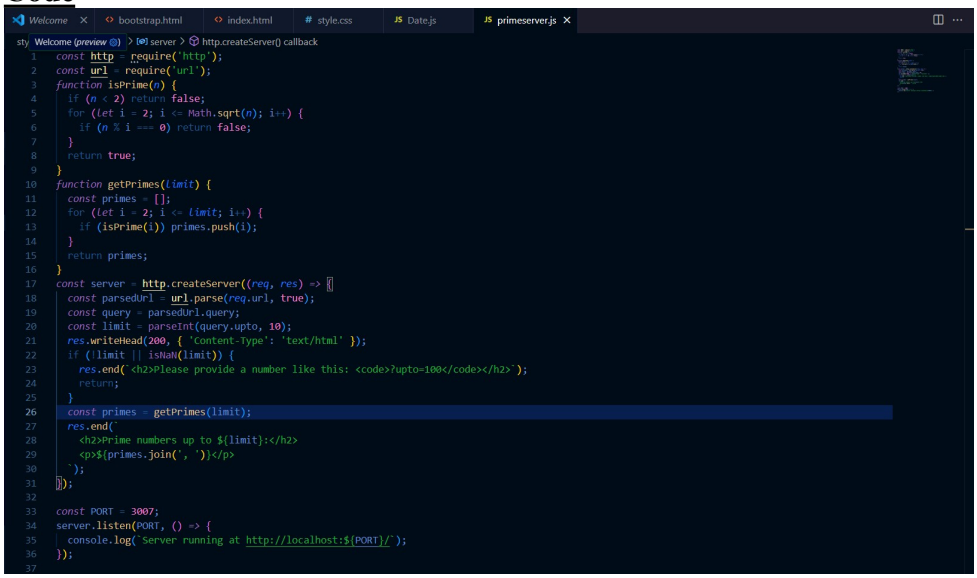




Practical 5: NodeJS :

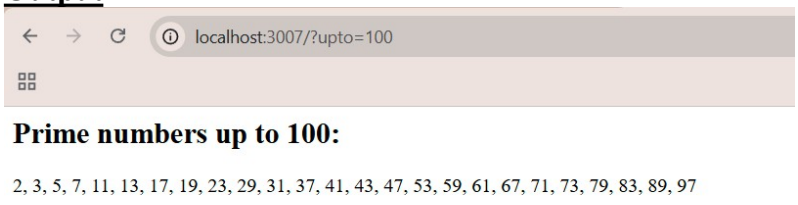
Write NodeJS code using HTTP module

Code



```
const http = require('http');
const url = require('url');
function isPrime(n) {
  if (n < 2) return false;
  for (let i = 2; i <= Math.sqrt(n); i++) {
    if (n % i === 0) return false;
  }
  return true;
}
function getPrimes(limit) {
  const primes = [];
  for (let i = 2; i <= limit; i++) {
    if (isPrime(i)) primes.push(i);
  }
  return primes;
}
const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true);
  const query = parsedUrl.query;
  const limit = parseInt(query.upto, 10);
  res.writeHead(200, { 'Content-Type': 'text/html' });
  if (!limit || isNaN(limit)) {
    res.end(`<h2>Please provide a number like this: <code>?upto=100</code></h2>`);
    return;
  }
  const primes = getPrimes(limit);
  res.end(`
    <h2>Prime numbers up to ${limit}:</h2>
    <p>${primes.join(', ')}</p>
  `);
});

const PORT = 3007;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

Output



Practical 6: ExpressJS :

Use middleware in ExpressJS

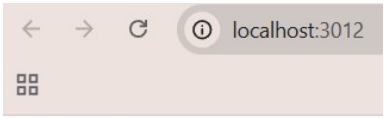
Code

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

// Custom Middleware
const loggerMiddleware = (req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next(); // Call next() to pass control to the next middleware or route handler
};

// Use the middleware
app.use(loggerMiddleware);

// Route
app.get('/', (req, res) => {
  res.send('Hello, Middleware!');
});

// Start the server
const PORT = 3012;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

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\style> 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



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 Bronxor 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 Bronxor 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 arranged 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.

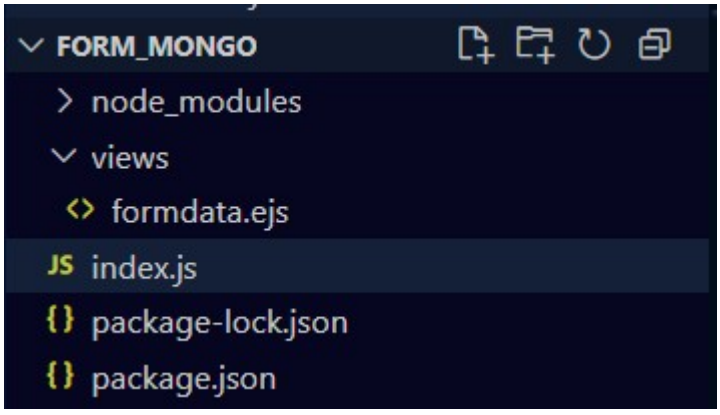
```
> db.Restaurant.aggregate([
    { $unwind: "$grades" },
    { $match: { "grades.grade": "A" } },
    {
      $group: {
        _id: "$_id",
        name: { $first: "$name" },
        borough: { $first: "$borough" },
        aCount: { $sum: 1 }
      }
    },
    { $sort: { borough: 1, aCount: -1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

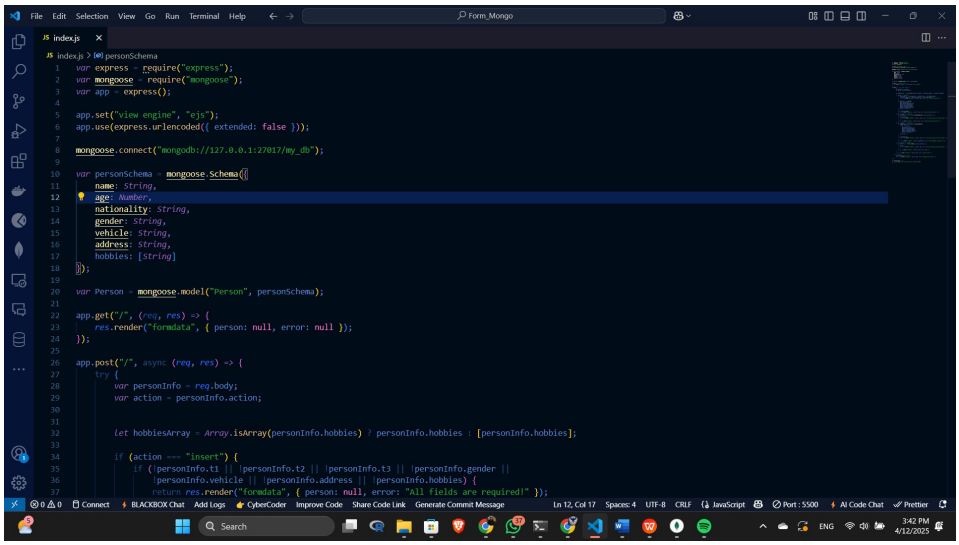
87. Write a MongoDB query to find the borough with the highest number of restaurants that have a grade of "A" and a score greater than or equal to 90.

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

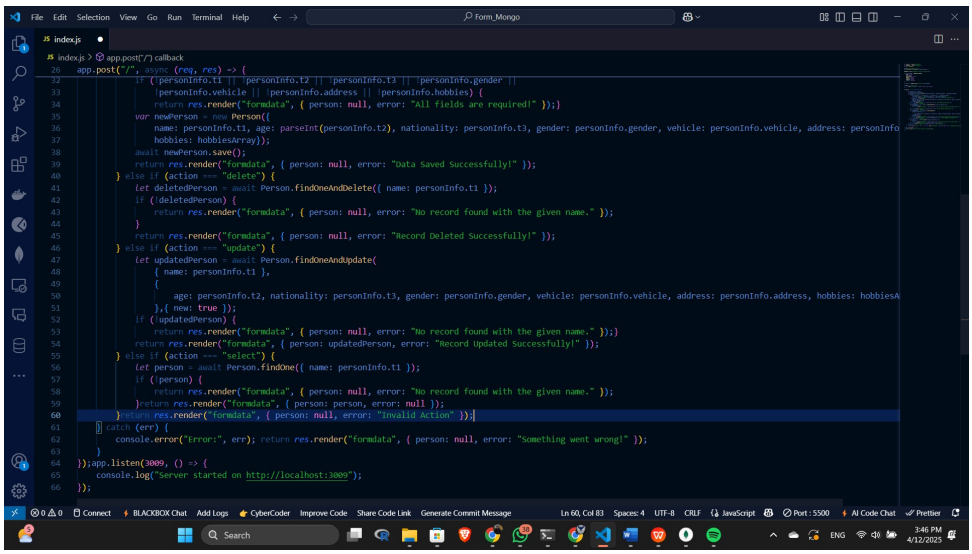
Practical 8: Database Connectivity:

Code





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

app.set("view engine", "ejs");
app.use(express.urlencoded({ extended: false }));

mongoose.connect("mongodb://127.0.0.1:27017/my_db");

var personSchema = mongoose.Schema({
    name: String,
    age: Number,
    nationality: String,
    gender: String,
    vehicle: String,
    address: String,
    hobbies: [String]
});

var Person = mongoose.model("Person", personSchema);

app.get("/", (req, res) => {
    res.render("formdata", { person: null, error: null });
});

app.post("/", async (req, res) => {
    try {
        var personInfo = req.body;
        var action = personInfo.action;


        let hobbiesArray = Array.isArray(personInfo.hobbies) ? personInfo.hobbies : [personInfo.hobbies];

        if (action === "insert") {
            if (!personInfo.t1 || !personInfo.t2 || !personInfo.t3 || !personInfo.gender ||
                !personInfo.vehicle || !personInfo.address || !personInfo.hobbies) {
                return res.render("formdata", { person: null, error: "All fields are required!" });
```



```
app.post("/", async (req, res) => {
            if (!personInfo.t1 || !personInfo.t2 || !personInfo.t3 || !personInfo.gender ||
                !personInfo.vehicle || !personInfo.address || !personInfo.hobbies) {
                return res.render("formdata", { person: null, error: "All fields are required!" });}
            var newPerson = new Person({
                name: personInfo.t1, age: parseInt(personInfo.t2), nationality: personInfo.t3, gender: personInfo.gender, vehicle: personInfo.vehicle, address: personInfo
                hobbies: hobbiesArray});
            await newPerson.save();
            return res.render("formdata", { person: null, error: "Data Saved Successfully!" });
        } else if (action === "delete") {
            let deletedPerson = await Person.findOneAndDelete({ name: personInfo.t1 });
            if (!deletedPerson) {
                return res.render("formdata", { person: null, error: "No record found with the given name." });
            }
            return res.render("formdata", { person: null, error: "Record Deleted Successfully!" });
        } else if (action === "update") {
            let updatedPerson = await Person.findOneAndUpdate(
                { name: personInfo.t1 },
                {
                    age: personInfo.t2, nationality: personInfo.t3, gender: personInfo.gender, vehicle: personInfo.vehicle, address: personInfo.address, hobbies: hobbiesA
                },{ new: true });
            if (!updatedPerson) {
                return res.render("formdata", { person: null, error: "No record found with the given name." });}
            return res.render("formdata", { person: updatedPerson, error: "Record Updated Successfully!" });
        } else if (action === "select") {
            let person = await Person.findOne({ name: personInfo.t1 });
            if (!person) {
                return res.render("formdata", { person: null, error: "No record found with the given name." });
            }return res.render("formdata", { person: person, error: null });
        }else res.render("formdata", { person: null, error: "Invalid Action" });
    } catch (err) {
        console.error("Error:", err); return res.render("formdata", { person: null, error: "Something went wrong!" });
    }
});app.listen(3009, () => {
    console.log("Server started on http://localhost:3009");
});
```

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

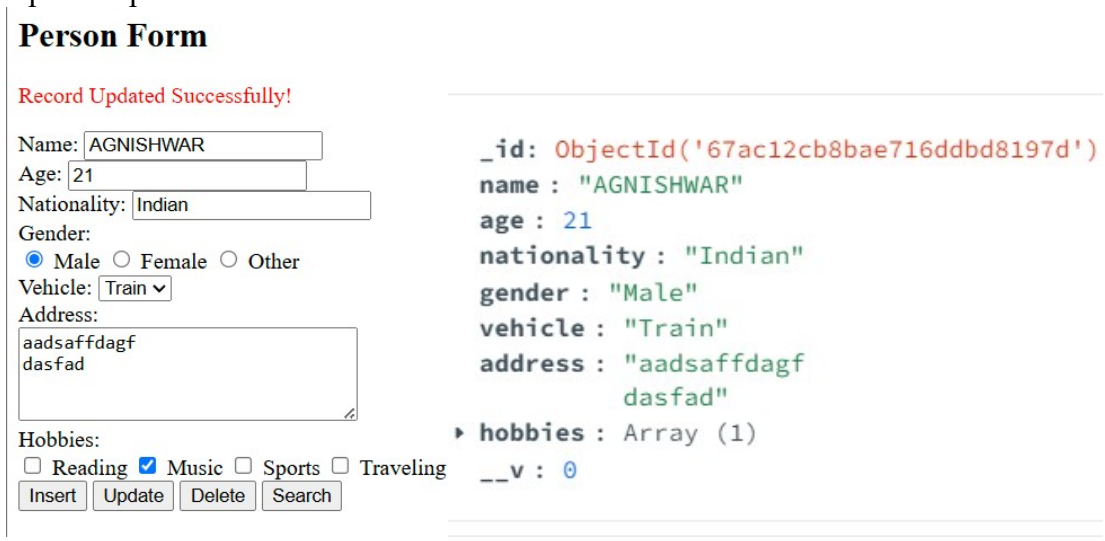
Person Form

Name:
Age:
Nationality:
Gender:
Male Female Other
Vehicle: Car
Address:
Hobbies:
Reading Music Sports Traveling
Insert Update Delete Search

Insert Operation

Search Operation

Update Operation



Delete Operation

Person Form

Record Deleted Successfully!

Name:
Age:
Nationality:
Gender:
Male Female Other
Vehicle: Car
Address:
Hobbies:
Reading Music Sports Traveling
Insert Update Delete Search

25 1 – 2 of 2

Practical 9: Session and Cookies Demo :

Session management

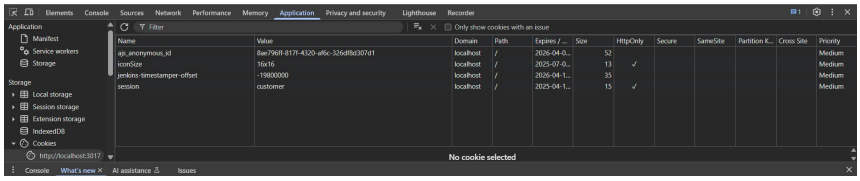
Code

```
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

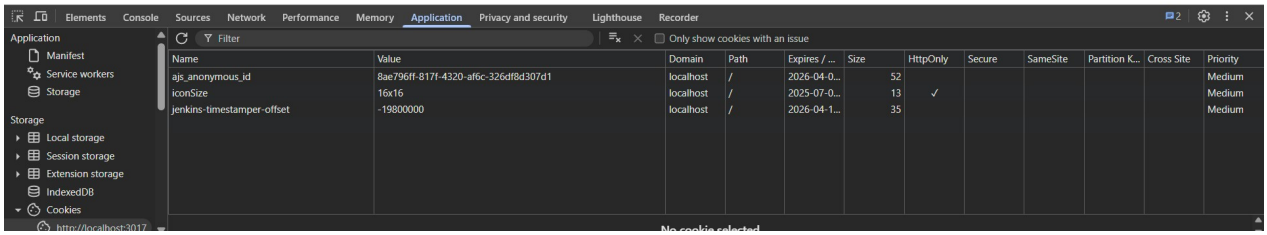
Welcome back, customer!

Logout



After 5 seconds

Username: Login



Make use of cookies for storing client data

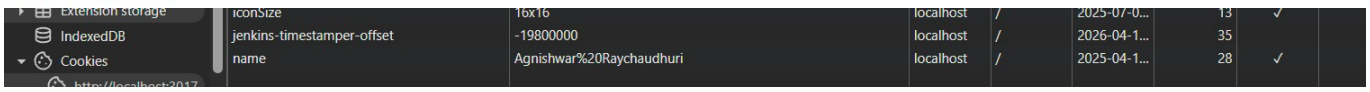
Code

```
index.js > ...
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();
app.use(cookieParser());
const app: Express coded({ extended: true }));
app.get('/', (req, res) => {
    const name = req.cookies.name;
    if (name) {
        res.send(`
            <h2>Hello, ${name}! 👋</h2>
            <a href="/clear">Clear Name</a>
        `);
    } else {
        res.send(`
            <form method="POST" action="/setname">
                <label>Enter your name: <input type="text" name="name" required></label>
                <button type="submit">Submit</button>
            </form>
        `);
    }
});
app.post('/setname', (req, res) => {
    const { name } = req.body;
    res.cookie('name', name, { maxAge: 24 * 60 * 60 * 1000, httpOnly: true });
    res.redirect('/');
});
app.get('/clear', (req, res) => {
    res.clearCookie('name');
    res.redirect('/');
});
const PORT = 3017;
app.listen(PORT, () => {
    console.log(`🚀 Server running at http://localhost:${PORT}`);
});
```

Output

Hello, Agnishwar Raychaudhuri! 👋

Clear Name



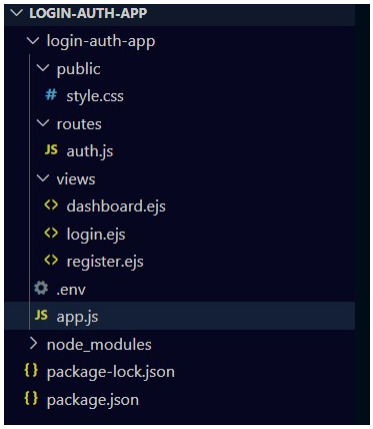
Practical 10: Login Exercise :

Install dependencies

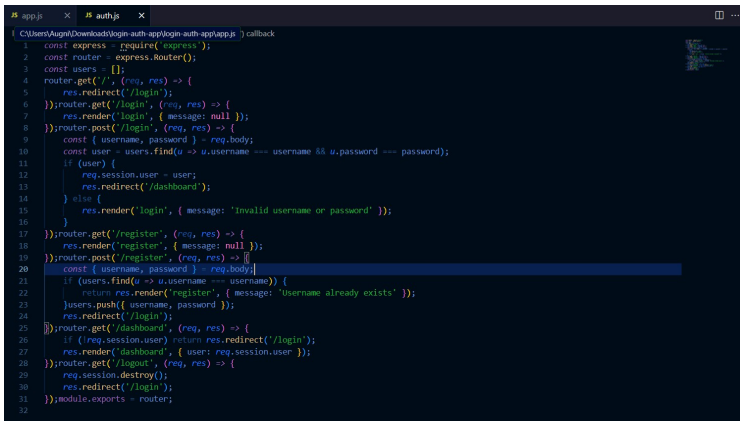
```
PS C:\Users\Augni\Downloads\login-auth-app> npm init -y
```

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

Code



```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const path = require('path');
const dotenv = require('dotenv');
dotenv.config();
const app = express();
const PORT = process.env.PORT || 4000;
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static('public'));
app.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: true
}));
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
const authRoutes = require('./routes/auth');
app.use('/', authRoutes);
app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

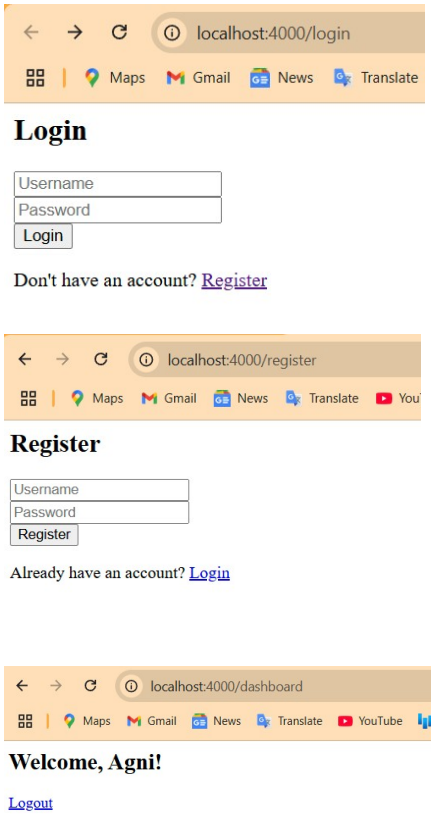


```
const express = require('express');
const router = express.Router();
const users = [];
router.get('/', (req, res) => {
    res.redirect('/login');
});router.get('/login', (req, res) => {
    res.render('login', { message: null });
});router.post('/login', (req, res) => {
    const { username, password } = req.body;
    const user = users.find(u => u.username === username && u.password === password);
    if (user) {
        req.session.user = user;
        res.redirect('/dashboard');
    } else {
        res.render('login', { message: 'Invalid username or password' });
    }
});router.get('/register', (req, res) => {
    res.render('register', { message: null });
});router.post('/register', (req, res) => {
    const { username, password } = req.body;
    if (users.find(u => u.username === username)) {
        return res.render('register', { message: 'Username already exists' });
    }users.push({ username, password });
    res.redirect('/login');
});router.get('/dashboard', (req, res) => {
    if (!req.session.user) return res.redirect('/login');
    res.render('dashboard', { user: req.session.user });
});router.get('/logout', (req, res) => {
    req.session.destroy();
    res.redirect('/login');
});module.exports = router;
```

Run the app

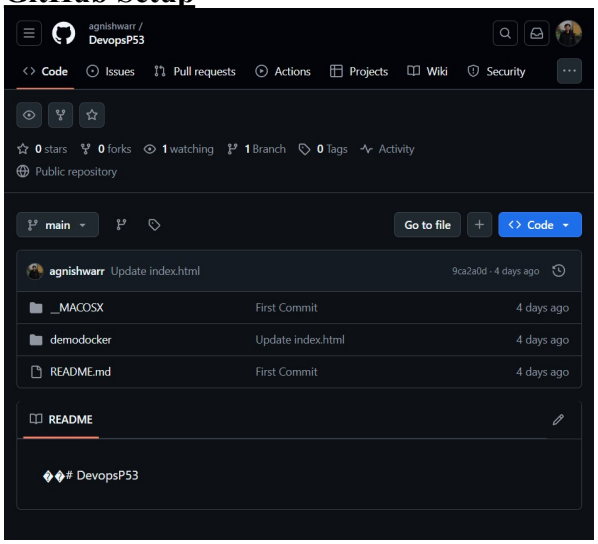
```
PS C:\Users\Augni\Downloads\login-auth-app\login-auth-app> node app.js
Server running on http://localhost:4000
```

Output

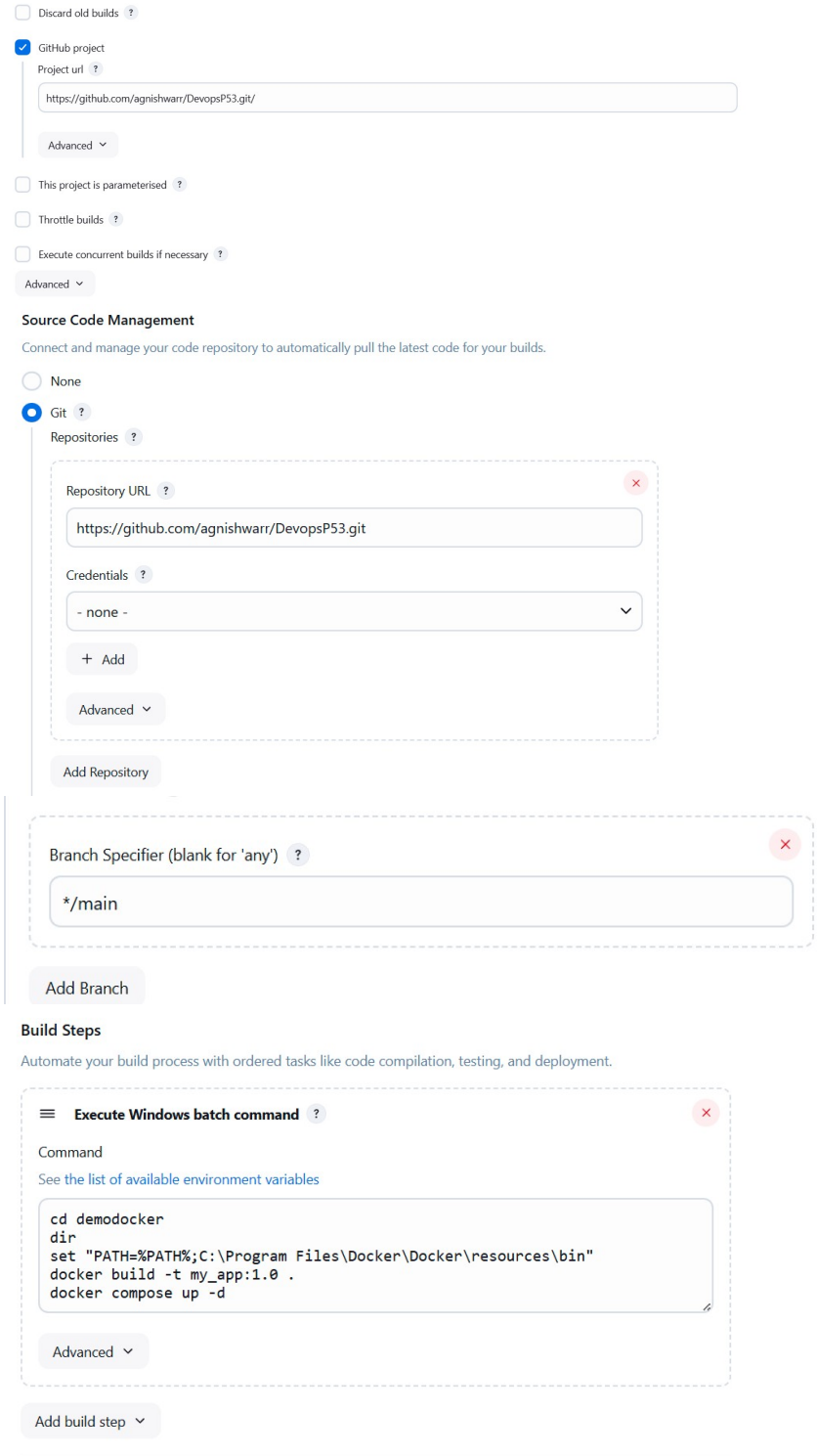


Practical 11: Setting up DevOPS for Project :

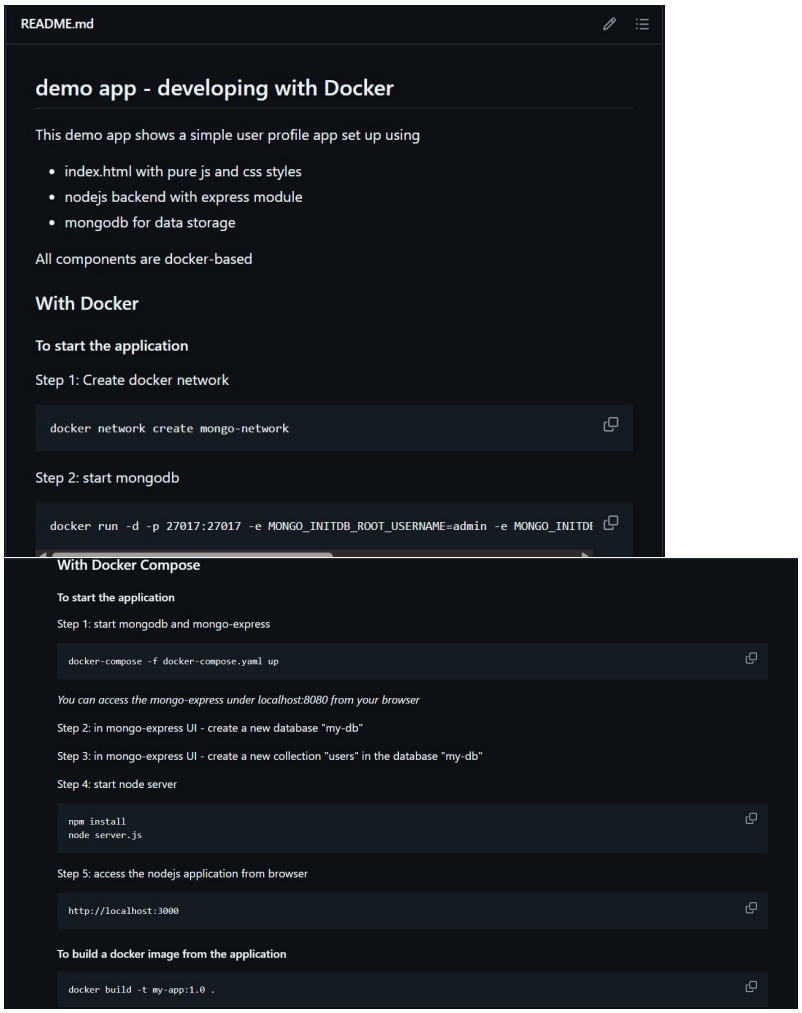
GitHub Setup



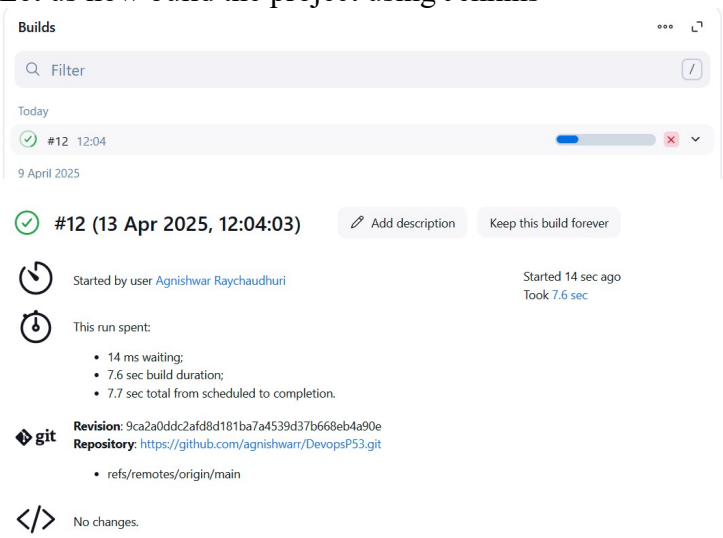
Jenkins Setup



Docker and Dockerfile



Let us now build the project using Jenkins



Output

User profile

Pet Name: **Cute Dog**

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

Interests: **coding**

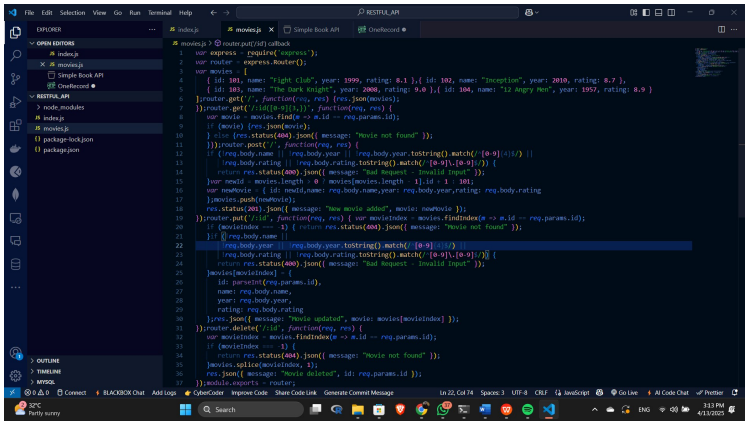
Edit

Practical 12: APIs :

Host Movie API and check it.

Code

```
index.js | movies.js | Simple Book API | OneRecord
C:\Users\Augnh\Desktop\NIIT\Academics\SEM VI\Capstone\RESTFUL_API\index.js
var express = require('express');
var bodyParser = require('body-parser');
var cookieParser = require('cookie-parser');
var multer = require('multer');
var upload = multer();
var app = express();
app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(upload.array());
var movies = require('./movies.js');
app.use('/movies', movies);
var PORT = process.env.PORT || 3004;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```



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